TU WIEN Informatics

Magic Shapes for SHACL Validation

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Logik und Computation

eingereicht von

Bianca Löhnert, BSc, BA
Matrikelnummer 01525825

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Associate Prof. Dr.techn. Magdalena Ortiz, MSc
Mitwirkung: Dr. Shqiponja Ahmetaj, Msc
Privatdoz. Dr.techn. Mantas Šimkus, Msc

Wien, 21. April 2022

Bianca Löhnert

Magdalena Ortiz

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

TU WIEN Informatics

Magic Shapes Algorithm for SHACL Validation

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Logic and Computation

by

Bianca Löhnert, BSc, BA
Registration Number 01525825

to the Faculty of Informatics

at the TU Wien

Advisor: Associate Prof. Dr.techn. Magdalena Ortiz, MSc
Assistance: Dr. Shqiponja Ahmetaj, Msc
Privatdoz. Dr.techn. Mantas Šimkus, Msc

Vienna, 21st April, 2022

Bianca Löhnert | Magdalena Ortiz

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Bianca Löhnert, BSc, BA

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 21. April 2022

Bianca Löhnert

Danksagung

Ich möchte mich bei meiner Betreuerin Prof. Magdalena Ortiz und meinen Mitbetreuern Dr. Shqiponja Ahmetaj und Dr. Mantas Simkus für die guten Ideen, die interessanten Diskussionen und das regelmäßige Feedback bedanken. Ich habe durch das Arbeiten an der Masterarbeit sehr viel gelernt.

Zudem danke ich der Fakultät für Informatik für das Stipendium, das mich bei meiner Masterarbeit finanziell unterstützt hat.

Außerdem möchte ich mich auch bei meinen Eltern, meinem Bruder Bernhard und Lukas für ihre Unterstützung bedanken, auf die ich immer zählen kann.

Acknowledgements

I would like to thank my supervisor Prof. Magdalena Ortiz and my co-supervisors Dr. Shqiponja Ahmetaj and Dr. Mantas Šimkus for the good ideas, interesting discussions and frequent feedback. I learned a lot while working on the master thesis.

Additional thanks to the Faculty of Informatics for the scholarship, that supported me financially with my master thesis.

Further, I would like to thank my parents, my brother Bernhard and Lukas for their support, which I can always count on.

Kurzfassung

Für die Validierung von RDF Graphen hat das W3C mit der *Shapes Constraint Language* (SHACL) einen Standard definiert. Für eine erfolgreiche Anwendung von SHACL, selbst auf sehr große RDF Graphen, werden Methoden benötigt, die eine effiziente Validierung ermöglichen. Ansätze für eine effiziente Validierung von SHACL existieren bereits, aber diese optimierten Validierer konzentrieren sich auf die "tractable" Fragmente der SHACL Sprache und unterstützen nicht die uneingeschränkte Interaktion zwischen Rekursion und Negation in Shapes Graphen. Bei Vorhandensein von Rekursion und Negation ist ein *Ziel-orientierter* Ansatz, wie bestehende Validatoren ihn verfolgen, nicht anwendbar, da dies die Betrachtung des gesamten Datengraphens und damit eine globale Berechnung erfordert. Dies ist auch erforderlich wenn die Targets des Shapes Graphen nur einen Teil des Datengraphen adressieren, der von der Interaktion zwischen Rekursion und Negation nicht betroffen ist. Neben der Tatsache, dass das Zusammenspiel von Rekursion und Negation die Validierung rechenintensiv macht, können bei der globalen Validierung der Graphen Inkonsistenzen auftreten, die aber für die Validierung der Ziele des Shapes Graphen möglicherweise irrelevant sind.

Aus diesem Grund beschäftigt sich diese Masterarbeit mit der Validierung von SHACL bei uneingeschränkter Interaktion von Rekursion und Negation. Zu diesem Zweck wird der *Magic Shapes Algorithmus* vorgestellt, der als Eingabe einen Shapes Graphen erhält und einen *Magic* Shapes Graphen ausgibt, der ebenfalls dem Standard von SHACL entspricht. Der Magic Shapes Graph enthält nur die für die Validierung der Targets relevanten Constraints. Diese Constraints werden um *magic* Shape Constraints erweitert, die den RDF Graphen während der Validierung auf die notwendigen Knoten reduzieren. Im Fall, dass keine Inkonsistenzen durch den RDF Graph und der Interaktion von Rekursion und Negation entstehen, ist das Ergebnis der Validierung für den Eingangs Shapes Graphen, sowie für die magic Variante äquivalent. Andernfalls ermöglicht der Magic Shapes Algorithmus eine Ziel-orientierte Validierung und lässt Inkonsistenzen, die das Validieren der fokusierten Knoten nicht betreffen außen vor. Im Rahmen der Masterarbeit wurde der Algorithmus implementiert und Experimente mit dem existierenden Validierer SHACL-ASP durchgeführt.

Abstract

For the validation of RDF graphs, the W3C has defined a standard in form of the *Shapes Constraint Language* (SHACL). However, for successful adoption of SHACL methods are needed that enable efficient validation even on very large RDF graphs. Approaches for efficient validation of SHACL already exist, but these optimized validators focus on the tractable fragments of the SHACL language and do not support the unrestricted interaction between recursion and negation in shapes graphs. In presence of recursion and negation, a *target-oriented* approach, as existing validators pursue, is not applicable, since this requires considering the whole data graph and thus global computation. This is also required even when the targets of the shapes graph address only some part of the data graph. Besides the fact that the interaction of recursion and negation makes the validation computationally costly, inconsistencies can arise when validating the graphs globally, even though these might be irrelevant for validating the targets of the shapes graph.

For this reason, the aim of this master thesis is the validation of SHACL in presence of unrestricted interaction of recursion and negation. To this end, the *magic shapes algorithm* is presented, which receives as input a shapes graph and outputs an *magic* shapes graph that conforms to the standard of SHACL. The magic shapes graph contains only the constraints relevant for validating the targets. These constraints are extended with *magic* shape constraints, which also reduce the RDF graph to the necessary nodes during validation. In case that no inconsistency arise from the RDF graph and the interaction of recursion and negation, the results of the validation are equivalent for the input shapes graph and the magic variant. Otherwise, the magic shapes algorithm allows for target-oriented validation and leaves out inconsistencies that do not affect the validation of the focused nodes. As part of the master thesis, the algorithm was implemented and experiments were performed with the existing validator SHACL-ASP.

