Freie Universität Bozen
Libera Università di Bolzano
Free University of Bozen - Bolzano

KRDB Research Centre
Knowledge Representation meets DataBases

Faculty of Computer Science, Free University of Bozen-Bolzano, Piazza Domenicani 3, 39100 Bolzano, Italy
Tel: +39 04710 16000, fax: +39 04710 16009

*KRDB Research Centre Technical Report:*

# Semantics and Validation of Recursive SHACL [Extended Version]

Julien Corman[1], Juan L. Reutter[2], Ognjen Savković[1]

| | |
|---|---|
| **Affiliation** | 1: Free University of Bozen-Bolzano, Bolzano, Italy<br>2: PUC Chile and Center for Semantic Web Research, Santiago, Chile, |
| **Corresponding author** | Julien Corman: `corman@inf.unibz.it`<br>Juan L. Reutter: `jreutter@ing.puc.cl`<br>Ognjen Savković: `ognjen.savkovic@unibz.it` |
| **Keywords** | SHACL, RDF Graph Validation, Graph Constraints,<br>RDF, Recursive Constraints |
| **Number** | KRDB18-01 |
| **Date** | Nov 6, 2018 |
| **URL** | http://www.inf.unibz.it/krdb/ |

©**KRDB Research Centre.** This work may not be copied or reproduced in whole or part for any commercial purpose. Permission to copy in whole or part without payment of fee is granted for non-profit educational and research purposes provided that all such whole or partial copies include the following: a notice that such copying is by permission of the KRDB Research Centre, Free University of Bozen-Bolzano, Italy; an acknowledgement of the authors and individual contributors to the work; all applicable portions of this copyright notice. Copying, reproducing, or republishing for any other purpose shall require a license with payment of fee to the KRDB Research Centre.

## 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 IMFD 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 design of SHACL forces us to consider cases in which these assignments are partial, or, in other words, where the truth value of a constraint at some nodes of a graph may be left unknown. Dealing with recursion also 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 constraints with 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 [4]. But RDF over time has turned into more than a simple data exchange format [2], and a key challenge for current RDF-based applications is checking quality (correctness and completeness) of a dataset. Several systems already provide facilities for RDF validation (see e.g. [12]), including commercial products.[3,4] 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,[5] which has become a W3C recommendation in 2017. SHACL groups constraints in so-called "shapes" to be verified by certain nodes of the graph under validation, and such that shapes may reference each other.

Figure 1 presents two SHACL shapes. The leftmost, named `:NIAddressShape`, is meant to define valid addresses in Northern Italy, whereas the right one, named

[3] `https://www.topquadrant.com/technology/shacl/`
[4] `https://www.stardog.com/docs/`
[5] `https://www.w3.org/TR/shacl/`

```
:NIAddressShape
  a sh:NodeShape ;
  sh:property [
    sh:path :telephone ;
    sh:maxCount 1
  ] ;
  sh:property [
    sh:path :locatedIn ;
    sh:minCount 1 ;
    sh:maxCount 1 ;
    sh:value :NorthernItaly
  ] .
```

```
:PolentoneShape
  a sh:NodeShape ;
  sh:targetClass :Polentone ;
  sh:property [
    sh:path :address ;
    sh:minCount 1 ;
    sh:maxCount 1 ;
    sh:node :NIAddressShape
  ] ;
  sh:property [
    sh:path :knows ;
    sh:node :PolentoneShape
  ] .
```

**Fig. 1.** Two SHACL shapes, about Polentoni and addresses in Northern Italy

`:PolentoneShape`, defines northern Italians, stereotypically referred to as Polentoni.[6] 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 recursive shapes 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, shapes references are a core feature of SHACL. Furthermore, in a Semantic Web context, where shapes are expected to be exchanged or reused, reference cycles 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

[6] This example is borrowed from Peter Patel-Schneider: `https://research.nuance.com/wp-content/uploads/2017/03/shacl.pdf`

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 in a satisfactory way validation of shapes with so-called *non-stratified* constraints, where negation is used arbitrarily in reference cycles.

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 [9]. In addition, an appendix with detailed proofs and a translation from SHACL into our abstract syntax and conversely can be found at [8].

## 2 Validating a graph against SHACL shapes

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

Checking whether a graph $G$ is valid against a set $S$ of shapes may be viewed as a two-step process. The first step consists in iterating over all shapes $s \in S$, and retrieve their respective target nodes in $G$. SHACL provides a dedicated language to describe

```
: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
  ] .
```

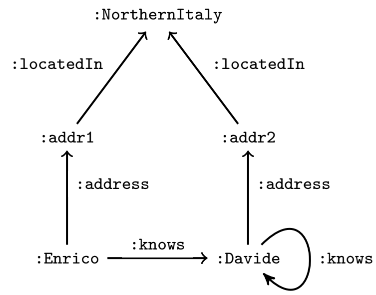


**Fig. 2.** A SHACL shapes for semi-Polentone, and a graph $G$ to be validated against this shape, together with the shapes of Figure 1

the intended targets of a shape (e.g. the `sh:targetClass` property in Figure 1), which is orthogonal to the language used to define constraints. Furthermore, this language has a limited expressivity, allowing all targets of shape $s$ in $G$ to be retrieved in $O(|G| \cdot \log |G|)$, before constraint validation.

The second step consists in iterating over each target node $v$ of each shape $s$, and check whether the node $v$ satisfies $s$. This check can be represented as a call to a recursive function *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 *validates*$(s, G, v)$ may also trigger a recursive call *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 node of $s'$. In turn, *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 `SemiPolentoneShape` in Figure 2 uses `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, *validates*$(\texttt{SemiPolentoneShape}, G, v)$ will succeed only if some successor of $v$ via property `:knows` *violates* the constraints for `:PolentoneShape`.[7]

### 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 `:SemiPolentoneShape sh:targetNode :Enrico` indicates that `:Enrico` is the unique target of shape `:SemiPolentoneShape`. This is also the only target to be validated in the graph.

To check if :Enrico validates `:SemiPolentoneShape`, the validation process described in the specification would call *validates*$(\texttt{SemiPolentoneShape}, G, \texttt{:Enrico})$, triggering an infinite sequence of recursive calls to *validates*$(\texttt{PolentoneShape}, G, \texttt{:Davide})$. Intuitively, the problems is that *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`

[7] Constraints on node sucessors in SHACL are by default universally quantified. This is why `sh:not` here requires one successor violating `:PolentoneShape` to exist.

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

```
:NaivePolentoneShape
  a sh:NodeShape ;
  sh:not [
    sh:path :knows ;
    sh:node :
       NaivePolentoneShape
  ] .
```

**Fig. 3.** Two SHACL shapes which illustrates the need for partial assignments

can be assigned `:SemiPolentoneShape`, and `:Davide` can be assigned the negation of `:PolentoneShape`. This assignment complies with the constraints and 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 does not comply with the target, therefore it would not allow us to validate the graph.

Several formal frameworks dealing with recursion (such as recursive Datalog[10]) have semantics based on a similar intuition. This notion of assignment is also used in [7] for ShEx, a constraint language for RDF very similar to SHACL. However, the semantics proposed in [7] would consider the graph of Figure 2 as invalid, taking only one assignment into consideration, where `:Davide` is assigned `:PolentoneShape`, and therefore `:Enrico` cannot verify `:SemiPolentoneShape`. The semantics defined in [7] is also restricted to *stratified* constraints, i.e. constraints such that reference cycles have no reference in the scope of a negation (see Definition 8 further below).

### 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 two shapes of Figure 3.

`:Davide` is the only target node, for shape `:HappyPersonShape`. This shape is validated iff `:Davide` has an address, or knows a naive polentone. Because `:Davide` has an address, a simple call to *validates*(`HappyPersonShape`, $G$, `:Davide`) would validate the graph. But a total assignment must also assign either `:NaivePolentoneShape` or its negation to `:Davide`. And this cannot be done in a consistent manner. If `:NaivePolentoneShape` is assigned, then `:Davide` does not verify the corresponding constraint; if the negation of `:NaivePolentoneShape` is assigned, then `:Davide` does not violate the constraint. Therefore a semantics based on total assignments would consider the graph invalid.

It should be emphasized that this example is not a limit case: the same problem appears for any (satisfiable) set of shapes containing a reference cycle (of any size), and such that an odd number of references in this cycle are in the scope of a negation. Therefore, if one wants to defines 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 Formal semantics for SHACL

This section provides a formal semantics for recursive SHACL. As explained above, constraint validation is based on *partial* assignment. This semantics *(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 set of shapes is validated iff *there exists* an assignment (called here *faithful*) complying with it. This is a key difference from query answering, or cautious reasoning in Datalog, interested in *certain answers*, i.e. holding for *all* valid assignments. For instance, in Figure 2, some faithful assignments assign `:PolentoneShape` to `:Davide`, and some do not.

### 3.1 Notation

Like the SHACL specification, we borrow from SPARQL the notion of *property path*, which describes regular constraints holding over a path in a graph (for the syntax and semantics, we defer to the SPARQL standard [15]). Following [16], if $r$ is a property path and $G$ a graph, we denote with $r(G)$ the evaluation of $r$, which consists of all pairs $(v, v')$ of nodes in $G$ such that there is a path from $v$ to $v'$ satisfying $r$.

Similarly, if $\psi$ is a SPARQL query, we denote with $\psi(G)$ the evaluation of $\psi$ in $G$. Finally, we use $|X|$ to denote the size of structure $X$.

### 3.2 Abstract Syntax and Semantics for SHACL constraints

**Syntax.** As usual, we find more convenient to work with a logical abstraction of the concrete SHACL language. Our abstraction uses a fragment of first order logic to simulate node shapes, and then unravels so-called SHACL "property shapes" as modal formulas over nodes. Like the SHACL specification, we make the unique name assumption, i.e. we assume that two blak nodes in an RDF graph cannot denote the same individual. We also abstract away from constraints on IRIs and literals (regular expresion, datatype, value comparison, etc.), and use a simple constant $I$ instead. Constraints are defined by the following grammar:

$$\phi \ ::= \ \top \ | \ s \ | \ I \ | \ \phi_1 \wedge \phi_2 \ | \ \neg\phi \ | \ \geq_n r.\phi \ | \ \mathrm{EQ}(r_1, r_2)$$

where $s$ is a shape name, $I$ is an IRI, $r$ is a property path, and $n \in \mathbb{N}^+$. As syntactic sugar, we use $\leq_n r.\phi$ for $\neg(\geq_{n+1} r.\phi)$, and $=_n r.\phi$ for $(\geq_n r.\phi) \wedge (\leq_n r.\phi)$.

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 [8] 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 the syntax with the example from Figure 1. To express SHACL cardinality constraints (e.g. `sh:maxCount`), we use $\leq_1 r.\phi$, which means that a node can have at most 1 $r$-successor satisfying $\phi$, or $=_1\ r.\phi$ for exactly one. Then the constraints for `:NIAddressShape` (abbreviated here as $s_{\texttt{niaddr}}$) can be translated as:

$$(\leq_1 \texttt{telephone}.\top) \wedge (=_1 \texttt{locatedIn}.\texttt{NorthernItaly})$$

where $\top$ is true at every node. In the same way, we can translate the constraints for `:PolentoneShape` (abbreviated here as $s_{\texttt{pol}}$). Both $s_{\texttt{niaddr}}$ and $s_{\texttt{pol}}$ appear in the constraint for $s_{\texttt{pol}}$. This mimics the SHACL syntax, where both shapes were mentioned:

$$(\leq_0 \texttt{knows}.\neg s_{\texttt{pol}}) \wedge (=_1 \texttt{address}.s_{\texttt{niaddr}})$$

**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 at each node.

**Definition 1 (Assignment).** *Let $N$ be a set of shape names, and $G$ a graph. An* assignment *$\sigma$ for $G$ and $N$ is a total function mapping nodes in $G$ to subsets of $N \cup \{\neg s \mid s \in N\}$, such that $s$ and $\neg s$ cannot be both in $\sigma(v)$*

**Definition 2 (Total assignment).** *A assignment $\sigma$ for $G$ and $N$ is* total *if either $s \in \sigma(v)$ or $\neg s \in \sigma(v)$, for each node in $G$ and $s \in N$*

The evaluation $[\![\phi]\!]^{v,G,\sigma}$ of formula $\phi$ at node $v$ in graph $G$ given $\sigma$ is defined in Table 1. In order to evaluate a formula given a partial assignment, we use a 3-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 be total, then this third value is not needed:

**Observation 1** *Let $\sigma$ be a total assignment for $G$ and $N$, and $\phi$ a constraint formula using shape names in $N$. 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 shapes to $\mathcal{L}$ constraints**. We model a 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 retrieving the target nodes of $s$. If $S$ is a set of shapes, we assume that for each $(s, \phi_s, \text{target}_s) \in S$, if $s'$ appears in $\phi_s$, then $(s', \phi'_s, \text{target}'_s) \in S$. An assignement for $G$ and $S$ is an assignment for $G$ and $\{s \mid (s, \phi_s, \text{target}_s) \in S\}$. Abusing notation, we write "$s \in S$" instead of "$(s, \phi_s, \text{target}_s) \in S$".

### 3.3 Validation

We finally have all components in place to define graph validation. Intuitively, a graph is valid against a set $S$ of shapes if one can find an assignment $\sigma$ for $G$ and $S$ complying with targets and constraints. We call such an assignment *faithful*, defined as follows:

**Definition 3 (Faithful Assignment).** *A assignment $\sigma$ for $G$ and $S$ is* faithful *iff $\text{target}_s(G) \subseteq \sigma(v)$ for each $(s, \phi_s, \text{target}_s) \in S$, and, for each node $v$ in $G$:*

$$
\begin{aligned}
[\![\top]\!]^{v,G,\sigma} &= 1 \\
[\![\neg\phi]\!]^{v,G,\sigma} &= 1 - [\![\phi]\!]^{v,G,\sigma} \\
[\![\phi_1 \wedge \phi_2]\!]^{v,G,\sigma} &= \min\{[\![\phi_1]\!]^{v,G,\sigma}, [\![\phi_2]\!]^{v,G,\sigma}\} \\
[\![(r_1 = r_2)]\!]^{v,\sigma,G} &= \begin{cases} 1, \text{ if } \{v' \mid (v,v') \in r_1(G)\} = \{v' \mid (v,v') \in r_2(G)\} \\ 0 \text{ otherwise} \end{cases} \\
[\![I]\!]^{v,\sigma,G} &= \begin{cases} 1, \text{ if } v \text{ is the IRI I,} \\ 0 \text{ otherwise} \end{cases} \\
[\![s]\!]^{v,G,\sigma} &= \begin{cases} 1, \text{ if } s \in \sigma(v) \\ 0, \text{ if } \neg s \in \sigma(v) \\ 0.5 \text{ otherwise} \end{cases} \\
[\![\geq_n r.\phi]\!]^{v,\sigma,G} &= \begin{cases} 1, \text{ if } |\{v' \mid (v,v') \in r(G) \text{ and } [\![\phi]\!]^{v'G,\sigma} = 1\}| \geq n \\ 0, \text{ if } |\{v' \mid (v,v') \in r(G)\}| - \\ \qquad |\{v' \mid (v,v') \in r(G) \text{ and } [\![\phi]\!]^{v'G,\sigma} = 0\}| < n \\ 0.5 \text{ otherwise} \end{cases}
\end{aligned}
$$

**Table 1.** Inductive evaluation of constraint formula $\phi$ at node $v$ in graph $G$ given assignment $\sigma$

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

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

The (online) appendix provides a full translation from SHACL to sets of shapes and conversely, which preserves validation, provided the shapes are non-recursive (i.e. contain no reference cycle). Our notion of validation is more robust though, as it is also well-defined for recursive shapes. In Section 4, we study the complexity of the validation problem. But for now, we provide some insight on properties of this semantics.