Contents

CHAPTER 1

Introduction

The *Resource Description Framework*[1] (RDF) is a key technology of the Semantic Web, that represents data via triples to connect arbitrary objects [HO09]. Such a set of triples is called a *data graph*. Over time the demand for quality checks like correctness and completeness of data graphs in RDF has emerged [CRS18]. An approach to validate RDF data graphs is the *SHapes Constraint Language*[2] (SHACL), which became a W3C recommendation in 2017. The SHACL language provides the ability to define a set of constraints structured as *shapes*, which are one part of so-called *shapes graphs*. A shapes graph also contains a set of *targets* that specifies the nodes of the data graph to be considered when validated against the shape constraints. For illustration, consider the following data graph G and a shapes graph (C, T), where C is a set of shape constraints and T the targets:

$$G = \{married(Sissi, Franz)\}$$
$$C = \{\mathsf{Queen} \leftarrow \exists married.\top \wedge \exists has.crown\}$$
$$T = \{\mathsf{Queen}(Sissi)\}$$

Further, Queen is a shape name, *married* and *has* are data predicates, i.e. properties, and *crown* is a constant. The target focuses on the node *Sissi* of the data graph and validates whether *Sissi* is a Queen or not. According to the set of shape constraints an individual is validated as Queen if it is married and has a crown. Since *Sissi* does not have a *crown* in G, the data graph does not validate (C, T), but the extended graph $G \cup \{has(Sissi, crown)\}$ does.

The W3C recommendation specifies the syntax for shape constraints and defines a semantics for the validation of RDF graphs against shapes graphs. This standard allows shapes to refer to other shapes, which can lead to cyclic dependencies, this is called

[1] https://www.w3.org/TR/rdf11-concepts/
[2] https://www.w3.org/TR/shacl/

recursion. However, the W3C recommendation leaves the semantics for validation of recursive shape constraints explicitly undefined and up to validation engines:

> *"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."*

W3C provides a list of existing SHACL validators[3]. These listed implementations validate according to the defined semantics of the W3C recommendation for SHACL, but since the semantics for validating recursive shapes graphs is not defined yet, these validators do not operate according to a specified semantics for recursive graphs. The problem of recursive SHACL is discussed by Corman et al. in [CRS18]. They come to the conclusion that the complexity of the validation problem for the language of full SHACL is NP-complete, i.e. non-tractable. The complexity of the validation problem becomes intractable, because the unrestricted interaction of recursion and negation requires global computation on the data graph even if the targets only consider part of the data graph. Another problem that arises from the need for a global computation is that inconsistencies could prevent validation, even if the inconsistencies are unrelated to the target(s). A data graph is inconsistent with a shapes graph if there exists no assignment from nodes of the data graph to shape names of the shapes graph, such that the shape constraints are validated to true. For demonstration, let's consider the following example of a data graph G and a shapes graph (C, T), where:

$$G = \{crownedBy(Sissi, Archbishop), crownedBy(Tim, Tim)\}$$
$$C = \{\mathsf{Crowned} \leftarrow \exists crownedBy.\neg\mathsf{Crowned}\}$$
$$T = \{\mathsf{Crowned}(Sissi)\}$$

The constraints in C involve recursion and negation, which might give rise for inconsistencies during validation. The target $\mathsf{Crowned}(Sissi)$ satisfies the shape constraints for Crowned, since the individual *Sissi* is crowned by the *Archbishop*, who himself is not Crowned. Although at first glance, it may seem like the target is valid, this is not the case for the 2-valued semantics proposed in [ACO+20] and [CFRS19b]. The data graph is invalid, since there is no consistent way to assign or not assign shape name Crowned to the individual *Tim*. This result of the validation may not be desired, since the part of the data graph that causes the inconsistency is irrelevant to the target $\mathsf{Crowned}(Sissi)$. This may be a toy example, but in huge RDF graphs it is not unlikely that erroneous facts occur.

Goal of the Thesis

Therefore, the aim of this thesis is SHACL validation in the presence of unrestricted recursion and negation. For that purpose an algorithm is proposed, which adapts the

[3] https://w3c.github.io/data-shapes/data-shapes-test-suite/

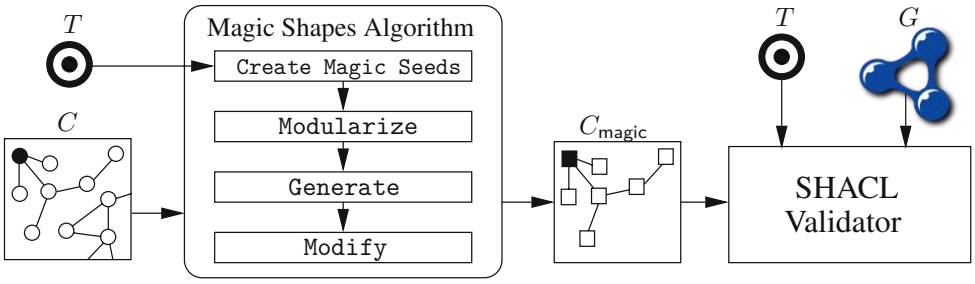


Figure 1.1: Workflow of SHACL validation with the Magic Shapes Algorithm. A input shapes graph (C, T) is transformed into a (typically smaller) magic shapes graph (C_{magic}, T), that contains only constraints relevant for validating the targets. A data graph G can be validated against the magic shapes graph with existing validators.

ideas of the *Magic Sets* optimization technique from Datalog (see [BMSU85, AFGL12]). The Magic Sets technique is a well-known method for efficient query-answering. It simulates top-down query evaluation while retaining the benefits of bottom-up validation. In [FGL07] Faber et al. proposed a generalization of the Magic Set technique for Datalog programs with unstratified negation (Datalog$^{\neg}$), which serves as a foundation for the *Magic Shapes Algorithm* for SHACL. Since the Magic Sets algorithm evaluates in a query-oriented manner and only considers the part of the Datalog program relevant to the query, it can be used for inconsistency-tolerant evaluation [FGL07]. The algorithm also provides an optimization technique, since the algorithm outputs a program containing the relevant parts necessary for evaluating the query. Thus, the research question to be answered by this master thesis is,

"Can the ideas underlying the Magic Sets transformation in Datalog be applied to SHACL shapes graphs and be used to improve the validation of unrestricted SHACL?"

This question pursues the aim of an algorithm to improve the validation of SHACL based on the Magic Set technique from logic programming [FGL07], that

- allows efficient, target-guided validation even in the presence of recursion and negation,
- allows inconsistency tolerant validation, i.e. validation in the presence of inconsistencies that do not affect the validation of the target(s), and
- can be used as a pre-processing step to increase the applicability of existing validators for tractable SHACL (non-recursive fragment, recursive fragment without negation in cycles).

Contribution

- Section 3 of this thesis proposes the **magic shapes algorithm**, that takes as input a shapes graph (C, T) and produces a potentially smaller *magic* shapes graph (C_{magic}, T). Figure 1.1 shows that the input shapes graph undergoes four steps. In `Create Magic Seeds` *magic* constraints are created based on the targets. Procedure `Modularize` identifies the relevant constraints of the shapes graph for validation. For each of these constraints additional *magic* constraints are generated by `Generate`. Finally in step `Modify` the constraints identified as relevant to the targets are enhanced by magic shape names, that are added to the magic shapes graph by `Generate`. The resulting shapes graph (C_{magic}, T) can then be passed to existing validators.