### 3.4 Properties of Validation

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

**Definition 5 (Immediate evaluation operator T).**
$\mathbf{T} : \Sigma^{G,S} \to \Sigma^{G,S}$ *is the function defined by*
$s \in (\mathbf{T}(\sigma))(v)$ *iff* $[\![\phi_s]\!]^{v,G,\sigma} = 1$*, and* $\neg s \in (\mathbf{T}(\sigma))(v)$ *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 $G$*

**Validation without target**. The SHACL specification states that a graph $G$ is valid against a set $S$ of shapes if no shape in $s$ has target in $G$. From Definitions 3 and 4, this also (trivially) holds in the recursive case for our semantics. Somehow surprisingly, validation without target may fail for *total* assignments. For instance, there is no total faithful assignment for the graph of Figure 2 and the set of shapes containing only shape `:NaivePolentoneShape` from Figure 3.

**A stricter notion of faithfulness**. From Definition 3, a faithful assignment $\sigma$ is only required to assign $s$ to a node $v$ if $\phi_s$ is verified by $v$ (given $\sigma$), and to assign $\neg s$ to $v$ if $\phi_s$ is violated by $v$ (given $\sigma$). But it is also possible to assign none of these two, even though $v$ verifies of violates $\phi_s$ (given $\sigma$). This may seem counterintuitive, which leads to the following stricter notion of faithfulness:

**Definition 7 (Strictly-faithful assignment).** *A assignment $\sigma$ for $G$ and $S$ is* strictly faithful *iff $\mathit{target}_s(G) \subseteq \sigma(v)$ for each $(s, \phi_s, \mathit{target}_s) \in S$, and, for each node $v$ in $G$:*
- *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 $G$ is* strictly valid *against a set of shapes $S$ if there is a strictly faithful assignment for $G$ and $S$.*

For instance, there is only one strictly faithful assignment for the graph of in Figure 2 and the two shapes of Figure 3. It assigns $\neg$`:HappyPersonShape` to `:addr1`, because `:addr1` violates the constraint for this shape. There are also several (non-strictly) faithful assignments, some of which assign neither `:HappyPersonShape` nor its negation to `:addr1`. So intuitively, non-strict validation allows some form of "lazy" constraint evaluation.

The operator $\mathbf{T}$ provides a more concise definition. Both faithful and strictly faithful assignments must comply with targets for $G$ and $S$. But in addition, a faithful assignment $\sigma$ must verify $\sigma \preceq \mathbf{T}(\sigma)$, whereas a strictly faithful assignment $\sigma'$ must verify $\sigma' = \mathbf{T}(\sigma')$.

Interestingly, these two notions of validation coincide. To prove this, we first need a useful property, the monotonicity of $\mathbf{T}$ 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 $\mathbf{T}(\sigma_1) \preceq \mathbf{T}(\sigma_2)$*

We can now state the equivalence:

**Proposition 1.** *For any $G$ and $S$, $G$ is valid against $S$ iff $G$ is strictly valid against $S$*

*Proof (Sketch).* The right direction is trivial, because a strictly faithful assignment is faithful. 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 $\sigma_0$, i.e. $\sigma' \in \Sigma'$ iff $\sigma_0 \preceq \sigma'$. From Lemma 1, $\mathbf{T}(\sigma_0) \preceq \mathbf{T}(\sigma')$. And because $\sigma_0$ is faithful, $\sigma_0 \preceq \mathbf{T}(\sigma)$. Therefore $\sigma_0 \preceq \mathbf{T}(\sigma')$, i.e. $\mathbf{T}(\sigma') \in \Sigma'$.

Now consider the (meet) semi-lattice $\langle \Sigma', \preceq \rangle$ rooted in $\sigma_0$. We just showed that for each $\sigma' \in \Sigma'$, $\mathbf{T}(\sigma') \in \Sigma'$. In addition, from Lemma 1, $\mathbf{T}$ is monotone over $\langle \Sigma', \preceq \rangle$. So from a (weaker version of) the Knaster-Tarski Theorem, $\mathbf{T}$ admits 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 strictly faithful for $G$ and $S$. □

**All we need is one target**. The following explains why the complexity results provided in Section 4 only consider graph validation with a single target node.

**Proposition 2.** *Given a graph $G$, set $S$ of shapes and target nodes in $G$ for each $s \in S$, one can construct in linear time a graph $G'$ and set $S'$ of shapes, such that $G$ is valid against $S$ iff $G'$ is valid 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 $v_1^1, .., v_1^{m1}, ..,$ $v_n^1, .., v_n^{mn}$. Extend $G$ with a fresh node $v_0$, and an edge $(v_o, e_i^j, v_i^j)$ for each $v_i^j$, with $e_i^j$ a fresh edge label. Then delete all target expressions in $S$, and extend $S$ with a fresh shape $s_0$, with target node $v_0$, and constraint $\phi_{s_0} \doteq (\geq_1 e_1^{m1}.\top) \wedge .... \wedge (\geq_1 e_1^{mn}.\top)$. □

### 3.5 Validation and Stratified Negation

Section 2.2 suggested that the need for partial assignments comes from constraints combining circular references with negation, called *non-stratified*. We now make this intuition more precise, showing that we can indeed focus solely on total assignments if the constraints are stratified.

To formalize this idea, we borrow the notion of stratification from Datalog[10] (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 a total function* $\mathrm{str} : S \to \mathbb{N}$ *such that:*
- *If $s_1$ appears in $\phi_{s_2}$, then* $\mathrm{str}(s_1) \leq \mathrm{str}(s_2)$
- *If $s_1$ appears in $\phi_{s_2}$ in the scope of a negation then* $\mathrm{str}(s_1) < \mathrm{str}(s_2)$.

It must be emphasized that the language $\mathcal{L}$ does not include $\leq_n r$ or $=_n r$. If these operators were included, then one would need to redefine the second condition accordingly, as $\leq_n r$ is a form of negation.

The following result confirms that a semantics based on total assignment is sufficient for stratified sets of shapes.

**Proposition 3.** *Let $S$ be a stratified set of shapes 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 represent assignments as sets of positive and negative atoms. Let $\sigma$ be a faithful assignment for $G$ and $S$, and let $S_1, .., S_n$ 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 $\mathbf{T}$ 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 $S_0, .., S_i$. 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 also implies that 3-valued validation is not easier than 2-valued validation, which may come as a surprise.

## 4 Complexity

We now study the computational complexity of the validation problem, defined as follows (full proofs are provided in the online appendix):

VALIDATION:
**Input**: Graph $G$, set $S$ of shapes
**Decide**: $G$ is valid against $S$

Based on Proposition 2, we focused on instances with one target node (for one shape in $S$). We also assume that this target node is already known. 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 VALIDATION($S$) and VALIDATION($G$), for a fixed set $S$ of shapes and fixed graph $G$, called *data complexity* and *constraint complexity* below.

We consider two fragments of the constraint language $\mathcal{L}$: *(i)* $\mathcal{L}_{\geq_1,\neg,\wedge}$ is the fragment defined by the grammar $\phi ::= \top \mid I \mid s \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \geq_1 p.\phi$, where $p$ is an IRI, and *(ii)* $\mathcal{L}_{\geq_n,\wedge,\vee,r,\mathrm{EQ}}$ is the fragment defined with $\phi ::= \top \mid I \mid s \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \geq_n r.\phi \mid \mathrm{EQ}(r_1, r_2)$ , where $r, r_1, r_2$ are property paths and $\phi_1 \vee \phi_2$ is interpreted (as expected) as $\neg(\neg\phi_1 \wedge \neg\phi_2)$.

| **Fragment** | *Data* | *Constraint* | *Combined* |
|---|---|---|---|
| $\mathcal{L}$ (= SHACL) | NP-c | NP-c | NP-c |
| stratified $\mathcal{L}_{\geq_1,\neg,\wedge}$ | NP-c | NP-c | NP-c |
| $\mathcal{L}_{\geq_n,\wedge,\vee,r,\mathrm{EQ}}$ | in P | in P | P-c |

**Table 2.** Computational complexity of VALIDATION. -c stands for *complete*.

We start by showing an NP upper bound for combined complexity, based on guessing a witnessing faithful assignment. Then we show that this upper bound is tight, even for a fixed set of shapes (data complexity) using stratified negation and basic operators ($\geq_1, \neg$ and $\wedge$). We also show that this bound is tight for a fixed graph. Lastly, we show that allowing disjunction but disallowing negation otherwise is sufficient to regain tractability.

Let us start with NP membership. First, all property paths present in $S$ can be materialized in time polynomial in $|G| \cdot |S|$ before validation. In addition, by introducing fresh shape names, $S$ can be transformed in polynomial time into an equivalent set $S'$ of shapes, whose constraints contain at most one operator. Then assuming that we can guess a faithful assignment $\sigma$ for $G$ and $S'$, we only to check $\sigma$ is indeed faithful. To do so, it is sufficient to compute the value of $[\![\phi_s]\!]^{v,G,\sigma}$ for each node $v$ in $G$ and $s \in S'$, which is again polynomial in $|G| + |S|$, even with a binary encoding of cardinality constraints. Summing up, we have:

**Proposition 4 (Combined – Upper Bound).** VALIDATION *is in* NP.

Now for the lower bound, validation is already intractable in data complexity for stratified $\mathcal{L}_{\geq_1,\neg,\wedge}$. This may come as a surprise, considering that data complexity of ground fact entailment in stratified Datalog is in PTIME [10]. We show NP-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 (in linear time) into a graph, and this graph is valid against $S$ iff the circuit is satisfiable.

**Proposition 5 (Data – Lower Bound).** *There is a stratified fixed set $S$ of shapes in $\mathcal{L}_{\geq_1,\neg,\wedge}$ such that* VALIDATION*($S$) is* NP-hard.

We also show that the problem is NP-hard in constraint complexity for the same fragment (with a reduction from SAT):

**Proposition 6 (Constraint – Lower Bound).** *There is a fixed graph $G$ such that* VALIDATION*($G$) is* NP-hard*, even if $S$ is restricted to stratified sets of shapes in $\mathcal{L}_{\geq_1,\neg,\wedge}$.*

As a more optimistic result, validation is in PTIME if one allows disjunction as a native operator, but disallows negation otherwise. The proof relies on the (unique) minimal fixed-point $\sigma$ of $\mathbf{T}$ w.r.t. $\preceq$, which can be computed in time polynomial in $|G|+|S|$. Let $v_0$ be the (unique) target node to validate, against shape $s_0$. If $\neg s_0 \in \sigma(v_0)$, then $G$ is invalid. Otherwise, it can be shown that there must be an extension of $\sigma$ (w.r.t. $\preceq$) which is faithful for $G$ and $S$.

**Proposition 7 (Combined – Upper Bound).** VALIDATION *is in P for $\mathcal{L}_{\geq_n,\wedge,\vee,r,EQ}$.*

Finally, we show PTIME hardness for a sub-fragment of $\mathcal{L}_{\geq_n,\wedge,\vee,r,\mathrm{EQ}}$ (without property paths and path equality), with a log-space reduction from the problem of evaluating a monotone boolean circuit.

**Proposition 8 (Combined – Lower Bound).** VALIDATION *is* P-hard *for $\mathcal{L}_{\geq_n,\wedge,\vee,r,EQ}$.*

## 5 Approximation

The above intractability result for data complexity (Proposition 6), 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 $G$ is valid against a set $S$ of shapes, with an integer parameter $k$. If $k$ is bounded, then the algorithm is sound, and runs in time polynomial in $|G|$. If $k$ 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 `Valid` (resp. `Invalid`) only if $G$ is valid (resp. not valid) against $S$.

For readability, from Proposition 2, 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 intuitively computes an assignment $\sigma_{\text{minFix}}$ matching all constraints enforced by the graph, regardless of the target. If the validity of $G$ cannot be decided after this (polynomial) step, then $\sigma_{\text{minFix}}$ is extended by assigning $s_0$ to $v_0$, and an attempt is made to propagate constraints from $v_0$ to its successors, in order for $v_0$ to satisfy $\phi_{s_0}$.

**Step 1: minimal fixed-point**. As a reminder from Section 3.3, we use $\Sigma^{G,S}$ to denote the set of all (possibly partial) assignments for $G$ and $S$. The first step of the algorithm computes the minimal fixed-point $\sigma_{\text{minFix}}$ of the operator $\mathbf{T}$ (see Definition 5) w.r.t. $\preceq$. Because $\langle \Sigma^{G,S}, \preceq \rangle$ is a semi-lattice and $\mathbf{T}$ is monotone w.r.t. $\preceq$ (Lemma 1), $\sigma_{\text{minFix}}$ must exist and be unique. It can also be computed in time polynomial in $|G|$, initializing $\sigma_{\text{minFix}}$ with the empty set, and then applying $\mathbf{T}$ to $\sigma_{\text{minFix}}$ recursively, until a fixed-point is reached. This is performed by procedure 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 $\mathbf{T}$ (see Section 3.3), and therefore must extend $\sigma_{\text{minFix}}$. So from Proposition 1, If $\neg s_0 \in \sigma_{\text{minFix}}(v_0)$, 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 abuse notation and use set operators ($\cup, \in$, etc.) to describe the stack. Similarly, for brevity, 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 $\neg s(v) \notin \sigma$,
- $n$ is the current depth of the exploration, incremented each time $\sigma$ is extended. When $n$ reaches $k$, the size of the stack cannot be extended anymore, which triggers a call to REDUCE, line 11, to merge some of the current branches.

Line 8, function EXTEND computes each minimal extensions $\sigma'$ of $\sigma$ such that:

- If $s \in \sigma^P(v)$, then $[\![\phi_s]\!]^{v,G,\sigma'} = 1$,
- If $\neg s \sigma^P(v)$, then $[\![\phi_s]\!]^{v,G,\sigma'} = 0$, and
- if $s(v) \in A$, then $\{s, \neg s\} \cap \sigma(v) = \emptyset$

It can be shown that each call to EXTEND can be executed in time $O(|G|^{|S|})$.

Finally, if the depth $n$ of the exploration reaches $k$, line 11, then procedure REDUCE prevents the number of elements in the stack to increase. Line 18, function GETCLOSESTPAIR retrieves the two closest assignments $\sigma_1$ and $\sigma_2$ (in terms of edit distance) in the Stack. Then function 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 $\neg s$ are in $\sigma_1(v) \cup \sigma_2(v)$, and the procedure REPLACE sets each $\sigma_i$ to $\sigma_i \setminus \{s(v), \neg s(v)\}$. After this step, either $\sigma_1 \preceq \sigma_2$ or $\sigma_2 \preceq \sigma_1$ must hold, and only the greater of the two (w.r.t $\preceq$) is retained (Line 23) and pushed in the stack.