- In Chapter 4 we show the **correctness** of the magic shapes algorithm. The main theorem states, that whenever the data graph is consistent with (C, T) and a global assignment exists, the validation of the input shapes graph and the magic output graph are equivalent.

- In Chapter 5 we discuss the **inconsistency-tolerant validation**. In case the data graph is not consistent with the input, it might still be validated by the magic shapes graph. This occurs if the part causing the inconsistency is not relevant for validating the targets. The magic shapes algorithm thus enables to perform an effective inconsistency-tolerant/3-valued validation.

- The **evaluation** of the magic shapes algorithm shows that it significantly improves the validation of the SHACL-ASP validator and enables inconsistency-tolerant validation with existing SHACL validators.

State-of-the-art

Currently there exists only two approaches for validation of recursive SHACL, described in the works "Stable model semantics for recursive SHACL" [ACO+20] and "Validating SHACL Constraints over a SPARQL Endpoint" [CFRS19b]. The former approach relies on the stable model semantics for logic programming. The SHACL-ASP[4] prototype implements this approach and covers the language of full SHACL. The latter work proposes a semantic that makes use of classical logic, for that first all relevant information for the validation of the shapes graph is obtained from the RDF graph using SPARQL, and then rules are generated according to specific patterns, considering the obtained information and the targets, in a next step the generated rules are handed to a SAT-solver. Further, this work presents two additional methods for the validation of tractable SHACL fragments. In the non-recursive case Corman et al. have shown that shapes graph can be expressed using a single SPARQL query, i.e., non-recursive shapes graph can be validated by means of SPARQL. The second method is an optimized algorithm for the three

[4] https://github.com/medinaandresel/shacl-asp

tractable fragments $\mathcal{L}_{non-rec}$ (non recursive), $\mathcal{L}^{+}_{\vee}$ (disjunction as native operator, but no negation) and $\mathcal{L}^{s}$ (no recursion with negation). The optimized version of the algorithm operates similar to the one for full SHACL, but this one selects the shapes considering the targets for further processing, and moreover there is no need for a SAT-solver, as it performs the validation on-the-fly. In the paper [CFRS19b] Corman et al. also claim that the order in which shapes are selected leaves room for optimization. Moreover, the prototype used to perform the experiments in this paper implements this optimized version of the algorithm that works only for SHACL fragments where the interaction between recursion and negation is restricted, otherwise it might be necessary to consider other parts of the shapes graph to maintain the correctness of the validation.

A very recent work regarding a more efficient SHACL validation by rearranging the order of shapes in a graph is Trav-SHACL [FRV21]. In addition, the SHACL engine detects invalid entities earlier by rewriting targets and constraint queries. Figuera et al. [FRV21] compared the execution times of Trav-SHACL to shacl2sparql, which is a prototype implementing the optimized algorithm of [CFRS19b], and came to the conclusion that Trav-SHACL performs up to a factor of 29.93 better. Like the shacl2sparql validator, Trav-SHACL is limited to the language fragment that restricts the interaction between recursion and negation, and thus does not deal with the full SHACL language either.

Structure of the Thesis

Chapter 2 provides background information in order to understand the proposed optimization technique. RDF as well as the language of SHACL and the notion of validation under the supported and stable model semantics are discussed.

Chapter 3 proposes the magic shapes algorithm, first for the positive language fragment of SHACL (i.e. the language in absence of negation) and in a second step an extended version of the magic shapes algorithm is introduced, which is applicable for the full language of SHACL.

Chapter 4 presents a proof to show the correctness of the proposed magic shapes algorithm.

Chapter 5 gives a definition of an inconsistency-tolerant semantics for both the supported- and stable model semantics. In this chapter it is shown, that the magic shapes technique makes it possible to find an so called *faithful* assignment, which allows to leave shape names undefined.

Chapter 6 outlines the results of the experiments, that show that the technique improves the performance of shacl-asp significantly. Since it is the only implementation that supports the full language of SHACL, i.e. includes validation in the presence of inconsistencies, the experiments are performed with this validator.

Chapter 7 concludes the thesis by a summary and an outlook for future work.

CHAPTER 2

Preliminaries

This chapter provides background information necessary to understand the magic shape algorithm in Chapter 3. For this purpose, the W3C recommendations for RDF and SHACL are explained first. In a next section the problem of validating data graph against a shapes graph is addressed.

2.1 Resource Description Framework

The *World Wide Web Consortium*[1] (W3C) declared the *Resource Description Framework*[2] (RDF) a W3C recommendation in 1999. Since then, the RDF formalism has become an essential part of the semantic web for data interchange. The language of RDF consists of triples, which consist of a subject, a predicate and an object. A triple can also be represented in form of a graph, where subjects and objects correspond to nodes and predicates to edges, as shown in Figure 2.1. The recommendation distinguishes

Figure 2.1: Graphical representation of an RDF triple

between three different types of nodes: *IRIs*, *literals* and *blank nodes*. IRIs (short for *Internationalized Resource Identifier*) and literals (values such as strings, numbers and dates) denote *resources*, i.e. something in the "universe of discourse". This means anything in the world can be a resource, e.g. things and abstract concepts. In contrast to IRIs and literals, blank nodes are used to determine that a relation exists, but without giving a reference to a specific resource. Moreover a set of triples corresponds to the

[1] https://www.w3.org/
[2] https://www.w3.org/TR/rdf11-concepts/

designation of an RDF *graph*. An abstract representation of an RDF graph, called a data graph, is given by Definition 1, where $\boldsymbol{N}$ is the set that forms the alphabet for subjects and objects, and $\boldsymbol{P}$ forms the alphabet for predicates. A triple has the form (subject)−[predicate]→(object), where subject, predicate and object can all be of type IRI; subject and object can be a blank node and only objects can be literals. The abstract representation of a triple is $p(v, v')$ where p is the predicate, v the subject, and v' the object.

Definition 1 (Data graph)**.** Let $\boldsymbol{N}$ and $\boldsymbol{P}$ denote infinite, disjoint sets of *nodes*, and *property names*, respectively. A *(data) graph* G is a finite set of atoms of the form $p(v, v')$, where $p \in \boldsymbol{P}$ and $v, v' \in \boldsymbol{N}$. The set of nodes appearing in G is denoted with $V(G)$.

Example 2 illustrates two different representations of the same data graph. One is the abstract syntax used for simplicity in the theoretical part of this thesis, and the other is a syntax defined by the W3C recommendation for RDF, which is commonly used by real-world applications.

Example 2. On the left side is a data graph in the syntax corresponding to Definition 1 and on the right side is the same graph in N-triples[3] syntax for RDF data graphs.

type(HarryPotter,Film).	`:HarryPotter dbo:type dbo:Film.`
starring(HarryPotter,EmmaWatson).	`:HarryPotter dbo:starring :EmmaWatson.`
director(HarryPotter,ChrisColumbus).	`:HarryPotter dbo:director :ChrisColumbus.`
type(ChrisColumbus,Person).	`:ChrisColumbus dbo:type dbo:Person.`
type(EmmaWatson,Person).	`:EmmaWatson dbo:type dbo:Person.`
birthDate(EmmaWatson,1990-04-15).	`:EmmaWatson dbo:birthDate 1990-04-15.`
birthPlace (EmmaWatson,Paris).	`:EmmaWatson dbo:birthPlace :Paris.`

SPARQL Query Language. RDF allows to store all kinds of information in form of triples without any further schema. In order to query the data, efficient query languages are needed, which is why a brief introduction to SPARQL is provided. SPARQL is a recursive acronym and abbreviates the *SPARQL query language for RDF*, which enables the querying of RDF data graphs. The results of these queries are either result sets or RDF graphs, as described in the W3C recommendation for SPARQL[4].

Example 3. The SPARQL query in this example asks for all nodes of type `dbo:Person`. In case of the data graph of Example 2, `:EmmaWatson` and `:ChrisColumbus` are returned as results.

```
Select distinct ?x where {
  ?x dbo:type dbo:Person
}
```

[3] https://www.w3.org/TR/n-triples/
[4] https://www.w3.org/TR/rdf-sparql-query/

2.2 Shape Constraint Language

The *Shape Constraint Language*[5] (SHACL) was established to check the quality of RDF data graphs and became a W3C recommendation in 2017. The SHACL language allows to define sets of constraints in terms of *shapes* and then validate a data graph against a so called *shapes graph*. An important concept of the SHACL language is the notion of a target. This notion narrows the validation to the so called *focus nodes*. The recommendation distinguishes between five different types of targets, i.e. node targets, class-based targets, implicit class targets, subjects-of targets and objects-of targets, of which only the *node targets* and *class-based targets* are taken into account in this thesis. But for the theory part of this thesis, we only consider ground targets, since the different types of targets can be converted to ground targets using SPARQL queries. The SHACL recommendation[6] provides a possible definition in SPARQL for each target type. Next, an example of a SHACL shapes graph in Turtle[7] syntax with a class-based target is given by Example 4.

Example 4. This example is a shapes graph with three shapes, namely `PersonShape`, `MusicianShape` and `LocationShape`. For a node to be validated as a `LocationShape` it has to have at least a `country` path. `PersonShape` has to have exactly one birth place that is a `LocationShape`, exactly one birth date and at least one birth name. A node validates as `MusicianShape`, if it plays at least one instrument, further `MusicianShape` refers to `PersonShape`, which means that a node must additionally satisfy all constraints of `PersonShape` in order to validate `MusicianShape` as true. Additionally `MusicianShape` has a whole class as target defined, i.e. `sh:targetClass dbo:Person`. This means all nodes of type `dbo:Person` fall into the focus of validation.

```
:PersonShape a sh:NodeShape;
  sh:property [
    sh:path dbo:birthPlace;
    sh:minCount 1;
    sh:maxCount 1;
    sh:node :LocationShape
  ] ;
  sh:property [
    sh:path dbo:birthDate;
    sh:minCount 1;
    sh:maxCount 1
  ] ;
  sh:property [
    sh:path dbo:birthName;
    sh:minCount 1
  ] .
```

```
:MusicianShape a sh:NodeShape;
  sh:targetClass dbo:Person;
  sh:node :PersonShape;
  sh:property [
    sh:path dbo:instrument;
    sh:minCount 1
  ] .

:LocationShape a sh:NodeShape;
  sh:property [
    sh:path dbo:country;
    sh:minCount 1
  ].
```

[5] https://www.w3.org/TR/shacl/
[6] https://www.w3.org/TR/shacl/#targets
[7] https://www.w3.org/TR/turtle/

In contrast to the Turtle syntax, we provide the definition for an abstract syntax of the SHACL language in the W3C specification in the following.

Definition 5 (Syntax of SHACL)**.** Let $\boldsymbol{S}$ be an infinite set of *shape names*, disjoint from $\boldsymbol{N}$ and $\boldsymbol{P}$. A *shape atom* is an expression of the form $s(a)$, where $s \in \boldsymbol{S}$ and $a \in \boldsymbol{N}$. A *path expression* E is a regular expression build from the operators $*$, $\cdot$, $\cup$, and symbols p or expressions p^-, where $p \in \boldsymbol{P}$. A *(shape) expression* is an expression ϕ of the form:

$$\phi := \top \mid s \mid a \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \neg\phi \mid \geq_n E.\phi \mid E = E',$$

where $s \in \boldsymbol{S}$, $a \in \boldsymbol{N}$, $n \in \mathbb{N}$, and E, E' are path expressions. The usual abbreviations $\exists E.\phi$ for $\geq_1 E.\phi$, and $\leq_n E.\phi$ for $\neg(\geq_{n+1} E.\phi)$ are used. A *(shape) constraint* is of the form $s \leftarrow \phi$, where $s \in \boldsymbol{S}$ and ϕ is a shape expression. In this thesis, s is also called the *head* and ϕ the *body* of a constraint.
A target set (or simply *targets*) is a set of shape atoms of the form $s(a)$ where $s \in \boldsymbol{S}$ and $a \in \boldsymbol{N}$. A *shapes graph* is a pair (C, T), where C is a set of constraints and T is the target set. The definition $\phi_{s,C}$ of a shape name s in a set of constraints C is the disjunction of all shape expressions in the body of a shape constraint in C whose head shape is s. That is, $\phi_{s,C} = \wedge_{s \leftarrow \phi \in C} \phi$. If C is clear from the context, $\phi_{s,C}$ may be simplified as ϕ_s.

Example 6. This shapes graph contains the same constraints and shapes as in Example 4 in formal syntax from Definition 5. For an explanation of the constraint see Example 4.

$$\begin{aligned}
&T = \{\mathsf{MusicianShape}(\mathit{mozart})\}\\
&C = \{\\
&\qquad \mathsf{PersonShape} \leftarrow \quad \geq_1 \mathit{birthPlace}.\mathsf{LocationShape} \wedge\\
&\qquad\qquad\qquad\quad \leq_1 \mathit{birthPlace}.\mathsf{LocationShape} \wedge\\
&\qquad\qquad\qquad\quad \geq_1 \mathit{birthDate}.\top \wedge\\
&\qquad\qquad\qquad\quad \leq_1 \mathit{birthDate}.\top \wedge\\
&\qquad\qquad\qquad\quad \geq_1 \mathit{birthName}.\top,\\
&\qquad \mathsf{MusicianShape} \leftarrow \quad \geq_1 \mathit{instrument}.\top \wedge \mathsf{PersonShape},\\
&\qquad \mathsf{LocationShape} \leftarrow \quad \geq_1 \mathit{country}.\top\\
&\qquad\qquad\qquad\quad \}
\end{aligned}$$

The dependencies between shape names of a set of shape constraints can be represented as a graph, which we call a *shape dependency graph*. With the shape dependency graph it is possible to identify if a shapes graph is recursive. We provide the definition for shape dependency graph, cycles and recursion in the following.