The number of possible assignments is of $O(2^{|G|})$, but the number of assignments created by EXTEND is $O(|G|^{|S|})$. So if the parameter $k$ is fixed, the reduced stack makes sure that the execution time is $O(|G|^{|S|.k})$.

## 6 Related Work

Several schema languages have been proposed or implemented for RDF before SHACL, and some of them are closely associated to the design 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, but is primarily used to infer implicit facts.

Among the proposals which do not relate (to our knowledge) to the genesis of SHACL, are proposals for RDF integrity constraints [1, 13]. We have not explored a formal comparison between these formalisms and SHACL, but conjecture that they are incomparable with SHACL.

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.

[8] `http://spinrdf.org/`

**Algorithm 1** APPROXIMATION

**Require:** $G', S, s_0, v_0, k$
$\sigma_{\text{minFix}} \leftarrow$ COMPUTEMINFIX$(G', S)$
**if** $s_0 \in \sigma_{\text{minFix}}(v_0)$ **then return** `Valid`
**if** $\neg s_0 \in \sigma_{\text{minFix}}(v_0)$ **then return** `Invalid`
Stack $\leftarrow \langle \sigma_{\text{minFix}} \cup \{s_0(v_0)\}, \{s_0(v_0)\}, \{\text{atoms}(G', S)\}, 0 \rangle$
**while** NONEMPTY(Stack) **do**
  $\langle \sigma, \sigma^P, A, n \rangle \leftarrow$ POP(Stack)
  **if** $\sigma^P = \emptyset$ **then return** `Valid`
  **for all** $\sigma' \in$ EXTEND$(\sigma, \sigma^P, A)$ **do**
    PUSH$(\mathcal{T}, \langle \sigma', \sigma' \setminus \sigma, A, n+1 \rangle)$
  **end for**
  **if** $n \geq k$ **then** Stack $\leftarrow$ REDUCE(Stack, $|\mathcal{T}|$)
**end while**
**return** `Unknown`

**procedure** REDUCE(Stack, $m$)
  $i = 0$
  **while** $i \leq m$ **do**
    $(\langle \sigma_1, \sigma_1^P, A_1, n_1 \rangle, \langle \sigma_2, \sigma_2^P, A_2, n_2 \rangle) \leftarrow$ GETCLOSESTPAIR(Stack)
    Stack $\leftarrow$ Stack $\setminus \{\langle \sigma_1, \sigma_1^P, A_1, n_1 \rangle, \langle \sigma_2, \sigma_2^P, A_2, n_2 \rangle\}$
    $A \leftarrow$ GETCONFLICTS$(\sigma_1, \sigma_2)$
    $\sigma_1 \leftarrow$ REPLACE$(\sigma_1, A)$
    $\sigma_2 \leftarrow$ REPLACE$(\sigma_2, A)$
    $\sigma = \max\{\sigma_1, \sigma_2\}$
    PUSH(Stack, $\langle \sigma, \sigma_1^P \cup \sigma_2^P, A \cup A_1 \cup A_2, \max\{n_1, n_2\} \rangle$)
    $i \leftarrow i + 1$
  **end while**
**end procedure**

ShEx has been actively developed since 2012 [6], as a dedicated constraint language for RDF, strongly inspired by XML schema languages. The first version of ShEx did support recursion, but no negation. A formal semantics was provided in [21], based on *regular bag expressions*. Recently, ShEx 2.0[9] incorporated negation, and a formal semantics was provided in [7], together with a abstract language called *Shape Schemas*. As highlighted in [5], ShEx and SHACL have lot in common, and the semantics provided in [7] 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* verifying target and constraints, similar to our notion of shape assignment. A difference though is that the semantics proposed in [7] is restricted to stratified constraints. Moreover, the (unique) typing used in [7] to define validation favors the validation of shapes in the lowest stratum, so that the graph of Figure 2 for instance would be considered invalid.

Another line of work is inspired by the Web Ontology Language (OWL), which is based on Description Logics (DLs) [3]. Like RDFS, OWL was not designed as a schema language, but adopts isntead the *open-world assumption*, not well-suited to express constraints. Still, proposals have been made to reason with DLs understood as

[9] `http://shex.io/shex-semantics/`

constraints: by introducing *auto-epistemic* operators [11], partitioning DL formulas into regular and constraint axioms [17, 22], or reasoning with closed predicates [19]. This last approach was actually proposed as a semantic grounding for SHACL [18], reducing constraint validation to first-order 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 (see e.g. [10]), and answer-set programming (see [20] in the context of SPARQL), where stable model semantics (SMS) is one of the most prominent paradigms [14]. But SMS is based on so-called *minimal* models, whereas shape assignments may not be minimal. This makes encoding SHACL into logical programming non trivial, as suggested by complexity results: ground-fact entailment is data-tractable for stratified Datalog, in contrast to our semantics (see Proposition 5). A possible way to relate the two semantics, at least for the stratified case, is to reason about shape "complements" under SMS. Still, our preliminary investigations tend to show that this is not straightforward.

## 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. Therefore we propose a sound approximation algorithm, parameterized by an integer $k$, which guarantees termination in time polynomial in the size of the graph.

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

*Acknowledgements.* This work was supported by the QUEST, ROBAST, OBATS and ADVANCE4KG projects at the Free University of Bozen-Bolzano, and the Millennium Institute for Foundational Research on Data (IMFD), Chile.

## Bibliography

[1] W. Akhtar, A. Cortés-Calabuig, and J. Paredaens. Constraints in RDF. In *Semantics in Data and Knowledge Bases - 4th International Workshops, SDKB 2010, Bordeaux, France, July 5, 2010, Revised Selected Papers*, pages 23–39, 2010.

[2] M. Arenas, C. Gutiérrez, and J. Pérez. Foundations of RDF Databases. In *Reasoning Web. Semantic Technologies for Information Systems*, pages 158–204, 2009.

[3] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider. *The description logic handbook: theory, implementation and applications*. Cambridge university press, 2003.

[4] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.
[5] I. Boneva. *Comparative expressiveness of ShEx and SHACL (Early working draft)*. 2016.
[6] I. Boneva, J. E. Labra-Gayo, S. Hym, E. G. Prud'hommeau, H. R. Solbrig, and S. Staworko. Validating RDF with shape expressions. *CoRR, abs/1404.1270*, 2014.
[7] I. Boneva, J. E. Labra-Gayo, and E. G. Prud'hommeaux. Semantics and Validation of Shapes Schemas for RDF. In *ISWC*, 2017.
[8] J. Corman, J. L. Reutter, and O. Savkovic. Semantics and validation of recursive shacl (extended version). *Technical Report KRDB18-1, KRDB Research Center, Free Univ. Bozen-Bolzano*, 2018.
[9] J. Corman, J. L. Reutter, and O. Savkovic. Validating graph data against recursive constraints: a semantics for shacl. *AMW (to appear)*, 2018.
[10] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3):374–425, 2001.
[11] F. M. Donini, D. Nardi, and R. Rosati. Description logics of minimal knowledge and negation as failure. *ACM Transactions on Computational Logic (TOCL)*, 3(2):177–225, 2002.
[12] F. J. Ekaputra and X. Lin. SHACL4p: SHACL constraints validation within Protégé ontology editor. In *ICoDSE*, 2016.
[13] P. M. Fischer, G. Lausen, A. Schätzle, and M. Schmidt. RDF constraint checking. In *Proceedings of the Workshops of the EDBT/ICDT 2015 Joint Conference (EDBT/ICDT), Brussels, Belgium, March 27th, 2015.*, pages 205–212, 2015.
[14] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. pages 1070–1080. MIT Press, 1988.
[15] S. Harris, A. Seaborne, and E. Prud'hommeaux. Sparql 1.1 query language. *W3C recommendation*, 21(10), 2013.
[16] E. V. Kostylev, J. L. Reutter, M. Romero, and D. Vrgoč. SPARQL with property paths. In *ISWC*, pages 3–18. Springer, 2015.
[17] B. Motik, I. Horrocks, and U. Sattler. Bridging the gap between OWL and relational databases. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(2):74–89, 2009.
[18] P. F. Patel-Schneider. Using Description Logics for RDF Constraint Checking and Closed-World Recognition. In *AAAI*, 2015.
[19] P. F. Patel-Schneider and E. Franconi. Ontology constraints in incomplete and complete data. In *ISWC*, 2012.
[20] A. Polleres and J. P. Wallner. On the relation between SPARQL1.1 and answer set programming. *Journal of Applied Non-Classical Logics*, 23(1-2):159–212, 2013.
[21] S. Staworko, I. Boneva, J. E. Labra-Gayo, S. Hym, E. G. Prud'hommeaux, and H. Solbrig. Complexity and Expressiveness of ShEx for RDF. In *ICDT*, 2015.
[22] J. Tao, E. Sirin, J. Bao, and D. L. McGuinness. Integrity Constraints in OWL. In *AAAI*, 2010.

# Semantics and Validation of Recursive SHACL: Appendix

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

# Table of Contents

The above article was submitted to the ISWC 2018 conference. The present 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 conversely

## 1.1 Scope

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 `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: `sh:datatype`, `sh:nodeKind`, `sh:minExclusive`, `sh:minInclusive`, `sh:maxExclusive`, `sh:maxInclusive`, `sh:minLength`, `sh:maxLength`, `sh:pattern`, `sh:languageln`, `sh:uniqueLang` or `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 `sh:lessThan` and `sh:lessThanOrEquals`.

## 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 $(s, \texttt{sh:hasValue}, I)$. We define:

$$t(s, \texttt{sh:hasValue}, I) \doteq I$$

### 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).

- $t(s, \texttt{sh:not}, s') = \neg s'$
- $t(s, \texttt{sh:and}, l) = \bigwedge_{s' \in l} s'$
- $t(s, \texttt{sh:or}, l) = \bigvee_{s' \in l} s'$
- $t(s, \texttt{sh:xone}, [s_1, .., s_n]) = \bigvee_{1 \leq i \leq n} (s_i \wedge \bigwedge_{1 \leq j \leq n, i \neq j} \neg s_j)$

### 1.2.3 Node shape definitions

Let $d_s$ be a node shape definition. Then $\mathrm{cons}(d_s)$ can be partitioned into 3 sets of triples $\mathrm{pp}(d_s)$, $\mathrm{closed}(d_s)$ and $\mathrm{loc}(d_s)$.

*1.2.3.1* $\mathrm{pp}(d_s)$. $\mathrm{pp}(d_s)$ is composed of all triples with property `sh:property`.
We translate it as:

$$t(\mathrm{pp}(d_s)) \doteq \bigwedge_{(s, \texttt{sh:property}, s') \in \mathrm{pp}(d_s)} t(d_{s'})$$

*1.2.3.2* $\mathrm{closed}(d_s)$. $\mathrm{closed}(d_s)$ containts all triples with property `sh:closed` or `sh:ignoredProperties`

- If $\mathrm{closed}(d_s)$ contains $(s, \texttt{sh:closed}, \texttt{false})$, then $t(\mathrm{closed}(d_s)) = \top$.
- If $\mathrm{closed}(d_s)$ contains $(s, \texttt{sh:closed}, \texttt{true})$, then we define $\mathrm{ignored}(d_s)$ as follows:
  - If $\mathrm{closed}(d_s)$ contains $(s, \texttt{sh:ignoredProperties}, l)$, then $\mathrm{ignored}(d_s) = \{p \mid p \in l\}$.
  - otherwise, $\mathrm{ignored}(d_s) = \emptyset$

  Now if $r$ is a property path, let $\mathrm{signat}(r)$ be all properties appearing in $r$.
  And let $R = \{r \mid (s, \texttt{sh:property}, s') \in \mathrm{pp}(d_s)$ and $(s', \texttt{sh:path}, r) \in d_{s'}\}$.
  We define the regular path expression $e(s)$, which is the complement (!) of the disjunction (|) of all properties appearing in $R$ or in $\mathrm{ignored}(d_s)$.
  More formally, let $E_s = (\bigcup_{r \in R} \mathrm{signat}(r)) \cup \mathrm{ignored}(d_s)$.
  Then $e_s = !(e_1|..|e_n)$, where $E_s = \{e_1, .., e_n\}$.

  We can now define $t(\mathrm{closed}(d_s)) \doteq \leq_0 e_s.\top$

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

*1.2.3.4 Translation of* $d_s$. We can now define the translation $t(d_s)$ of $\mathrm{cons}(d_s)$ into $\mathcal{L}$ as:

$$t(d_s) \doteq t(\mathrm{pp}(d_s)) \wedge t(\mathrm{closed}(d_s)) \wedge t(\mathrm{loc}(d_s))$$

### 1.2.4 Property shape definition

Let $d_s$ be a property shape definition, with $r_s$ the value for `sh: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:hasValue`, `sh:not`, `sh:and`, `sh:xone` or `sh:or`.

If $\text{loc}(d_s) = \emptyset$, then $t(\text{loc}(d_s)) \doteq \top$

Otherwise, $t(\text{loc}(d_s)) \doteq \bigwedge_{m \in \text{loc}(d_s)} t(m)$

*1.2.4.2* $\text{nsRef}(d_s)$. Let $\text{nsRef}(s)$ be the set of triples with property `sh:node` in $d_s$.

Then $t(\text{nsRef}(d_s)) \doteq \bigwedge_{(s, \texttt{sh:node}, s') \in \text{nsRef}(d_s)} s'$

*1.2.4.3* $\text{nsRefQ}(d_s)$. Similarly, let $\text{nsRefQ}(d_s)$ be the set of triples with property `sh:qualifiedValueShape` in $d_s$.

Then $t(\text{nsRefQ}(d_s)) \doteq \bigwedge_{(s, \texttt{sh:qualifiedValueShape}, s') \in \text{nsRefQ}(d_s)} s'$

*1.2.4.4* $\text{eq}(d_s)$. Let $\text{eq}(d_s)$ be the set of triples with property path `sh:equals` in $d_s$

Then $t(\text{eq}(d_s)) \doteq \bigwedge_{(s, \texttt{sh:equals}, r) \in \text{eq}(d_s)} r_s = r$

*1.2.4.5* $\text{disj}(d_s)$. Let $\text{disj}(d_s)$ be the set of triples with property path `sh:disjoint`.

Then $t(\text{disj}(d_s)) \doteq \bigwedge_{(s, \texttt{sh:disjoint}, r) \in \text{disj}(d_s)} (\leq_0 r_s.\neg s_r^1) \wedge (\leq_0 r.\neg s_r^2)$, where $s_r^1$ and $s_r^2$ are fresh shape names, whose definitions in $\mathcal{L}$ are $\neg s_r^2$ and $\neg s_r^1$ respectively.

*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:minCount` or `sh:maxCount`. For each triple $(s, p, n) \in \text{quant}(d_s)$, we define:

$t(s, p, n) \doteq \geq_n r_s.$ if $p$ is `sh:minCount`, and

$t(s, p, n) \doteq \leq_n r_s.$ if $p$ is `sh: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:qualifiedMinCount` or `sh:qualifiedMaxCount`. For each triple $(s, p, n) \in \text{quantQ}(s)$, we define:

$t(s, p, n) \doteq \geq_n r_s.$ if $p$ is `sh:qualifiedMinCount`, and

$t(s, p, n) \doteq \leq_n r_s.$ if $p$ is `sh: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 $(s, \texttt{sh:qualifiedValueShapesDisjoint}, \texttt{true})$, and $\text{sib}(s) = \emptyset$ otherwise.