Definition 7 (Shape Dependency Graph, Cycle, Recursion)**.** The *shape dependency graph* of C is a directed graph DG_C, where there is an edge (s, s') from node s to node s' if there is a constraint $s \leftarrow \phi$ in C such that s' occurs in ϕ.
A *cycle* of DG_C is a sequence of nodes $n_1, \ldots, n_k$, such that $n_1 = n_k$, each n_i for $1 < i < k$ occurs exactly once in S, and (n_i, n_{i+1}) for $(1 \leq i < k)$ is an edge in DG_C.
A shapes graph (C, T) is *recursive* if the shape dependency graph DG_C of C contains a cycle.

$$[\![\top]\!]^I = V(I) \qquad [\![c]\!]^I = \{c\}$$

$$[\![p]\!]^I = \{(v, v') \mid p(v, v') \in I\} \qquad [\![p^-]\!]^I = \{(v, v') \mid p(v', v) \in I\}$$

$$[\![E \cup E']\!]^I = [\![E]\!]^I \cup [\![E']\!]^I \qquad [\![E \cdot E']\!]^I = [\![E]\!]^I \circ [\![E']\!]^I$$

$$[\![E^*]\!]^I = \{(v, v) \mid v \in V(I)\} \cup [\![E]\!]^I \cup [\![E \cdot E]\!]^I \cup [\![E \cdot E \cdot E]\!]^I \cup \cdots$$

$$[\![s]\!]^I = \{v \mid s(v) \in I\} \qquad [\![\neg\phi]\!]^I = V(I) \setminus [\![\phi]\!]^I$$

$$[\![\phi_1 \vee \phi_2]\!]^I = [\![\phi_1]\!]^I \cup [\![\phi_2]\!]^I \qquad [\![\phi_1 \wedge \phi_2]\!]^I = [\![\phi_1]\!]^I \cap [\![\phi_2]\!]^I$$

$$[\![\geq_n E.\phi]\!]^I = \{v \mid |\{(v, v') \in [\![E]\!]^I \text{ and } v' \in [\![\phi]\!]^I\}| \geq n\}$$

$$[\![E = E']\!]^I = \{v \mid \forall v' : (v, v') \in [\![E]\!]^I \text{ iff } (v, v') \in [\![E']\!]^I\}$$

Table 2.1: Evaluation of shape expressions

2.3 SHACL Validation

The W3C recommendation for SHACL provides semantics for the validation of constraints, but since the validation for recursive shapes graphs is explicitly omitted and left to implementations, there have been several approaches to the semantics of validating recursive SHACL graphs. In this section we present different approaches for SHACL validation, namely the *supported model semantics*[CFRS19a] and the *stable model semantics*[ACO$^+$20], followed by a listing of tractable fragments of the SHACL language. But before going into details of the semantics, we first define the notion of an *assignment* and *decorated graph*.

Definition 8 (Assignment, decorated graph)**.** An *assignment* A for a graph G is a set of shape atoms such that $v \in V(G)$ for each $s(v) \in A$. The set $I = G \cup A$ is called a *decorated graph*.

2.3.1 Supported Model Semantics

In order to answer the validation problem for a data graph and a possibly recursive shapes graph Corman et al. proposed a semantics in [CRS18] and Andresel et al. denoted this semantics in [ACO$^+$20] as the *supported model semantics*. In the supported model semantics an RDF data graph validates to true against a shapes graph if the condition in Definition 9 is satisfied given the evaluation function for complex shape constraints in Table 2.1. Throughout this thesis we assume that when validating a shapes graph (C, T) against a data graph G, constants that occur in (C, T) also appear in G.

Definition 9 (Supported Model Semantics)**.** Given a set of shape constraints C, a decorated graph I is called a *supported model* of C, if $[\![s]\!]^I = [\![\phi_s]\!]^I$ for all $s \leftarrow \phi_s \in C$.

The following examples illustrate validation under the supported model semantics.

Example 10. Let (C, T) be a shapes graph and G be a data graph defined as follows:

$$
\begin{aligned}
C = \{&\mathsf{Queen} \leftarrow \exists \mathit{gender}.\mathit{female} \wedge (\exists \mathit{married}.\mathsf{King} \vee \exists \mathit{has}.\mathit{crown}),\\
&\mathsf{King} \leftarrow \exists \mathit{gender}.\mathit{male} \wedge (\exists \mathit{married}.\mathsf{Queen} \vee \exists \mathit{has}.\mathit{crown})\}\\
T = \{&\mathsf{King}(\mathit{Franz})\}\\
G = \{&\mathit{married}(\mathit{Harry}, \mathit{Meghan}), \mathit{married}(\mathit{Meghan}, \mathit{Harry}),\\
&\mathit{married}(\mathit{Franz}, \mathit{Sissi}), \mathit{married}(\mathit{Sissi}, \mathit{Franz}),\\
&\mathit{gender}(\mathit{Harry}, \mathit{male}), \mathit{gender}(\mathit{Meghan}, \mathit{female})\\
&\mathit{gender}(\mathit{Franz}, \mathit{male}), \mathit{gender}(\mathit{Sissi}, \mathit{female})\\
&\mathit{has}(\mathit{Franz}, \mathit{crown})\}
\end{aligned}
$$

The constraint in C state that a Queen is a female, who is either be married to a King or has a crown. Equivalent to this a King is a male that is either married to a Queen or has a crown. Now, consider the following shape assignments:

$$
\begin{aligned}
A_1 &= \{\mathsf{King}(\mathit{Harry}), \mathsf{Queen}(\mathit{Meghan}), \mathsf{King}(\mathit{Franz}), \mathsf{Queen}(\mathit{Sissi})\}\\
A_2 &= \{\mathsf{King}(\mathit{Franz}), \mathsf{Queen}(\mathit{Sissi})\}
\end{aligned}
$$

The decorated graphs $I_i = G \cup A_i$ with $1 \leq i \leq 2$ both satisfy the condition $[\![\phi_s]\!]^{I_i} = [\![s]\!]^{I_i}$ for each shape name s occurring in C, and hence, I_1 and I_2 are a supported model of C.

2.3.2 Stable Model Semantics

The stable model semantics for SHACL validation proposed in [ACO+20] refines the supported model semantics from [CRS18] by requiring that the validation of the targets has a well-founded justification.