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

$$\text{nqual}(d_s) \doteq t(\text{loc}(d_s)) \wedge t(\text{nsRef}(d_s))$$

Then $t(d_s)$ is defined as:

$$t(d_s) \doteq t(\text{eq}(d_s)) \ \wedge \ t(\text{disj}(d_s)) \wedge \ \leq_0 r.\neg \ \text{nqual}(d_s) \wedge$$
$$\bigwedge_{q \in \text{quant}(d_s)} t(\text{quant}(s))(\text{nsRef}(d_s)) \wedge$$
$$\bigwedge_{q \in \text{quantQ}(d_s)} t(\text{quantQ}(d_s))(t(\text{nsRefQ}(d_s)) \wedge \bigwedge_{s \in \text{sib}(d_s)} \neg s)$$

### 1.3 Translation from $\mathcal{L}$ to SHACL

Let $S = \{(s_1, \phi_{s_1}, \text{target}_{s_1}), .., (s_n, \phi_{s_n}, \text{target}_{s_n})\}$ 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 $\{u(\phi_{s_i}), .., u(\phi_{s_n})\}$, 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 $\text{target}_{s_i}$ 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:

$$\phi \ ::= \ \top \mid s \mid I \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \ \geq_n r.\phi \mid r_1 = r_2$$

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 can now define $h(\phi)$, by induction on $\phi$:

- If $\phi = \top$, then $h(\phi)$ is a shape definition with no constraint, i.e.:
  $s_\phi$ `a sh:NodeShape .`
- If $\phi = s$, then $h(\phi)$ is defined as:
  $s_\phi$ `a sh:NodeShape ; sh:and` $([s, s])$ `.`
  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:
  $s_\phi$ `a sh:NodeShape ; sh:hasValue` $I$ `.`
- If $\phi = \phi_1 \wedge \phi_2$, then $h(\phi)$ is defined as:
  $s_\phi$ `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:
  $s_\phi$ `a sh:NodeShape ; sh:not` $\text{sh}(h(\phi_1))$`.`
- If $\phi = \geq_n r.\phi'$, then $h(\phi)$ is defined as:

```
 sφ a sh:NodeShape ;
    sh:property [
        sh:path (r) ;
        sh:qualifiedValueShape ;
        sh:qualifiedMinCount n ;
        sh:node sh(h(φ'))
   ] .
```

  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 `sh:minCount` only.

– If $\phi$ is $r1 = r2$, then $h(\phi)$ is defined as:

```
s_φ 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 `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, $S = \{s_0 \mapsto \phi_{s_0}, .., s_n \mapsto \phi_{s_n}\}$ 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 $\langle V_G, E_G\rangle$, 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 $\Diamond_r\phi$ for $\geq_1 r.\phi$, and $\Box_r\phi$ for $\leq_0 r.\neg\phi$.

#### 2.1.3 Function, lattice, fixed point

- If $f$ is a function, then $\mathrm{dom}(f)$ designates its domain, and $\mathrm{range}(f)$ its range.
- If $A \subseteq \mathrm{dom}(f)$, then $f|_A$ designates $f$ restricted to $A$.
- If $f$ is a function, and $A \subseteq \mathrm{dom}(f)$, then $\mathrm{fix}(f, A)$ designates all fixed points of $f$ over $A$.
- If $P = \langle U, \preceq \rangle$ is a partially ordered set and $u \in U$, then $\mathrm{ext}_P(u)$ 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 $u \in \mathrm{fix}(f, U)$, and $u \preceq u'$ for all $u' \in \mathrm{fix}(f, U)$.*

**Definition 2.** *Let $P = \langle U, \preceq \rangle$ be a partially ordered set. A function $f : U \to U$ 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 $f : U \to U$ 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 $\mathrm{atoms}(G, S) = \{s(v) \mid \phi_s \in S$ 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* $\mathrm{atoms}(G, S)$ *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* $\mathrm{atoms}(G, S)$ *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 $\mathrm{signat}(S)$ designates all predicates and constants appearing in some constraint formula of $S$.

**Definition 5 (Target compliant assignment).**
*$\sigma$ is a* target compliant assignment *for $\langle G, S, s_0(v_0) \rangle$ 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 $s(v) \in \mathrm{atoms}(G, S)$, $\sigma s(v) = 0$ implies $[\![\phi_s]\!]^{v,G,\sigma} = 0$, and $\sigma s(v) = 1$ implies $[\![\phi_s]\!]^{v,G,\sigma} = 1$.*

**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 $s(v) \in \text{atoms}(G, S)$, $\sigma s(v) = [\![\phi_s]\!]^{v,G,\sigma}$*

**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 $S$
- $\Sigma^{n,\text{tar}}_{G,S,s_0(v_0)}$ designates all $n$-valued target compliant assignments for $G$, $S$ and $s_0(v_0)$
- $\Sigma^{n,\text{cst}}_{G,S}$ designates all $n$-valued constraint satisfying assignments for $G$ and $S$
- $\Sigma^{n,\text{fix}}_{G,S}$ designates all $n$-valued fixed-point assignments for $G$ and $S$
- $\Sigma^{n,\text{fai}}_{G,S,s_0(v_0)}$ designates all $n$-valued faithful assignments for $G$, $S$ and $s_0(v_0)$
- $\Sigma^{n,\text{str}}_{G,S,s_0(v_0)}$ designates all $n$-valued strictly faithful assignments for $G$, $S$ and $s_0(v_0)$

Similarly, $\Sigma^n_S$ 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 $\sigma \in \Sigma^3_{G,S}$, $\langle \text{ext}_L(\sigma), \preceq\rangle$ 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 $S_1, .., S_n$ (from lowest to highest), we use $S_{\leq j}$ to designate $\bigcup\limits_{i=1}^{j} S_i$.

### 2.1.8 Immediate evaluation

**Definition 12.** *Given a graph $G$ and set $S$ of shapes, the function $\mathbf{T}^{G,S} : \Sigma^n_{G,S} \to \Sigma^n_{G,S}$ is defined by:*
$\mathbf{T}^{G,S}(\sigma)s(v) = [\![\phi_s]\!]^{v,G,\sigma}$

- For readability, we write $\mathbf{T}^{G,S}\sigma$ instead of $\mathbf{T}^{G,S}(\sigma)$.
- Abusing notation, if $\Sigma' \subseteq \Sigma^n_{G,S}$, we use $\mathbf{T}^{G,S}\Sigma'$ to designate $\{\mathbf{T}^{G,S}\sigma \mid \sigma \in \Sigma'\}$.

### 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.

$$\phi \ ::= \ \top \mid s \mid I \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \ \geq_n r.\phi \mid r_1 = r_2 \tag{1}$$

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

The following additional operators are used as syntactic sugar:
- $\phi_1 \vee \phi_2$ for $\neg(\neg\phi_1 \wedge \neg\phi_2)$
- $\leq_n r.\phi$ for $\neg(\geq_{n+1} r.\phi)$
- $= n.r\phi$ for $\geq_n r.\phi \wedge \ \leq_n r.\phi$,
- $\Diamond_r\phi$ for $\geq_1 r.\phi$
- $\Box_r\phi$ for $\leq_0 r.\neg\phi$
- $\bot$ for $\neg\top$

*2.1.9.2* $\mathcal{L}_{\Diamond,\neg,\wedge}$ The constraint language $\mathcal{L}_{\Diamond,\neg,\wedge}$ 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:

$$\phi \ ::= \ \top \mid s \mid I \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \ \geq_1 p.\phi \tag{2}$$

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

*2.1.9.3* $\mathcal{L}_{n,\wedge,\vee}$ The constraint language $\mathcal{L}_{n,\wedge,\vee}$ 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:

$$\phi \ ::= \ \top \mid s \mid I \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \ \geq_n p.\phi \tag{3}$$

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 $\sigma_1, \sigma_2 \in \Sigma^3_{G,S}$, with $\sigma_1 \preceq \sigma_2$. For any $\phi$ over $\mathrm{signat}(S)$ and $v \in V_G$, if $[\![\phi]\!]^{v,G,\sigma_1} \neq 0.5$, then $[\![\phi]\!]^{v,G,\sigma_1} = [\![\phi]\!]^{v,G,\sigma_2}$.*

*Proof.* By induction on $\phi$.

$\phi = \top$ **:**
$[\![\top]\!]^{v,G,\sigma_1} = [\![\top]\!]^{v,G,\sigma_2} = 1$

$\phi = I$ **:**
If $[\![I]\!]^{v,G,\sigma_1} = 1$, then $v = I$, therefore $[\![I]\!]^{v,G,\sigma_2} = 1$.
If $[\![I]\!]^{v,G,\sigma_1} = 0$, then $v \neq I$, therefore $[\![I]\!]^{v,G,\sigma_2} = 0$.
$\phi = s$ **:**
Let $\sigma_1 s(v) = 0$. Because $\sigma_1 \preceq \sigma_2$, $\sigma_2 s(v) = [\![s]\!]^{v,G,\sigma_2} = 0$.
Similarly for the case $\sigma_1 s(v) = 1$.
$\phi = \phi_1 \wedge \phi_2$ **:**
If $[\![\phi]\!]^{v,G,\sigma_1} = 0$, then $[\![\phi_1]\!]^{v,G,\sigma_1} = 0$ or $[\![\phi_2]\!]^{v,G,\sigma_1} = 0$.
So by IH, $[\![\phi_1]\!]^{v,G,\sigma_2} = 0$ or $[\![\phi_2]\!]^{v,G,\sigma_2} = 0$.
Therefore $[\![\phi]\!]^{v,G,\sigma_2} = 0$.
If $[\![\phi]\!]^{v,G,\sigma_1} = 1$, then $[\![\phi_1]\!]^{v,G,\sigma_1} = 1$ and $[\![\phi_1]\!]^{v,G,\sigma_2} = 1$.
So by IH, $[\![\phi_1]\!]^{v,G,\sigma_2} = 1$ and $[\![\phi_2]\!]^{v,G,\sigma_2} = 1$.
Therefore $[\![\phi]\!]^{v,G,\sigma_2} = 1$.
$\phi = \neg\phi'$ **:**
If $[\![\phi]\!]^{v,G,\sigma_1} = 0$, then $[\![\phi']\!]^{v,G,\sigma_1} = 1$.
So by IH, $[\![\phi']\!]^{v,G,\sigma_2} = 1$.
Therefore $[\![\phi]\!]^{v,G,\sigma_2} = 0$.
Similarly for the case $[\![\phi]\!]^{v,G,\sigma_1} = 1$.
$\phi = \geq_n r.\phi'$ **:**
If $[\![\phi]\!]^{v,G,\sigma_1} = 1$, then $|\{v' \mid G \models r(v,v') \text{ and } [\![\phi']\!]^{v',G,\sigma_1} = 1\}| \geq n$.
By IH, if $[\![\phi']\!]^{v',G,\sigma_1} = 1$, then $[\![\phi']\!]^{v',G,\sigma_2} = 1$.
So $|\{v' \mid G \models r(v,v') \text{ and } [\![\phi']\!]^{v',G,\sigma_2} = 1\}| \geq$
$|\{v' \mid G \models r(v,v') \text{ and } [\![\phi']\!]^{v',G,\sigma_1} = 1\}| \geq n$.
Therefore $[\![\phi]\!]^{v,G,\sigma_2} = 1$.
If $[\![\phi]\!]^{v,G,\sigma_1} = 0$, then $|\{v' \mid G \models r(v,v')\}| -$
$|\{v' \mid G \models r(v,v') \text{ and } [\![\phi']\!]^{v',G,\sigma_1} = 0\}| < n$.
And by IH, if $[\![\phi']\!]^{v',G,\sigma_1} = 0$, then $[\![\phi']\!]^{v',G,\sigma_2} = 0$.
So $|\{v' \mid G \models r(v,v') \text{ and } [\![\phi']\!]^{v',G,\sigma_1} = 0\}| \leq$
$|\{v' \mid G \models r(v,v') \text{ and } [\![\phi']\!]^{v',G,\sigma_2} = 0\}|$.
It follows that $|\{v' \mid G \models r(v,v')\}| - |\{v' \mid G \models r(v,v') \text{ and } [\![\phi']\!]^{v',G,\sigma_2} = 0\}| \leq$
$|\{v' \mid G \models r(v,v')\}| - |\{v' \mid G \models r(v,v') \text{ and } [\![\phi']\!]^{v',G,\sigma_1} = 0\}| < n$.
Therefore $[\![\phi]\!]^{v,G,\sigma_2} = 0$. □

**Lemma 2.** *For any graph $G$ and set $S$ of shapes, $\mathbf{T}^{G,S}$ is monotone over $\langle \Sigma^3_{G,S}, \preceq \rangle$.*

*Proof.* For each $s \in \mathrm{dom}(S)$, $\phi_s$ will designate $S(s)$.
Let $\sigma_1, \sigma_2 \in \Sigma^3_{G,S}$, with $\sigma_1 \preceq \sigma_2$.
Take any $s(v) \in \mathrm{atoms}(G,S)$. 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, $\mathbf{T}^{G,S}\sigma_i s(v) = [\![\phi_s]\!]^{v,G,\sigma_i}$.
Therefore $\mathbf{T}^{G,S}\sigma_1 \preceq \mathbf{T}^{G,S}\sigma_2$. □

**Lemma 3.** *Let $G$ be a graph, $S$ a set of shapes, and $L = \langle \Sigma^3_{G,S}, \preceq \rangle$.*
*For any $\sigma \in \Sigma^3_{G,S}$, $\sigma \preceq \mathbf{T}^{G,S}\sigma$ iff $\mathbf{T}^{G,S}\,\mathrm{ext}_L(\sigma) \preceq \mathrm{ext}_L(\sigma)$.*

*Proof.*
$\Rightarrow$.

Let $\sigma \in \Sigma^3_{G,S}$, and $\sigma \preceq \mathbf{T}^{G,S}\sigma$. We need to show that $\mathbf{T}^{G,S}\sigma' \in \mathrm{ext}_L(\sigma)$ for any $\sigma' \in \mathrm{ext}_L(\sigma)$.
Because $\sigma' \in \mathrm{ext}_L(\sigma)$, $\sigma \preceq \sigma'$.
So from Lemma 2, $\mathbf{T}^{G,S}\sigma \preceq \mathbf{T}^{G,S}\sigma'$.
Then as $\sigma \preceq \mathbf{T}^{G,S}\sigma$, from the transitivity of $\preceq$, $\sigma \preceq \mathbf{T}^{G,S}\sigma'$.
Therefore $\mathbf{T}^{G,S}\sigma' \in \mathrm{ext}_L(\sigma)$.
$\Leftarrow$.
Let $\mathbf{T}^{G,S}\,\mathrm{ext}_L(\sigma) \preceq \mathrm{ext}_L(\sigma)$.
Then for each $\sigma' \in \mathrm{ext}_L(\sigma)$, $\mathbf{T}^{G,S}\sigma' \in \mathrm{ext}_L(\sigma)$.
In particular, because $\sigma \in \mathrm{ext}_L(\sigma)$, $\mathbf{T}^{G,S}\sigma \in \mathrm{ext}_L(\sigma)$.
So from the definition of $\mathrm{ext}_L(\sigma)$, $\sigma \preceq \mathbf{T}^{G,S}\sigma$. □

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

*Proof.* For each $s \in \mathrm{dom}(S)$, $\phi_s$ will designate $S(s)$.
Let $L = \langle \Sigma^3_{G,S}, \preceq \rangle$, and let $\sigma \in \Sigma^{3,\mathrm{cst}}_{G,S}$, such that $\sigma s_0(v_0) = 1$.
Take any $s(v) \in \mathrm{atoms}(G,S)$.

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

Therefore $\sigma \preceq \mathbf{T}^{G,S}\sigma$.
So from Lemma 3, $\mathbf{T}^{G,S}\,\mathrm{ext}_L(\sigma) \subseteq \mathrm{ext}_L(\sigma)$, i.e. $\mathbf{T}^{G,S}|_{\mathrm{ext}_L(\sigma)}$ is a function from $\mathrm{ext}_L(\sigma)$ to $\mathrm{ext}_L(\sigma)$.
In addition, because $\mathrm{ext}_L(\sigma) \subseteq \Sigma^3_{G,S}$, from Lemma 2, $\mathbf{T}^{G,S}$ is monotone over $\langle \mathrm{ext}_L(\sigma), \preceq \rangle$.
Therefore from Theorem 1, $\mathbf{T}^{G,S}$ has a minimal fixpoint $\sigma'$ over $\langle \mathrm{ext}_L(\sigma), \preceq \rangle$.
So $\sigma' \in \Sigma^{3,\mathrm{fix}}_{G,S}$. And because $\sigma' \in \mathrm{ext}_L(\sigma)$, $\sigma' s_0(v_0) = 1$. □

**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 $\sigma \in \Sigma^{3,fix}_{G,S_{\leq i}}$ and $\sigma|_{S_{\leq i-1}} \in \Sigma^{2,fix}_{G,S_{\leq i-1}}$, then $\mathrm{fil}(\sigma)|_{S_{\leq i}} \in \Sigma^{2,fix}_{G,S_{\leq i}}$.*

*Proof.* For each $s \in \mathrm{dom}(S)$, $\phi_s$ will designate $S(s)$.
Let $\sigma \in \Sigma^{3,\mathrm{fix}}_{G,S_{\leq i}}$ with $\sigma|_{S_{\leq i-1}} \in \Sigma^{2,\mathrm{fix}}_{G,S_{\leq i-1}}$, let $\sigma' = \mathrm{fil}(\sigma)|_{S_{\leq i}}$, and let $s(v) \in S_{\leq i}$.
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 $\sigma \in \Sigma^{3,\mathrm{fix}}_{G,S_{\leq i}}$, $\sigma s(v) = [\![\phi_s]\!]^{v,G,\sigma}$.
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 $\sigma' = \mathrm{fil}(\sigma|_{S_{\leq i}})$, $\sigma' s(v) = 1$. So we need to show that $[\![\phi_s]\!]^{v,G,\sigma'} = 1$.
First, because $\sigma s(v) = 0.5$ and $\sigma \in \Sigma^{3,\mathrm{fix}}_{G,S_{\leq i}}$, $[\![\phi_s]\!]^{v,G,\sigma} = 0.5$ 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 $\sigma' = \text{fil}(\sigma|_{S_{\leq i}})$, $\sigma' s(w) = 1$.
Therefore $[\![s]\!]^{w,G,\sigma'} = 1$.

**$\phi = \phi_1 \wedge \phi_2$ :**
If $[\![\phi]\!]^{w,G,\sigma} = 0$, 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 $[\![\phi]\!]^{w,G,\sigma} \geq 0.5$, 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 $\sigma|_{S_{\leq i-1}} \in \Sigma^{2,\text{fix}}_{G,S_{\leq i-1}}$, 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 $[\![\phi']\!]^{w,G,\sigma'} = 1$.

**$\phi = \geq_n r.\phi'$ :**
Let $Y = \{y \mid G \models r(w, y)\}$, $Y^0_\sigma = \{y \in Y \mid [\![\phi']\!]^{y,G,\sigma} = 0\}$, $Y^{0.5}_\sigma = \{y \in Y \mid [\![\phi']\!]^{y,G,\sigma} = 0.5\}$, and $Y^1_\sigma = \{y \in Y \mid [\![\phi']\!]^{y,G,\sigma} = 1\}$.
Similarly, define $Y^0_{\sigma'}$, $Y^{0.5}_{\sigma'}$ and $Y^1_{\sigma'}$.
If $[\![\phi]\!]^{w,G,\sigma} = 0$, then $|Y| - |Y^0_\sigma| < n$.
By IH, for each $y \in |Y^0|$, $[\![\phi']\!]^{y,G,\sigma'} = 0$.
So $|Y| - |Y^0_{\sigma'}| < n$.
Therefore $[\![\phi]\!]^{w,G,\sigma'} = 0$.
If $[\![\phi]\!]^{w,G,\sigma} \geq 0.5$, then $|Y| - (|Y^{0.5}_\sigma| + |Y^1_\sigma|) \geq n$.
By IH, for each $y \in Y^{0.5}_\sigma \cup Y^1_\sigma$, $[\![\phi']\!]^{y,G,\sigma'} = 1$.

So $|Y_{\sigma}^{0.5}| + |Y_{\sigma}^{1}| = |Y_{\sigma'}^{1}|$.
This yields $|Y| - |Y_{\sigma'}^{1}| \geq n$.
Therefore $[\![\phi]\!]^{w,G,\sigma'} = 1$. □