Definition 11 (Level assignment)**.** Let I be a supported model. A *level assignment* for I is a function level that maps tuples in $\{(\phi, v) \mid v \in [\![\phi]\!]^I\}$ to integers, and satisfies the following conditions:

(i) $\mathit{level}(\phi_1 \wedge \phi_2, v) = \mathit{max}(\{\mathit{level}(\phi_1, v), \mathit{level}(\phi_2, v)\})$

(ii) $\mathit{level}(\phi_1 \vee \phi_2, v) = \mathit{min}(\{\mathit{level}(\phi_1, v), \mathit{level}(\phi_2, v)\})$

(iii) $\mathit{level}(\geq_n E.\phi, v)$ is the smallest $k \geq 0$ for which there exist n nodes $v_1, v_2, \ldots, v_n$ such that for all $1 \leq i \leq n$

  (a) $(v, v_i) \in [\![E]\!]^I$, $v_i \in [\![\phi]\!]^I$, and

  (b) $\mathit{level}(\phi, v_i) \leq k$.

Definition 12 (Stable Model Semantics)**.** A decorated graph I is a stable model of a set C of shape constraints, if

(i) I is a supported model of C, and

(ii) there exists a level assignment such that for all $s(v) \in I, level(\phi, v) < level(s, v)$, with $s \leftarrow \phi$ the constraints in C.

Every stable model is by definition also a supported model, but the converse may not be the case. This is shown by the following example.

Example 13. Consider the shapes graph (C, T), the data graph G and its supported models I_1 and I_2 from Example 10. Model I_1 has no level assignment that fulfills the conditions of Definition 12 item (ii), hence it is not a stable models. However, I_2 is a stable models, to show that observe the following level assignment, with $v \in \{Harry, Megan, Franz, Sissi\}$:

$$level(\top, v) = 0 \qquad level(\exists has.crown, Franz) = 0$$
$$level(\mathsf{King}, Franz) = 1 \qquad level(\mathsf{Queen}, Sissi) = 1$$

Note that $level(\phi_s, v) < level(s, v)$ for all $s(v) \in G \cup A_2$ as required.

2.3.3 Brave- and Cautious Validation

Definition 14 (Brave- and cautious validation)**.** A data graph G is *bravely valid* against a shapes graph (C, T) under the supported (or stable) model semantics, if there *exists* an assignment A such that

(i) $G \cup A$ is a supported (or stable) model of C, and

(ii) $T \subseteq A$

A data graph G is *cautiously valid* against a shapes graph (C, T) under the supported (or stable) model semantics, if for *every* stable model $G \cup A$ of C (ii) holds. C is called *consistent* with G if there exists a supported (or stable) model I of C; otherwise C is called *inconsistent* with G under the supported (or stable) model semantics.

Example 15. Let's continue Example 10. The target T is contained in the supported models with assignment A_1 and A_2. Hence, the data graph G bravely and cautiously, validates the shapes graph (C, T) under the supported model semantics. For the stable model semantics there exist only one model, namely $G \cup A_2$. Since $T \subseteq A_2$ the data graph G bravely and cautiously validates the shapes graph (C, T) under the stable model semantics.

In the process of validating a data graph, we look for supported- or stable models. If such a model exists and the targets are a subset of this model, we say the data graph is valid against the shapes graph, otherwise the data graph is not valid. Deciding whether

a data graph is valid with respect to a shapes graph is called the *validation problem*, we provide the definition of it below.

Validation Problem
Input: A graph G and a shapes graph (C, T)
Question: Is G valid against (C, T)?

2.3.4 Normal form for SHACL

To ease the presentation of shapes graphs, in what follows it is assumed that shape constraints occur in normal form, which is defined below.

Definition 16 (Normal form for constraints)**.** A constraint is in *normal form* if it has one of the following forms:

$$\begin{array}{lll} \text{(NF1) } s \leftarrow \top & \text{(NF2) } s \leftarrow a & \text{(NF3) } s \leftarrow E = E' \\ \text{(NF4) } s \leftarrow \neg s' & \text{(NF5) } s \leftarrow s_1 \wedge \cdots \wedge s_n & \text{(NF6) } s \leftarrow \geq_n E.s' \end{array}$$

It was shown in [ACO+20] (Proposition 4.2) that a set of constrains C can be transformed in polynomial time in a normalized set of constraints C' such that for every graph G and target set T, G validates (C, T) iff G validates (C', T) under supported- and stable model semantics. It can be further shown that a model I of (C, T) can be transformed into a model I' of (C', T). Hence, both brave- and cautious validation are preserved.

Example 17. The shapes graph from Example 6 after the transformation in normal form:

$$\begin{aligned} T = \{&\mathsf{MusicianShape} \leftarrow \mathit{Person})\} \\ C' = \{&\mathsf{PersonShape} \leftarrow \mathsf{PS}_1 \wedge \mathsf{PS}_2 \wedge \mathsf{PS}_3 \wedge \mathsf{PS}_4 \wedge \mathsf{PS}_5, \\ &\mathsf{PS}_1 \leftarrow \geq_1 \mathit{birthPlace}.\mathsf{LocationShape}, \\ &\mathsf{PS}_2 \leftarrow \neg \mathsf{PS}_{21}, \\ &\mathsf{PS}_{21} \leftarrow \geq_2 \mathit{birthPlace}.\mathsf{LocationShape}, \\ &\mathsf{PS}_3 \leftarrow \geq_1 \mathit{birthDate}.\top, \\ &\mathsf{PS}_4 \leftarrow \neg \mathsf{PS}_{41}, \\ &\mathsf{PS}_{41} \leftarrow \geq_2 \mathit{birthDate}.\top, \\ &\mathsf{PS}_5 \leftarrow \geq_1 \mathit{birthName}.\top, \\ &\mathsf{MusicianShape} \leftarrow \mathsf{MS}_1 \wedge \mathsf{MS}_2, \\ &\mathsf{MS}_1 \leftarrow \geq_1 \mathit{instrument}.\top, \\ &\mathsf{MS}_2 \leftarrow \mathsf{PersonShape}, \\ &\mathsf{LocationShape} \leftarrow \geq_1 \mathit{country}.\top\} \end{aligned}$$

Fragment	Data	Constraint	Combined
$\mathcal{L}$	NP-complete	NP-complete	NP-complete
$\mathcal{L}_{\vee}^{+}$	P	P	P-complete
$\mathcal{L}_{s}$	NL-complete	—	—
$\mathcal{L}_{non-rec}$	P-complete	—	—

Table 2.2: Complexity of the validation problem for different SHACL language fragments from [CFRS19b] and [CRS18]

2.3.5 Tractable Fragments

In this section we discuss the complexity of the validation problem. In [CRS18] Corman et al. investigates the complexity of validation for the supported model semantics. Since the size of G and C can grow they studied two different kind of complexity, namely *data complexity* (for a fixed C) and *constraint complexity* (for a fixed G). They come to the conclusion that the data-, constraint- and combined complexity of solving the validation problem for the full language of SHACL, which we denote as $\mathcal{L}$ is NP-complete. Further, they showed that validation is in P (data-, constraint- and combined complexity) for the language fragment, that disallows negation, but allows disjunction as a native operator, denoted as $\mathcal{L}_{\vee}^{+}$.