**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 $\sigma \in \Sigma_{G,S}^{3,\mathit{fix}}$ such that $\sigma s_0(v_0) = 1$, then there is a $\sigma' \in \Sigma_{G,S}^{2,\mathit{fix}}$ such that $\sigma' s_0(v_0) = 1$.*

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

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

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

- $\sigma'|_{S_{\leq 1}} = \text{fil}(\sigma|_{S_1})$.
- Let $\tau_i = \sigma'|_{S_{i-1}} \cup \sigma|_{S_i}$, and let $\theta_i$ be the minimal fixpoint of $\mathbf{T}^{G,S_{\leq i}}$ over $\langle \text{ext}_L(\tau_i), \preceq \rangle$ (we show below that $\theta_i$ must exist). Then $\sigma'|_{S_{\leq i}} = \text{fil}(\theta_i)$.

We now show that $\sigma'|_{S_{\leq i}} \in \Sigma_{G,S_{\leq i}}^{2,\text{fix}}$ for $1 \leq i \leq n$. It follows that $\sigma' = \sigma'|_{S_{\leq n}} \in \Sigma_{G,S}^{2,\text{fix}}$.

First, observe that for any $1 \leq i \leq n$, because $\theta_i$ is a fixpoint of $\mathbf{T}^{G,S_{\leq i}}$ over $\langle \text{ext}_L(\theta_i), \preceq \rangle$, and because $\text{ext}_L(\theta_i) \subseteq \Sigma_{G,S}^{3}$, $\theta_i$ is also a fixpoint of $\mathbf{T}^{G,S_{\leq i}}$ over $\langle \Sigma_{G,S}^{3}, \preceq \rangle$, i.e. $\theta_i \in \Sigma_{G,S_{\leq i}}^{3,\text{fix}}$.

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

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

To complete the proof, we show that $\theta_i$ must exists (in the inductive case $i > 1$), i.e. that $\mathbf{T}^{G,S_{\leq i}}$ must admit a fixpoint over $\langle \text{ext}_L(\tau_i), \preceq \rangle$.
From Lemma 3, it is sufficient to show that $\tau_i \preceq \mathbf{T}^{G,S_{\leq i}} \tau_i$, i.e. that for any $s(v) \in \text{atoms}(G, S_{\leq i})$, if $\tau_i s(v) \neq 0.5$, then $\tau_i s(v) = [\![\phi_s]\!]^{v,G,\tau_i}$.
We will first consider the case where $s$ is defined in $S_{\leq i-1}$, and then the case where $s$ is defined in $S_i$.

**$s$ is defined in $S_{\leq i-1}$**

From the definition of $\tau_i$, $\tau_i|_{S_{\leq i-1}} = \sigma'|_{S_{\leq i-1}}$.
So $\tau_i s(v) = \sigma'|_{S_{\leq i-1}} s(v)$, and $[\![\phi_s]\!]^{v,G,\tau_i|_{S_{\leq i-1}}} = [\![\phi_s]\!]^{v,G,\sigma'|_{S_{\leq i-1}}}$.
Then because $S$ is stratified, $[\![\phi_s]\!]^{v,G,\tau_i} = [\![\phi_s]\!]^{v,G,\tau_i|_{S_{\leq i-1}}}$.
Finally, because $\sigma'|_{S_{\leq i-1}} \in \Sigma_{G,S_{\leq i}}^{2,\text{fix}}$, $\sigma'|_{S_{\leq i-1}} s(v) = [\![\phi_s]\!]^{v,G,\sigma'|_{S_{\leq i-1}}}$.
This yields $\tau_i s(v) = [\![\phi_s]\!]^{v,G,\tau_i}$.

**$s$ 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 $\sigma \in \Sigma^{3,\text{fix}}_{G,S}$ and $S$ is stratified, $\sigma|_{S_{\leq i}} \in \Sigma^{3,\text{fix}}_{G,S_{\leq i}}$.
So $\sigma|_{S_{\leq i}} s(v) = [\![\phi_s]\!]^{v,G,\sigma|_{S_{\leq i}}}$.
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 $\sigma|_{S_{\leq i}} s(v) = [\![\phi_s]\!]^{v,G,\sigma|_{S_{\leq i}}}$, $[\![\phi_s]\!]^{v,G,\sigma|_{S_{\leq i}}} \neq 0.5$.
Then as $\sigma|_{S_{\leq i}} \preceq \tau_i$, from Lemma 1, $[\![\phi_s]\!]^{v,G,\sigma|_{S_{\leq i}}} = [\![\phi_s]\!]^{v,G,\tau_i}$.
This yields $\tau_i s(v) = [\![\phi_s]\!]^{v,G,\tau_i}$. □

**Proposition 3.** *Let $G$ be a graph, and $S$ a set of shape constraint definitions. Then $\mathbf{T}^{G,S}$ admits a unique minimal fixed-point $\sigma$ over $\langle \Sigma^3_{G,S}, \preceq \rangle$, and $\sigma$ is a constraint-compliant assignment for $G$ and $S$.*

*Proof.* Because $\langle \Sigma^3_{G,S}, \preceq \rangle$ is a meet semi-lattice and $dom(\mathbf{T}^{G,S}) = \Sigma^3_{G,S}$ and $\text{range}(\mathbf{T}^{G,S}) \subseteq \Sigma^3_{G,S}$, from Lemma 2 and Theorem 1, $\mathbf{T}^{G,S}$ admits a unique minimal fixed point over $\langle \Sigma^3_{G,S}, \preceq \rangle$.
Then from Definitions 6 and 7, a fixed-point assignment for $G$ ands $S$ is also a constraint compliant assignment for $G$ and $S$. □

### 2.3 Complexity Proofs

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

*Proof.* Reduction from CIRCUIT-SAT. Let $\psi$ be a boolean formula with variables $\{x_1, .., x_n\}$. We assume wl.o.g. that $\psi$ contains only AND and NOT as boolean operators. We build an instance $\langle G_\psi, S, s_0(v_0)\rangle$ of VALIDATION such that $\psi$ is satisfiable iff $\langle G_\psi, S, s_0(v_0)\rangle$ 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 $\psi = \text{NOT } \psi'$, then $H_\psi = H_{\psi'} \cup \{(v_0, \text{u}, v_\psi), (v_\psi, \text{self}, v_\psi), (v_\psi, \text{not}, v_{\psi'})\}$
- if $\psi = \psi_1 \text{ AND } \psi_2$, then $H_\psi = H_{\psi_1} \cup H_{\psi_2} \cup$ $\{(v_0, \text{u}, v_\psi), (v_\psi, \text{self}, v_\psi), (v_\psi, \text{and}, v_{\psi_1}), (v_\psi, \text{and}, v_{\psi_2})\}$

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

- $\phi_{s_\top} = \Diamond_{\text{self}} s_\top$
- $\phi_{s_{\text{TV}}} = (\Diamond_{\text{self}} s_\top \wedge \Box_{\text{and}} s_\top \wedge \Box_{\text{not}} \neg s_\top) \vee (\Diamond_{\text{self}} \neg s_\top \wedge (\Diamond_{\text{and}} \neg s_\top \vee \Diamond_{\text{not}} s_\top \vee (\Box_{\text{not}} \bot \wedge \Box_{\text{and}} \bot)))$
- $\phi_{s_0} = \Diamond_{\text{eval}} s_\top \wedge \Box_{\text{u}} s_{\text{TV}}$

$S$ is stratified, as shown by the function $\{s_\top \mapsto 1, s_{\text{TV}} \mapsto 2, s_0 \mapsto 3\}$.

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

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 $S' = \{\phi_{s_{\text{TV}}}, \phi_{s_\top}\}$ of $S$, and the subgraph $G_\psi^R$ of $G_\psi$, defined by $V_{G_\psi^R} = V_{G_\psi} \setminus \{v_0\}$, and $E_{G_\psi^R}$ is the set of triples in $E_{G_\psi}$ with edge label in $\{\text{self}, \text{and}, \text{not}\}$. Note that $V_{G_\psi^R} = \{v_{\psi'} \mid \psi' \in \Psi\}$.

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 $\Sigma^2_{G_\psi^R, S'}$ defined by $t(\beta)s_\top(v_{\psi'}) = [\![\psi']\!]^\beta$, and $t(\beta)s_{\text{TV}}(v_{\psi'}) = 1$. And let $t(B(X_\psi)) = \{t(\beta) \mid \beta \in B(X_\psi)\}$. Finally, let $\Sigma_{\text{TV}} = \{\sigma \in \Sigma^{2,\text{fix}}_{G_\psi^R, S'} \mid \sigma s_{\text{TV}}(v_\psi) = 1\}$. We will show below that $t(B(X_\psi)) = \Sigma_{\text{TV}}$.

For now, assuming that this claim holds, we show that $\psi$ is satisfiable iff there is a $\sigma \in \Sigma^{2,\text{fix}}_{G_\psi, S}$ 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 $\sigma = t(\beta) \cup \{\neg s_0(v) \mid v \in G_\psi^R\} \cup \{s_0(v_0), \neg s_{\text{TV}}(s_0), \neg s_\top(s_0)\}$.
  Then $\sigma s_\top(v_\psi) = 1$.
  Therefore because $\text{suc}_{\text{eval}}^{G_\psi}(v_0) = \{v_\psi\}$, $[\![\Diamond_{\text{eval}} \top]\!]^{v_0, G_\psi, \sigma} = 1$.
  Similarly, for all $v_{\psi'} \in V_{G_\psi^R}$, $\sigma s_{\text{TV}}(v_{\psi'}) = 1$, and $\text{suc}_{\text{eval}}^{G_\psi}(v_0) = \{V_{G_\psi^R}\}$, therefore $[\![\Box_{\text{TV}} \top]\!]^{v_0, G_\psi, \sigma} = 1$.
  So from the definition of $\phi_{s_0}$, $[\![\phi_{s_0}]\!]^{v_0, G_\psi, \sigma} = \sigma s_0(v_0) = 1$.
  In addition, because $\text{suc}_{\text{self}}^{G_\psi}(v_0) = \emptyset$, from the definition of $\phi_{s_{\text{self}}}$, $[\![\phi_{s_{\text{TV}}}]\!]^{v_0, G_\psi, \sigma} = \sigma s_{\text{TV}}(v_0) = 0$.

  Now take any $v \in V_{G_\psi^R}$.
  Because $t(\beta) \preceq \sigma$, $\sigma s_\top(v) = t(\beta)s_\top(v)$.
  Then because $\beta \in B(X_\psi)$ and $t(B(X_\psi)) = \Sigma_{\text{TV}}$, $t(\beta) \in \Sigma^{2,\text{fix}}_{G_\psi^R, S'}$.
  Therefore $t(\beta)s_\top(v) = [\![\phi_{s_\top}]\!]^{v, G_\psi^R, t(\beta)}$.
  Then from the definition of $G_\psi^R$, $\text{suc}_{\text{self}}^{G_\psi}(v) = \text{suc}_{\text{self}}^{G_\psi^R}(v)$.
  Therefore, from the definition of $\phi_\top$,
  $[\![\phi_{s_\top}]\!]^{v, G_\psi^R, t(\beta)} = [\![\phi_{s_\top}]\!]^{v, G_\psi^R, \sigma} = [\![\phi_{s_\top}]\!]^{v, G_\psi, \sigma}$ must hold.
  This yields $\sigma s_\top(v) = t(\beta)s_\top(v) = [\![\phi_{s_\top}]\!]^{v, G_\psi^R, t(\beta)} = [\![\phi_{s_\top}]\!]^{v, G_\psi, \sigma}$,
  therefore $\sigma s_\top(v) = [\![\phi_{s_\top}]\!]^{v, G_\psi, \sigma}$.
  A similar argument can be used to show $\sigma s_{\text{TV}}(v) = [\![\phi_{s_{\text{TV}}}]\!]^{v, G_\psi, \sigma}$.
  Finally, because $\text{suc}_{\text{eval}}^{G_\psi}(v) = \emptyset$, from the definition of $\phi_{s_0}$,
  $[\![\phi_{s_0}]\!]^{v, G_\psi, \sigma} = \sigma s_0(v_0) = 0$.

  So for any $v' \in \{v_0\} \cup V_{G_\psi^R} = V_{G_\psi}$, and for any $s \in S$,
  $[\![\phi_s]\!]^{v', G_\psi, \sigma} = \sigma s(v')$ holds.
  Therefore $\sigma \in \Sigma^{2,\text{fix}}_{G_\psi, S}$.

- $\Leftarrow$. Let $\sigma \in \Sigma^{2,\text{fix}}_{G_\psi,S}$ such that $\sigma s_0(v_0) = 1$.

  Take any $v \in V_{G^R_\psi}$.
  Because $s_\top \in S'$, $s_\top(v) \in \text{atoms}(G^R_\psi, S')$.
  And because $E_{G^R_\psi} \subseteq E_{G_\psi}$ and $S' \subseteq S$, $\text{atoms}(G^R_\psi, S') \subseteq \text{atoms}(G_\psi, S)$.
  Thefore $\sigma|_{\text{atoms}(G^R_\psi,S')} s_\top(v) = \sigma s_\top(v)$.
  Then because $\sigma \in \Sigma^{2,\text{fix}}_{G_\psi,S}$, $\sigma s_\top(v) = [\![\phi_{s_\top}]\!]^{v,G_\psi,\sigma}$.
  Finally, because $\text{suc}^{G_\psi}_{\text{self}}(v) = \text{suc}^{G^R_\psi}_{\text{self}}(v)$, from the definitions of $\phi_{s_\top}$,
  $[\![\phi_{s_\top}]\!]^{v,G_\psi,\sigma} = [\![\phi_{s_\top}]\!]^{v,G^R_\psi,\sigma|_{\text{atoms}(G^R_\psi,S')}}$.
  This yields $\sigma|_{\text{atoms}(G^R_\psi,S')} s_\top(v) = \sigma s_\top(v) = [\![\phi_{s_\top}]\!]^{v,G^R_\psi,\sigma} = [\![\phi_{s_\top}]\!]^{v,G^R_\psi,\sigma|_{\text{atoms}(G^R_\psi,S')}}$, therefore $\sigma|_{\text{atoms}(G^R_\psi,S')} s_\top(v) = [\![\phi_{s_\top}]\!]^{v,G^R_\psi,\sigma|_{\text{atoms}(G^R_\psi,S')}}$.
  A similar argument can be used to show $\sigma|_{\text{atoms}(G^R_\psi,S')} s_{\text{TV}}(v) = [\![\phi_{s_{\text{tv}}}]\!]^{v,G^R_\psi,\sigma|_{\text{atoms}(G^R_\psi,S')}}$.
  So as $S' = \{s_\top, s_{\text{TV}}\}$, $\sigma|_{\text{atoms}(G^R_\psi,S')} \in \Sigma^{2,\text{fix}}_{G^R_\psi,S'}$ holds.

  Now because $\sigma s_0(v_0) = 1$ and $\text{suc}^{G_\psi}_{\text{u}} = \{V_{G^R}\}$, from the definition of $\phi_{s_0}$, $\sigma|_{\text{atoms}(G^R_\psi,S')} s_{\text{TV}}(v) = 1$ must hold for each $v \in V_{G^R}$.
  Therefore $\sigma|_{\text{atoms}(G^R_\psi,S')} \in \Sigma_{\text{TV}}$.
  Finally, because $\sigma s_0(v_0) = 1$ and $\text{suc}^{G_\psi}_{\text{eval}} = \{v_\psi\}$, from the definition of $\phi_{s_0}$, $\sigma s_\top(v_\psi) = \sigma|_{\text{atoms}(G^R_\psi,S')} s_\top(v_\psi) = 1$ must hold.
  Then as $\sigma|_{\text{atoms}(G^R_\psi,S')} \in \Sigma_{\text{TV}}$, because $t(B(X_\psi)) = \Sigma_{\text{TV}}$, from the definition of $t$, there must be a $\beta \in B(X_\psi)$ such that $[\![\psi]\!]^\beta = 1$.
  Therefore $\psi$ is satisfiable.

To complete the proof, we need to show that $t(B(X_\psi)) = \Sigma_{\text{TV}}$:

- $\Rightarrow$.
  Let $\sigma \in t(B(X_\psi))$.
  Fom the definition of $f$, for all $v \in V_{G^R}$, $\sigma s_{\text{TV}}(v) = 1$ holds.
  So from the definition of $\Sigma_{\text{TV}}$, we only need to show that $\sigma \in \Sigma^{2,\text{fix}}_{G^R_\psi,S'}$
  for $S' = \{s_\top, s_{\text{TV}}\}$, i.e. that $\sigma s(v) = [\![\phi_s]\!]^{v,G^R_\psi,\sigma}$ for each $s(v) \in \text{atoms}(G^R_\psi, S')$.
  If $\sigma s_\top(v) = 0$, then because $\text{suc}^{G^R_\psi}_{\text{self}}(v) = \{v\}$, $[\![\Diamond_{\text{self}}]\!]^{v,G^R_\psi,\sigma} = 0$.
  So from the definition of $\phi_{s_\top}$, $\sigma s_\top(v) = [\![\phi_{s_\top}]\!]^{v,G^R_\psi,\sigma} = 0$.
  Similarly, if $\sigma s_\top(v) = 1$, then $[\![\phi_{s_\top}]\!]^{v,G^R_\psi,\sigma} = 1$.

  So we only need to show that $\sigma s_{\text{TV}}(v) = [\![\phi_{s_{\text{TV}}}]\!]^{v,G^R_\psi,\sigma}$.
  And because $\sigma \in t(B(X_\psi))$, from the definition of $t$, $\sigma s_{\text{TV}}(v) = 1$.
  Therefore it is sufficient to show that $[\![\phi_{s_{\text{TV}}}]\!]^{v,G^R_{\psi'},\sigma} = 1$.
  By induction on $\psi$:

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

  Let $\sigma s_\top(v_\psi) = 0$.
  And let $\phi_1$ be the right disjunct of $\phi_{s_{\text{TV}}}$,
  i.e. $\phi_1 = \Diamond_{\text{self}}\, \neg s_\top \wedge (\Diamond_{\text{and}} \neg s_\top \;\vee\; \Diamond_{\text{not}}\, s_\top \vee (\Box_{\text{not}} \bot \wedge \Box_{\text{and}} \bot))$.
  Because $\sigma s_\top(v_\psi) = 0$, $[\![\Diamond_{\text{self}} \neg s_\top]\!]^{v_\psi, G^R_\psi, \sigma} = 1$.
  In addition, because $\text{suc}^{G^R_\psi}_{\text{and}} = \text{suc}^{G^R_\psi}_{\text{not}} = \emptyset$,
  $[\![\Box_{\text{not}} \bot \wedge \Box_{\text{and}} \bot]\!]^{v_\psi, G^R_\psi, \sigma} = 1$.
  So $[\![\phi_1]\!]^{v_\psi, G^R_\psi, \sigma} = 1$.
  Therefore $[\![\phi_{\text{TV}}]\!]^{v_\psi, G^R_\psi, \sigma} = 1$.

  Now let Let $\sigma s_\top(v_\psi) = 0$.
  And let $\phi_2$ be the left disjunct of $\phi_{s_{\text{TV}}}$,
  i.e. $\phi_2 = \Diamond_{\text{self}} s_\top \wedge\; \Box_{\text{and}} s_\top \wedge\; \Box_{\text{not}} \neg s_\top$.
  Because $\sigma s_\top(v_\psi) = 1$, $[\![\Diamond_{\text{self}} s_\top]\!]^{v_\psi, G^R_\psi, \sigma} = 1$.
  In addition, because $\text{suc}^{G^R_\psi}_{\text{and}} = \text{suc}^{G^R_\psi}_{\text{not}} = \emptyset$,
  $[\![\Box_{\text{and}} s_\top \wedge\; \Box_{\text{not}} \neg s_\top]\!]^{v_\psi, G^R_\psi, \sigma} = 1$.
  So $[\![\phi_2]\!]^{v_\psi, G^R_\psi, \sigma} = 1$.
  Therefore $[\![\phi_{\text{TV}}]\!]^{v_\psi, G^R_\psi, \sigma} = 1$.

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

  Let $\sigma s_\top(v_\psi) = 0$.
  Because $\sigma \in t(B(X_\psi))$, $\sigma = t(\beta)$ for some $\beta \in B(X_\psi)$.
  And because $\sigma s_\top(v_\psi) = 0$, from the definition of $t$, $[\![\psi]\!]^\beta = 0$.
  Then as $\psi = \text{NOT } \psi'$, $[\![\psi']\!]^\beta = 1$ must hold,
  and from the definition of $t$, $\sigma s_\top(v_{\psi'}) = 1$.
  Finally, because $\text{suc}^{G^R_\psi}_{\text{and}}(v_\psi) = \emptyset$ and $\text{suc}^{G^R_\psi}_{\text{not}}(v_\psi) = \{v'_\psi\}$, from the definition of $\phi_{\text{TV}}$, $[\![\phi_{s_{\text{TV}}}]\!]^{v_\psi, G^R_{\psi'}, \sigma} = 1$.

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

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

  Let $\sigma s_\top(v_\psi) = 0$.
  Because $\sigma \in t(B(X_\psi))$, $\sigma = t(\beta)$ for some $\beta \in B(X_\psi)$.
  And because $\sigma s_\top(v_\psi) = 0$, 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 $[\![\psi_i]\!]^\beta = 0$, then $\sigma s_\top(v_{\psi_i}) = 0$.
  Finally, because $\text{suc}^{G^R_\psi}_{\text{and}}(v_\psi) = \{v_{\psi_1}, v_{\psi_2}\}$, from the definition of $\phi_{\text{TV}}$,

$[\![\phi_{s_{\text{TV}}}]\!]^{v_\psi, G^R_{\psi'}, \sigma} = 1$.

Now let $\sigma s_\top(v_\psi) = 1$.
Because $\sigma \in t(B(X_\psi))$, $\sigma = t(\beta)$ for some $\beta \in B(X_\psi)$.
And because $\sigma s_\top(v_\psi) = 1$, 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 $t$, $\sigma s_\top(v_{\psi_i}) = 1$.
Finally, because $\text{suc}^{G^R_\psi}_{\text{and}}(v_\psi) = \{v_{\psi_1}, v_{\psi_2}\}$, from the definition of $\phi_{\text{TV}}$, $[\![\phi_{s_{\text{TV}}}]\!]^{v_\psi, G^R_{\psi'}, \sigma} = 1$.

– $\Leftarrow$.
Let $\sigma \in \Sigma_{\text{TV}}$.
We need show that there is a $\beta \in B(X_\psi)$ such that $\sigma = t(\beta)$.
From the definition of $\Sigma_{\text{TV}}$, we already know that $s_{\text{TV}}(v_{\psi'}) = 1$ 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 $\sigma s_\top(v_{\psi'}) = [\![\psi']\!]^\beta$ for each $v_{\psi'} \in V_{G^R}$.
By induction on $\psi$:

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

- $\psi =$ 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 $e \in E_{G^R_{\psi'}}$,
  $\text{suc}_e^{G^R_\psi}(v) = \text{suc}_e^{G^R_{\psi'}}(v)$.
  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 $\sigma \in \Sigma_{\text{TV}}$, $\sigma \in \Sigma^{2,\text{fix}}_{G^R_\psi, S'}$.
  Therefore $\sigma' \in \Sigma^{2,\text{fix}}_{G^R_{\psi'}, S'}$.
  So by IH, $\sigma' = t(\beta)$ for some $\beta \in B(X_{\psi'})$.
  Therefore for each $v_{\psi_m} \in G^R_{\psi'}$, $\sigma' s_\top(v_{\psi_m}) = [\![\psi_m]\!]^\beta$.
  So as $\sigma' \preceq \sigma$, for each $v_{\psi_m} \in G^R_{\psi'}$, $\sigma s_\top(v_{\psi_m}) = [\![\psi_m]\!]^\beta$ holds.
  Therefore as $V_{G^R_\psi} \setminus V_{G^R_{\psi'}} = \{v_\psi\}$, we only need to show that $\sigma s_\top(v_\psi) = [\![\psi]\!]^\beta$.

  If $[\![\psi]\!]^\beta = 0$, then $[\![\psi']\!]^\beta = 1$.
  So as $\sigma' \preceq \sigma$ and $\sigma' = t(\beta)$, $\sigma s_\top(v_{\psi'}) = 1$ must hold.
  So from the definition of $G^R_\psi$, $[\![\Box_{\text{not}} \neg s_\top]\!]^{v_\psi, G^R, \sigma} = 0$,
  and $[\![\Diamond_{\text{not}} s_\top]\!]^{v_\psi, G^R, \sigma} = 1$.
  In addition, because $\sigma \in \Sigma_{\text{TV}}$, $[\![\phi_{\text{TV}}]\!]^{v_\psi, G^R, \sigma} = 1$.
  So from the definition of $\phi_{\text{TV}}$, $[\![\Diamond_{\text{self}} \neg s_\top]\!]^{v_\psi, G^R, \sigma} = 1$ must hold.
  And because $\text{suc}^{G^R}_{\text{self}}(v_\psi) = \{v_\psi\}$, $[\![s_\top]\!]^{v_\psi, G^R, \sigma} = 0$ must hold.

Finally, because $\sigma \in \Sigma_{\text{TV}}$, $[\![s_\top]\!]^{v_\psi, G^R, \sigma} = \sigma s_\top(v_\psi)$.
Thefore $\sigma s_\top(v_\psi) = 0$.
A similar argument can be used for the case $[\![\psi]\!]^\beta = 1$.

- $\psi = \psi_1$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 $v \in V_{G^R_{\psi_i}}$ and $e \in E_{G^R_{\psi_i}}$, $\text{suc}_e^{G^R_\psi}(v) = \text{suc}_e^{G^R_{\psi_i}}(v)$.
So for any $\phi$ and $v \in G^R_{\psi_i}$, $[\![\phi]\!]^{v, G^R_\psi, \sigma} = [\![\phi]\!]^{v, G^R_{\psi_i}, \sigma_i}$.
In addition, $\sigma_i \preceq \sigma$, and because $\sigma \in \Sigma_{\text{TV}}$, $\sigma \in \Sigma^{2,\text{fix}}_{G^R_\psi, S'}$.
Therefore $\sigma_i \in \Sigma^{2,\text{fix}}_{G^R_{\psi_i}, S'}$.
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 $v_x \in V_{G^R_{\psi_i}}$ with $x$ a propositional variable, $\sigma_i s_\top(v_x) = \beta_i(x)$.
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 $\sigma_1 s_\top(v_x) = \sigma_2 s_\top(v_x) = \sigma s_\top(v_x)$.
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 $\sigma_i = t(\beta_i)$ for $i \in \{1, 2\}$, for each $v_{\psi'} \in G^R_{\psi_i}$, $\sigma_i s_\top(v_{\psi'}) = [\![\psi']\!]^{\beta_i} = [\![\psi']\!]^\beta$.
So as $\sigma_1 \cup \sigma_2 \preceq \sigma$, for each $v_{\psi'} \in G^R_{\psi_1} \cup G^R_{\psi_2}$, $\sigma s_\top(v_{\psi'}) = [\![\psi']\!]^\beta$ holds.
Therefore as $V_{G^R_\phi} \setminus V_{G^R_{\psi_1} \cup G^R_{\psi_2}} = \{v_\psi\}$, we only need to show that $\sigma s_\top(v_\psi) = [\![\psi]\!]^\beta$.

If $[\![\psi]\!]^\beta = 0$, then $[\![\psi_i]\!]^\beta = 0$ for some $i \in \{1, 2\}$.
So as $\sigma_i \preceq \sigma$ and $\sigma_i = t(\beta_i)$ for some $\beta_i \in B(X_{\psi_i})$, $\sigma s_\top(v_{\psi_i}) = 0$ must hold.
So from the definition of $G^R_\psi$, $[\![\Box_{\text{and}} s_\top]\!]^{v_\psi, G^R, \sigma} = 0$,
and $[\![\Diamond_{\text{and}} \neg s_\top]\!]^{v_\psi, G^R, \sigma} = 1$.
In addition, because $\sigma \in \Sigma_{\text{TV}}$, $[\![\phi_{\text{TV}}]\!]^{v_\psi, G^R, \sigma} = 1$.
So from the definition of $\phi_{\text{TV}}$, $[\![\Diamond_{\text{self}} \neg s_\top]\!]^{v_\psi, G^R, \sigma} = 1$ must hold.
And because $\text{suc}^{G^R}_{\text{self}}(v_\psi) = \{v_\psi\}$, $[\![s_\top]\!]^{v_\psi, G^R, \sigma} = 0$ must hold.
Finally, because $\sigma \in \Sigma_{\text{TV}}$, $[\![s_\top]\!]^{v_\psi, G^R, \sigma} = \sigma s_\top(v_\psi)$.
Thefore $\sigma s_\top(v_\psi) = 0$.
A similar argument can be used for the case $[\![\psi]\!]^\beta = 1$.

$\square$

**Proposition 5 (Constraint – Lower Bound).** *For stratified $\mathcal{L}_{\Diamond,\neg,\wedge}$,* 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 $\{x_1, .., x_n\}$. 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 $\phi_{s_i^+} = \Diamond_p \neg s_i^-$, and $\phi_{s_i^-} = \Diamond_p s_i^-$.

Then we use an additional shape $s_0$, with unique target $v_0$, and defined by $\phi_{s_0} = \Diamond_r \operatorname{tr}(\mathrm{NNF}(\psi))$, where $\mathrm{NNF}(\psi)$ is $\psi$ in negation normal form, and $\operatorname{tr}(\mathrm{NNF}(\psi))$ is the straightforward boolean encoding of $\mathrm{NNF}(\psi)$, i.e. $\operatorname{tr}(\psi)$ is defined inductively over $\mathrm{NNF}(\psi)$ as follows:

- if $x_i$ is a non-negated variable, then $\operatorname{tr}(x_i) = s_i^+$
- if $x_i$ is a negated variable, then $\operatorname{tr}(x_i) = s_i^-$
- $\operatorname{tr}(\psi_1$ AND $\psi_2) = \operatorname{tr}(\psi_1) \wedge \operatorname{tr}(\psi_2)$
- $\operatorname{tr}(\psi_1$ OR $\psi_2) = \neg(\neg \operatorname{tr}(\psi_1) \wedge \neg \operatorname{tr}(\psi_2))$

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

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

**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 $E_{G'} = E_G \cup \{(v_1, p_r, v_2) \mid r \in R$ 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 $(v_1, v_2) \in [\![r]\!]^G$.
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 quadratic in $|G| + |S|$.

Now let $f_R$ be the function which takes an $\mathcal{L}$ formula $\phi$, and returns $\phi$ where each occurence of $r$ is replace by $p_r$, for each $r \in R$.
And let $S'$ be the function defined by $\mathrm{dom}(S') = \mathrm{dom}(S)$, and for each $s \in \mathrm{dom}(S)$,

$S'(s) = f_R(S(s))$.
Then for each $(v_1, v_2) \in (V_G)^2 = (V_{G'})^2$, $[\![r]\!]^G = [\![p_r]\!]^{G'}$.
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 $\langle G', S', s_0(v_0)\rangle$. □

**Lemma 6.** *Let $\langle G, S, s_0(v_0)\rangle$ be an instance of* VALIDATION*, such that no constraints in* $\mathrm{range}(S)$ *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 $\mathrm{range}(S_n)$ 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 $\mathrm{range}(S)$.

More formally, if $\phi \in \mathcal{L}$, let $\mathrm{opN}(\phi)$ be the number of operators (either $\neg$, $\wedge$ or $\geq_q r$) present in $\phi$.
We define the function $\mathrm{norm}()$, which takes a function $S = \{s_0 \mapsto \phi^S_{s_0}, .., s_m \mapsto \phi^S_{s_m}\}$ from shape names to formulas in $\mathcal{L}$, and returns another:

- If $\mathrm{opN}(\phi) \leq 1$ for each $\phi \in \mathrm{range}(S)$, then $\mathrm{norm}(S) = S$.
- Otherwise, let $s_i$ be some $s_i \in \mathrm{dom}(S)$ such that $\mathrm{opN}(\phi^S_{s_i}) > 1$. And let $N_S$ be a set of shape names disjoint from $\mathrm{dom}(S)$.
  Then $\mathrm{norm}(S)$ is defined as $\mathrm{norm}(S) = S|_{\mathrm{dom}(S)\setminus\{s_i\}} \cup \mathrm{fold}(s_i, \phi^S_{s_i}, N_S)$, where $\mathrm{fold}(s_i, \phi^S_{s_i}, N_S)$ is defined as follows:
  - if $\phi^S_{s_i} = \phi_1 \wedge \phi_2$, then $\mathrm{fold}(s_i, \phi^S_{s_i}, N_S) = \{s_i \mapsto s'_1 \wedge s'_2, s'_1 \mapsto \phi_1, s'_2 \mapsto \phi_2\}$, with $s'_1, s'_2 \in N_S$
  - if $\phi^S_{s_i} = \neg\phi$, then $\mathrm{fold}(s_i, \phi^S_{s_i}, N_S) = \{s_i \mapsto \neg s', s' \mapsto \phi\}$ with $s' \in N_S$
  - if $\phi^S_{s_i} =\geq_n r\phi$, then $\mathrm{fold}(s_i, \phi^S_{s_i}, N_S) = \{s_i \mapsto \geq_n r\ s', s' \mapsto \phi\}$, with $s' \in N_S$

For any $s \in \mathrm{dom}(S)$, let $\phi^{\mathrm{norm}(S)}_s$ designate $(\mathrm{norm}(S))(s)$.
Then immediately from the definition of $[\![\ ]\!]$, for any graph $G$ and $v \in V_G$,
$[\![\phi^S_s(v)]\!]^{G,S} = [\![\phi^{\mathrm{norm}(S)}_s(v)]\!]^{G,\mathrm{norm}(S)}$.
In addition, if no constraint in $\mathrm{range}(S)$ contains a property path expression, then no constraint in $\mathrm{range}(\mathrm{norm}(S))$ contains a property path expression.
Now let $\mathcal{S} = S_0, S_1, ..$ be the sequence defined by $S_0 = S$, and $S_{k+1} = \mathrm{norm}(S_k)$.
And let $s_k \in \mathrm{dom}(S_k)$ be the shape name selected when computing $\mathrm{norm}(S_k)$, i.e. such that $S_{k+1} = S_k|_{\mathrm{dom}(S_k)\setminus\{s_k\}} \cup \mathrm{fold}(s_k, \phi^{S_k}_{s_k}, N_{S_k})$.
From the above definition, $\mathrm{opN}(\phi^{S_{k+1}}_{s_k}) = 1$, and for each fresh $s' \in \mathrm{dom}(S_{k+1}) \setminus \mathrm{dom}(S_k)$, $\mathrm{opN}(\phi^{S_{k+1}}_{s'}) = \mathrm{opN}(\phi^{S_k}_{s_k}) - 1$.
It follows that $\mathcal{S}$ must reach a fixed point $S_n$ of $\mathrm{norm}()$, with $\mathrm{opN}(S_n(s)) \leq 1$ for each $s \in \mathrm{dom}(S_n)$.

Each application of $\mathrm{norm}()$ can introduce at most 2 fresh shape names ($s'_1$ and $s'_2$), therefore $n$ is bounded by $2 \times \sum_{s_i \in \mathrm{dom}(S)} \mathrm{opN}(S(s_i)) = O(|S|)$.

It follows also that the size $|S_n|$ of $S_n$ is bounded by $|S| + 2 \times \sum_{s_i \in \mathrm{dom}(S)} \mathrm{opN}(S(s_i)) = O(|S|)$.
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 $\mathrm{norm}()$, $\mathrm{norm}(S_{k+1})$ 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 $\langle G, S, s_0(v_0)\rangle$ be an instance of VALIDATION, such that no constraints in $\mathrm{range}(S)$ contains a property path expression.
From the above observations, $S_n$ can be computed in time polynomial in $|S|$, and each constraint in $\mathrm{range}(S_n)$ has at most one operator.
By induction on $n$, no constraint in $\mathrm{range}(S_n)$ contains a property path expression.
In addition, for any $v \in V_G$ and $s \in \mathrm{dom}(S)$, by induction on $n$ still,
$[\![\phi_s^S(v)]\!]^{G,S} = [\![\phi_s^{S_n}(v)]\!]^{G,S_n}$.
In particular $[\![\phi_{s_0}^S(v)]\!]^{G,S} = [\![\phi_{s_0}^{S_n}(v)]\!]^{G,S_n}$.
It follows that $\langle G, S, s_0(v_0)\rangle$ is valid iff $\langle G, S_n, s_0(v_0)\rangle$ is valid. $\square$