In [CFRS19b] Corman et al. provide data complexities for further fragments of the SHACL language. They argue that in database literature it is usual to measure complexity in size of the data graph, since the size of the data graph is likely to grow faster. Therefore, Corman et al. provide the data complexity only, for the non-recursive fragment $\mathcal{L}_{non-rec}$ and the fragment that restricts the interplay between recursion and negation $\mathcal{L}_{s}$. The validation problem of both language fragments is PTIME-complete. Table 2.2 summarizes the just mentioned complexities for the supported model semantics.

For the stable model semantics Andresel et al. reveal in [ACO+20] that the complexity of solving the validation problem for the full language of SHACL is also NP-complete.

CHAPTER 3

Magic Shapes Algorithm

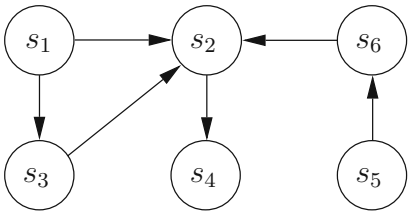
In this chapter we introduce the magic shapes algorithm. For this purpose, we first present the algorithm for the positive language fragment of SHACL and then extend it to support the unrestricted interaction between recursion and negation. The magic shapes algorithm is based on the magic set technique from logic programming and deductive databases proposed in [FGL07] for DATALOG with unstratified negation.

3.1 Algorithm for positive fragment of SHACL

The positive language fragment of SHACL contains all shape constraints of the normal from Definition 17 except negation (NF4). Note that the normal form does not contain explicit disjunction or universal quantification. The general idea of the algorithm is to have a SHACL shapes graph as input and return a new SHACL shapes graph (see Figure 1.1). The produced *magic* shapes graph then contains only constraints relevant for the validation of the targets. Depending on the target set, constraints may be removed from the initial shapes graph, the remaining ones are extended with additional *magic shapes*, which intuitively ensure that during validation the original constraints are only instantiated in the relevant part of the data graph. Before discussing the individual procedures of the algorithm we give the definition of reachable set and module, which rely on the shape dependency graph in Definition 7. For illustration we provide an example of a shape dependency graph below.

Example 18. This example demonstrates what the shape dependency graph looks like for the set C containing the following constraints:

$$s_1 \leftarrow s_2 \wedge s_3 \qquad s_2 \leftarrow \exists r.s_4$$
$$s_3 \leftarrow s_2 \qquad s_4 \leftarrow \exists r.\top$$
$$s_5 \leftarrow s_6 \qquad s_6 \leftarrow \exists r.s_2$$



In a next step the shape names *reachable* from the targets are identified by means of the shape dependency graph. Based on this set of shape names, the relevant constraints for the validation of the targets can then be selected.

Definition 19 (Reachable Set, Module)**.** Given a shapes graph (C, T), where C is in the positive fragment and DG_C is the shape dependency graph of C. Let $\mathit{reach}(C, T)$ be the smallest set of shape names such that:

(i) if $s(v) \in T$, then $s \in \mathit{reach}(C, T)$, and

(ii) if $s(v) \in T$ and there exists an edge from s to s' in DG_C , then $s' \in \mathit{reach}(C, T)$

The set of constraints $C_T = \{s \leftarrow \phi \in C \mid s \in \mathit{reach}(C, T)\}$ is called the *module* of C w.r.t. T.

Shape names in $\mathit{reach}(C, T)$ are said to be *reachable* in C from shape names in T. Example 20 illustrates the computation of the reachable set and module for the set of constraints given in Example 18.

Example 20. Consider a shapes graph (C, T) where C is the set of constraints from Example 18 and the target set is $T = \{s_1(a), s_1(b)\}$. Then the reachable shape names from the targets are $\mathit{reach}(C, T) = \{s_1, s_2, s_3, s_4\}$. Since there exists no *directed* path from s_1 to s_5 or s_6, these shapes are not in the reachable set. The reachable set results in the module

$$C_T = \{s_1 \leftarrow s_2 \wedge s_3 \qquad s_2 \leftarrow \exists r.s_4$$
$$s_3 \leftarrow s_2 \qquad s_4 \leftarrow \exists r.\top\}.$$

Recall that Figure 1.1 gives an overview of the input, output and procedures of the magic shapes algorithm. Now we present the algorithm, by providing a detailed breakdown of the functionality of the magic shapes algorithm in Algorithm 3.1 and an explanation of the four main steps.

Algorithm 3.1: Magic Shape Algorithm for positive SHACL

Input : shapes graph (C, T)
Output : Optimized shapes graph (C_{magic}, T).

```
1  begin
2      C_generate := {magic_s ← v | s(v) ∈ T} ;
3      C_T := {s ← φ ∈ C | s ∈ reach(C, T)};
4      foreach s ← φ ∈ C_T do
5          C_generated := C_generated ∪ { Generate(s ← φ_s)};
6      end
7      C_modified := {s ← magic_s ∧ φ | s ← φ ∈ C_T};
8      C_magic = C_generated ∪ C_modified;
9      return (C_magic, T);
10 end
```

Create magic seeds

In line 2 constraints of the form $\mathsf{magic\_}s \leftarrow v$, called *magic seeds*, are added for each shape atom $s(v)$ in T to the set of generated constraints $C_{\mathsf{generated}}$. This process creates magic shapes for shape atoms in the target set. By the magic seeds the validation is narrowed to the focus nodes.

Example 21. Continuing with Example 20, the algorithm adds two constraints to the set of generated constraints $C_{\mathsf{generated}}$. Therefore, after step *create magic seeds* the set $C_{\mathsf{generated}}$ contains the following:

$$\mathsf{magic\_}s_1 \leftarrow a \qquad\qquad \mathsf{magic\_}s_1 \leftarrow b$$

Modularize

The step modularize produces in line 3 the set of constraints relevant for validating the targets, i.e. the module C_T. For this purpose, the set $reach(C, T)$ is obtained as specified in Definition 19. Note that the reachable set (and module) depends on the shape *names* in the target set (e.g. s_1 for the (C, T) in Example 20) and not the targeted node(s) (e.g. a and b in Example 20), therefore the module from Example 20 would not change for a target set like $T' = \{s_1(a)\}$, since it still contains only s_1 as shape name.