**Lemma 7.** *Let $\langle G, S, s_0(v_0)\rangle$ be a valid instance of* VALIDATION *such that each constraint in $S$ contains at most one operator and no property path expression, and let $\sigma \in \Sigma^{3,str}_{G,S,s_0(v_0)}$. Then it can be verified in time polynomial in $|\sigma| + |G| + |S|$ that $\sigma \in \Sigma^{3,str}_{G,S,s_0(v_0)}$, even with a binary encoding of cardinality constraints.*

*Proof.* Let $\langle G, S, s_0(v_0)\rangle$ be a valid instance of VALIDATION such that each constraint in $S$ contains at most one operator and no property path expression. And let $\sigma \in \Sigma^{3,\mathrm{str}}_{G,S,s_0(v_0)}$.
For each $s \in \mathrm{dom}(S)$, $\phi_s$ will designate $S(s)$.

In order to verify that $\sigma \in \Sigma^{3,\mathrm{str}}_{G,S,s_0(v_0)}$, it is sufficient to verify that $\sigma s(v) = [\![\phi_s(v)]\!]^{G,S}$ for each $s(v) \in \mathrm{atoms}(G, S)$.

We show below that $\sigma s(v) = [\![\phi_s(v)]\!]^{G,S}$ can be verified in time polynomial in $|\sigma| + |G| + |\phi_s|$. Because $|\mathrm{atoms}(G, S)| = |G| \times |S|$ and $|\phi_s| < |S|$, it follows that $\sigma \in \Sigma^{3,\mathrm{str}}_{G,S,s_0(v_0)}$ can be verified in time in time polynomial in $|\sigma| + |G| + |S|$.

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

- If $\phi_s \doteq \top$, then verify whether $\sigma s(v) = 1$, in $O(|\phi_s| + |\sigma|)$
- If $\phi_s \doteq \neg\top$, then verify whether $\sigma s(v) = 0$, in $O(|\phi_s| + |\sigma|)$
- If $\phi_s \doteq s'$, then verify whether $\sigma s(v) = \sigma s'(v)$, in $O(|\phi_s| + |\sigma|)$
- If $\phi_s \doteq \neg s'$, then verify whether $\sigma s(v) = \sigma s'(v)$, in $O(|\phi_s| + |\sigma|)$
- If $\phi_s \doteq s_1 \wedge s_2$, then verify whether $\sigma s(v) = \min\{\sigma s_1(v), \sigma s_2(v)\}$, in $O(|\phi_s| + |\sigma|)$
- If $\phi_s \doteq (p_1 = p_2)$, then check whether there is a $v_2 \in V_G$ such that either $(v, p_1, v_2) \in E_G$ and $(v, p_2, v_2) \notin E_G$, or $(v, p_2, v_2) \in E_G$ and $(v, p_1, v_2) \notin E_G$. If this is the case, then verify whether $\sigma s(v) = 0$, otherwise verify whether $\sigma s(v) = 1$. The whole procedure is in $O(|\phi_s| + |G|^2 + |\sigma|)$.
- If $\phi_s \doteq \geq_n p.s$ , then let $V' = \{v' \in V_G \mid \in (v, p, v') \in E_G\}$. $V'$ 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 $\phi_s \doteq \geq_n p.\top$, the procedure is identical to the previous one, but without the need to check $\sigma s(v') = 1$.

□

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

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

From Lemma 2.3, it can be transformed in polynomial time into an instance $\langle G_1, S_1, s_0(v_0)\rangle$ such that $\langle G_0, S_0, s_0(v_0)\rangle$ is valid iff $\langle G_1, S_1, s_0(v_0)\rangle$ is valid, and constraints in $S_1$ contain no property path expression.

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

In order to verify that $\langle G_0, S_0, s_0(v_0)\rangle$ is valid, it is sufficient to verify that $\langle G, S, s_0(v_0)\rangle$ is valid.
Let us assume that $\langle G, S, s_0(v_0)\rangle$ is valid.
Then $\Sigma^{3,\text{str}}_{G,S,s_0(v_0)} \neq \emptyset$.
Let $\sigma \in \Sigma^{3,\text{str}}_{G,S,s_0(v_0)}$.
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 $\langle G, S, s_0(v_0)\rangle$.

Now let us assume an oracle which, given a valid instance $\langle G, S, s_0(v_0)\rangle$ of VALIDATION, returns some $\sigma \in \Sigma^{3,\text{str}}_{G,S,s_0(v_0)}$.
From Lemma 7, it can be verified in time polynomial in $|\sigma| + |G| + |S|$ that $\sigma \in \Sigma^{3,\text{str}}_{G,S,s_0(v_0)}$, even with a binary encoding of cardinality constraints.
It follows that VALIDATION is in NP. □

**Proposition 7 (Combined – Lower bound).** VALIDATION *is* PTIME*-hard for* $\mathcal{L}_{n,\wedge,\vee}$

*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 $I_1, \ldots, I_n, I_{\text{true}}$ 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 $C_i$, let $B_1, \ldots, B_\ell$ be the inputs of $C_i$. We create a triple $(C_i, \text{and}, B_j)$ for $1 \leq j \leq \ell$. Similarly, for each OR gate $C_i$, if $B_1, \ldots, B_\ell$ are the inputs of $C_i$, we create a triple $(C_i, \text{or}, B_j)$ for $1 \leq j \leq \ell$. Finally, for each input gate $C_i$ whose value is 1, we create a triple $(C_i, \text{value}, I_{\text{true}})$.

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

Shape $s_i$ is defined depending on the gate $C_i$:

- If $C_i$ is an AND gate with $k$ inputs, then $\phi_{s_i} = \big(\geq_k \text{ and}.(s_1 \vee \cdots \vee s_n)\big) \wedge \geq_1 \text{ name}.I_i$
- If $C_i$ is an OR gate, then $\phi_{s_i} = \big(\geq_1 \text{ or}.(s_1 \vee \cdots \vee s_n)\big) \wedge \geq_1 \text{ name}.I_i$
- If $C_i$ is an input gate, then $\phi_{s_i} = (\geq_1 \text{ name}.I_i) \wedge \geq_1 \text{ value}.I_{\text{one}}$

Let $S = \{s_1 \mapsto \phi_{s_1}, .., s_n \mapsto \phi_{s_n}\}$. Then it is straightforward to check that the circuit evaluates to 1 iff there is a faithful assignment for $\langle G, S, s_n(C_n)\rangle$. And clearly, this reduction can be constructed in LOGSPACE, assuming any reasonable encoding of the monotone circuit value problem. □

**Lemma 8.** *Let $G$ be a graph, and $S$ a set of shape constraint definitions without property path. Then the (unique) minimal fixed-point of $\mathbf{T}^{G,S}$ over $\langle \Sigma^3_{G,S}, \preceq\rangle$ can be computed in time polynomial in $|G| + |S|$.*

*Proof.* In what follows, $\mathbf{T}^{G,S}$ will be abbreviated as $\mathbf{T}$.
From Lemma 3, the mininal fixed-point $\sigma_{\text{minFix}}$ of $\mathbf{T}^{G,S}$ over $\langle \Sigma^3_{G,S}, \preceq\rangle$ must exists and be unique.

We show that it can be computed in polynomial time, as follows:

- Start with the "empty" assignment $\sigma_0$, defined by $\sigma_0 s(v) = 0.5$ for each $s(v) \in \text{atoms}(G, S)$.
- Apply $\mathbf{T}$ to $\sigma_0$ recursively, until a fixed point is reached.

Let $\sigma_{i+1} = \mathbf{T}(\sigma_i)$ for $0 \leq i$.
Because $\langle \Sigma^3_{G,S}, \preceq\rangle$ is a semi-lattice, $\preceq$ admits no cycle over $\Sigma^3_{G,S}$.
From Lemma 2, and together with the fact that $\Sigma^3_{G,S}$ is finite, this guarantees termination, i.e. that the procedure reaches a fixed-point $\sigma_n$ of $\mathbf{T}$.

We now show that the procedure is polynomial in $|G| + |S|$.
Because $\mathbf{T}$ is monotone over $\langle \Sigma^3_{G,S}, \preceq\rangle$ (from Lemma 2), for $0 \leq i < n$, $\sigma_i \prec \sigma_{i+1}$.
Now each $\sigma_i$ ca be viewed as a set $a(\sigma_i)$ of positive and negative atoms, defined by $s(v) \in a(\sigma_i)$ iff $\sigma_i s(v) = 1$, and $\neg s(v) \in a(\sigma_i)$ iff $\sigma_i s(v) = 0$.
So for any $\sigma, \sigma' \in \Sigma^3_{G,S}$, $\sigma \prec \sigma'$ iff $a(\sigma) \subset a(\sigma')$.
In particular, for $0 \leq i < n$, $a(\sigma_i) \subset a(\sigma_{i+1})$ must hold.
Now for any $\sigma \in \Sigma^3_{G,S}$, $a(\sigma_i) \subseteq \text{atoms}(G, S) \cup \{\neg s(v) | s(v) \in \text{atoms}(G, S)\}$.
So as $a(\sigma_i) \subset a(\sigma_{i+1})$, the procedure must terminate after less than $2 \cdot |\text{atoms}(G, S)| = O(|G| \times |S|)$ applications of $\mathbf{T}$, i.e. $n = O(|G| \times |S|)$.

Finally, we show correctness, i.e. $\sigma_n = \sigma_{\mathit{minFix}}$.
Because $\sigma_n$ is a fixed-point of $\mathbf{T}$, and because $\sigma_{\mathit{minFix}}$ is the unique minimal fixed-point of $\mathbf{T}$ over $\langle \Sigma^3_{G,S}, \preceq\rangle$, $\sigma_{\mathit{minFix}} \preceq \sigma_n$ must hold.
The other direction holds by induction on $n$:

- base case $n = 0$.
  $\sigma_0 \preceq \sigma$ holds for any $\sigma \in \Sigma^3_{G,S}$.
  In particular, $\sigma_0 = \sigma_n \preceq \sigma_{\mathit{minFix}}$.
- inductive case $n = i + 1$.
  By IH, $\sigma_i \preceq \sigma_{\mathit{minFix}}$.
  So from Lemma 2, $\mathbf{T}(\sigma_i) \preceq \mathbf{T}(\sigma_{\mathit{minFix}})$, i.e. $\sigma_n \preceq \sigma_{\mathit{minFix}}$.

$\square$

**Proposition 8 (Combined – Upper bound).** VALIDATION *is in* PTIME *for* $\mathcal{L}_{n,\wedge,\vee,r,\mathit{EQ}}$

*Proof.* Let $\langle G', S', s_0(v_0)\rangle$ be an instance of VALIDATION. From Lemma 2.3, 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. In addition, from the construction of $S$ in the proof of Lemma 2.3, if each definition in $S'$ is in $\mathcal{L}_{n,\wedge,\vee,r,\mathit{EQ}}$, then each definition in $S$ is in $\mathcal{L}_{n,\wedge,\vee,\mathit{EQ}}$. Therefore it is sufficient to show that VALIDATION is in PTIME for $\mathcal{L}_{n,\wedge,\vee,\mathit{EQ}}$.

Now let $\langle G, S, s_0(v_0)\rangle$ be an instance of VALIDATION such that each definition in $S$ is in $\mathcal{L}_{n,\wedge,\vee,\mathit{EQ}}$.

From Lemma 3, the minimal fixed-point $\sigma_{\mathit{minFix}}$ of $\mathbf{T}^{G,S}$ is unique, and $\sigma_{\mathit{minFix}} \in \Sigma^{3,\text{cst}}_{G,S}$.
In addition, from Lemma 8, $\sigma_{\mathit{minFix}}$ can be computed in time polynomial in $|G| + |S|$.

Now we have 3 cases to consider:

- $\sigma_{\mathit{minFix}} s_0(v_0) = 1$.
  Because $\sigma_{\mathit{minFix}} \in \Sigma^{3,\text{cst}}_{G,S}$, $\langle G, S, s_0(v_0)\rangle$ is valid.
- $\sigma_{\mathit{minFix}} s_0(v_0) = 0$.
  $\sigma_{\mathit{minFix}}$ is a minimal fixed-point of $\mathbf{T}^{G,S}$, i.e. $\sigma_{\mathit{minFix}} \in \Sigma^{3,\text{fix}}_{G,S}$, and for each $\sigma \in \Sigma^{3,\text{fix}}_{G,S}$, $\sigma_{\mathit{minFix}} \preceq \sigma$.
  So for each $\sigma \in \Sigma^{3,\text{fix}}_{G,S}$, $\sigma s_0(v_0) = 0$.
  Now from Proposition 1, $\langle G, S, s_0(v_0)\rangle$ is valid iff there is a $\sigma \in \Sigma^{3,\text{fix}}_{G,S}$ such that $\sigma s_0(v_0) = 1$.
  It follows that $\langle G, S, s_0(v_0)\rangle$ is invalid.
- $\sigma_{\mathit{minFix}} s_0(v_0) = 0.5$.
  Consider the assignment $\sigma \in \Sigma^2_{G,S}$ defined by $\sigma s(v) = 1$ if $\sigma_{\mathit{minFix}} s(v) = 0.5$, and $\sigma s(v) = \sigma_{\mathit{minFix}} s(v)$ otherwise.
  Because $\sigma \in \Sigma^2_{G,S}$, from the definition of $[\![]\!]^{v,G,\sigma}$, for any $s(v) \in \text{atoms}(G, S)$, $[\![\phi_s]\!]^{v,G,\sigma} \neq 0.5$.
  In addition, from the construction of $\sigma$, for any $s(v) \in \text{atoms}(G, S)$, $\sigma s(v) = 0$ iff $\sigma_{\mathit{minFix}} s(v) = 0$.
  We show below that $[\![\phi_s]\!]^{v,G,\sigma} = 0$ iff $[\![\phi_s]\!]^{v,G,\sigma_{\mathit{minFix}}} = 0$.
  It follows that $\sigma s(v) = 1$ iff $[\![\phi_s]\!]^{v,G,\sigma} = 1$, so $\sigma \in \Sigma^{2,\text{fix}}_{G,S}$.
  Then as $\sigma s_0(v_0) = 1$, $\sigma \in \Sigma^{2,\text{str}}_{G,S}$.
  Therefore $\langle G, S, s_0(v_0)\rangle$ is valid.

  So we only need to show that for any $s(v) \in \text{atoms}(G, S)$, $[\![\phi_s]\!]^{v,G,\sigma} = 0$ iff $[\![\phi_s]\!]^{v,G,\sigma_{\mathit{minFix}}} = 0$.
  The left direction is straightforward from the observation that $\sigma_{\mathit{minFix}} \preceq \sigma$ (from the

construction of $\sigma$).
For the right direction, we show that a stronger property holds: for any $s(v) \in \mathrm{atoms}(G, S)$, and for any $\phi \in \mathcal{L}_{n,\wedge,\vee,EQ}$, if $[\![\phi]\!]^{v,G,\sigma} = 0$, then $[\![\phi]\!]^{v,G,\sigma_{minFix}} = 0$.
By induction on $\phi$:

- $\phi \doteq \top$.
  Then $[\![\phi]\!]^{v,G,\sigma} \neq 0$.
- $\phi = v'$.
  If $[\![\phi]\!]^{v,G,\sigma} = 0$, then $v \neq v'$, therefore $[\![\phi]\!]^{v,G,\sigma_{minFix}} = 0$.
- $\phi \doteq \mathrm{EQ}(p_1, p_2)$.
  If $[\![\phi]\!]^{v,G,\sigma} = 0$, then there is a $v' \in V_G$ s.t. $(v, p_1, v') \in E_G$ and $(v, p_2, v') \notin E_G$, or s.t. $(v, p_2, v') \in E_G$ and $(v, p_1, v') \notin E_G$.
  In both cases, $[\![\phi]\!]^{v,G,\sigma_{minFix}} = 0$.
- $\phi \doteq \phi_1 \wedge \phi_2$.
  If $[\![\phi]\!]^{v,G,\sigma} = 0$, then $[\![\phi_1]\!]^{v,G,\sigma} = 0$ or $[\![\phi_2]\!]^{v,G,\sigma} = 0$.
  So by IH, $[\![\phi_1]\!]^{v,G,\sigma_{minFix}} = 0$ or $[\![\phi_2]\!]^{v,G,\sigma_{minFix}} = 0$.
  Therefore $[\![\phi]\!]^{v,G,\sigma_{minFix}} = 0$.
- $\phi \doteq \phi_1 \vee \phi_2$.
  If $[\![\phi]\!]^{v,G,\sigma} = 0$, then $[\![\phi_1]\!]^{v,G,\sigma} = 0$ and $[\![\phi_2]\!]^{v,G,\sigma} = 0$.
  So by IH, $[\![\phi_1]\!]^{v,G,\sigma_{minFix}} = 0$ and $[\![\phi_2]\!]^{v,G,\sigma_{minFix}} = 0$.
  Therefore $[\![\phi]\!]^{v,G,\sigma_{minFix}} = 0$.
- $\phi \doteq \geq_n p.\phi'$.
  Let $\mathrm{succ}(v) = \{v' | (v, p, v') \in E_G\}$.
  If $[\![\phi]\!]^{v,G,\sigma} = 0$, then $|\,\mathrm{succ}(v)| - |\{v' \in \mathrm{succ}(v) | [\![\phi']\!]^{v',G,\sigma} = 0\}| < n$.
  And by IH, for each $v' \in \mathrm{succ}(v)$, if $[\![\phi']\!]^{v',G,\sigma} = 0$, then $[\![\phi']\!]^{v',G,\sigma_{minFix}} = 0$.
  So $|\,\mathrm{succ}(v)| - |\{v' \in \mathrm{succ}(v) | [\![\phi']\!]^{v',G,\sigma_{minFix}} = 0\}| < n$.
  Therefore $[\![\phi]\!]^{v,G,\sigma_{minFix}} = 0$.

□

## Bibliography

[1] E. V. Kostylev, J. L. Reutter, M. Romero, and D. Vrgoč. SPARQL with property paths. In *ISWC*, pages 3–18. Springer, 2015.