Generation

For each *reachable* constraint of the targets the procedure `Generate`$(s \leftarrow \phi)$ creates *magic* constraints in lines 4-6. Algorithm 3.2 shows the sequence of procedure `Generate`, namely it generates for a constraint $s \leftarrow \phi$ new constraints of the form $\texttt{magic\_}s' \leftarrow \texttt{magic\_}s$ for each shape name s' in ϕ. In case the constraint connects two shapes by a path expression E, the path expression gets additionally inverted. The generated constraints are thereafter added to the set of generated constraints $C_{\mathsf{generated}}$.

Algorithm 3.2: Generation

```
1 function Generate(s ← φ) is
2     if φ =≥n E.s′ then
3         return magic_s′ ←≥n E⁻.magic_s;
4     else
5         foreach shape name s′ ≠ s in φ do
6             return magic_s′ ← magic_s;
7         end
8     end
9 end
```

Example 22. For the module C_T of Example 20 procedure `Generate` described in Algorithm 3.2 adds the following constraints to the set $C_{\mathsf{generated}}$:

$$\mathsf{magic\_}s_2 \leftarrow \mathsf{magic\_}s_1 \qquad \mathsf{magic\_}s_3 \leftarrow \mathsf{magic\_}s_1$$
$$\mathsf{magic\_}s_4 \leftarrow \exists r^-.\mathsf{magic\_}s_2 \qquad \mathsf{magic\_}s_2 \leftarrow \mathsf{magic\_}s_3.$$

Thus, after the lines 4 to 6 have been processed, the set $C_{\mathsf{generated}}$ consists of the *magic seeds* and the four constraints from above.

Modification

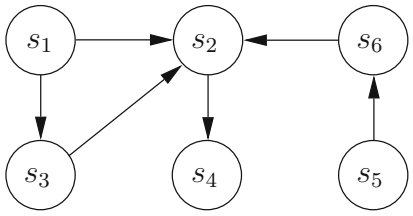
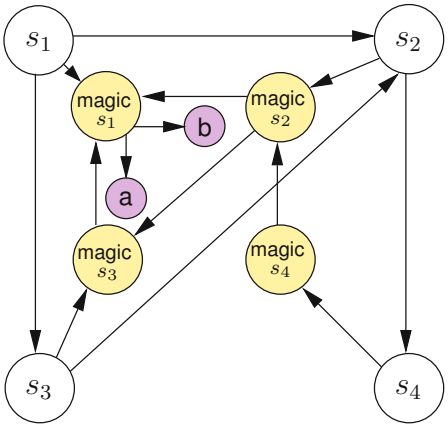
In this step (line 3) the body of each constraint in the module C_T is enhanced with a magic version of the shape name occurring in the head of the constraint, i.e. a constraint $s \leftarrow \phi$ is rewritten as $s \leftarrow \mathtt{magic\_}s \wedge \phi$ and added to the set of modified constrains C_{modified}. This ensures that the validation is only further executed for nodes that are recognizes as relevant, i.e. nodes that occur in the respective magic shape atom.

Example 23. Consider again C_T from Example 20, then the *Modification* step adds a constraint for each constraint in the module to the set C_{modified}:

$$s_1 \leftarrow \mathsf{magic\_}s_1 \wedge s_2 \wedge s_3 \qquad s_2 \leftarrow \mathsf{magic\_}s_2 \wedge \exists r.s_4$$
$$s_3 \leftarrow \mathsf{magic\_}s_3 \wedge s_2 \qquad s_4 \leftarrow \mathsf{magic\_}s_4 \wedge \exists r.\top$$

The output C_{magic} of the magic shapes algorithm consists of the union of the sets $C_{\mathsf{generated}}$ and C_{modified}. Example 24 gives an overview of the changes the magic shapes algorithm performs on the input.

Example 24. In the following a comparison between the input set of constraints C and the result C_{magic} from previous examples is given. A discussion of each row is provided after the table.

C	C_{magic}
	$\mathsf{magic\_}s_1 \leftarrow a$ $\mathsf{magic\_}s_1 \leftarrow b$ $\mathsf{magic\_}s_2 \leftarrow \mathsf{magic\_}s_1$ $\mathsf{magic\_}s_3 \leftarrow \mathsf{magic\_}s_1$ $\mathsf{magic\_}s_4 \leftarrow \exists r^-.\mathsf{magic\_}s_2$ $\mathsf{magic\_}s_2 \leftarrow \mathsf{magic\_}s_3$
$s_1 \leftarrow s_2 \wedge s_3$ $s_2 \leftarrow \exists r.s_4$ $s_3 \leftarrow s_2$ $s_4 \leftarrow \exists r.\top$ $s_5 \leftarrow s_6$ $s_6 \leftarrow \exists r.s_2$	$s_1 \leftarrow \mathsf{magic\_}s_1 \wedge s_2 \wedge s_3$ $s_2 \leftarrow \mathsf{magic\_}s_2 \wedge \exists r.s_4$ $s_3 \leftarrow \mathsf{magic\_}s_3 \wedge s_2$ $s_4 \leftarrow \mathsf{magic\_}s_4 \wedge \exists r.\top$
	
$G = \{ r(a,a), r(a,c) \}$ $A = \{ s_1(a), s_2(a), s_3(a), s_4(a), s_5(a), s_6(a), s_1(c), s_2(c), s_3(c), s_4(c) \}$	$G = \{ r(a,a), r(a,c) \}$ $A' = \{ s_1(a), s_2(a), s_3(a), s_4(a), s_5(a), s_6(a) \}$ $M = \{ \mathsf{magic\_}s_1(a), \mathsf{magic\_}s_1(b), \mathsf{magic\_}s_2(a), \mathsf{magic\_}s_2(b), \mathsf{magic\_}s_3(a), \mathsf{magic\_}s_3(b), \mathsf{magic\_}s_4(a) \}$

For a better overview, the set of shape constraints for the input shapes graph (C, T) and the magic output (C_{magic}, T) are summarized in the first row. As a reminder, the target set of the shapes graphs is $T = \{s_1(a), s_1(b)\}$. The second row contains the shape dependency graphs of the constraints. As can be seen the set C_{magic} no longer contains shape names s_5 and s_6, but four additional *magic shapes* have been added to the set. The removal of s_5 and s_6, the four additional magic shape constraints and the fact that

magic_s_1 refers to the target nodes narrow the validation of the data graph to what is relevant for validating the targets $s_1(a)$ and $s_1(b)$. In order to demonstrate the difference between the validation of C and C_{magic}, the third row shows a stable model $G \cup A$ for C and $G \cup M \cup A'$ for C_{magic}. The target set T is neither a subset of A nor of A', therefore the data graph G does not (bravely and cautiously) validate the shapes graphs (C, T) and (C_{magic}, T).

As this example shows, the returned set of magic shape constraints C_{magic} might contain disjunction, in the sense that several constraints with the same shape name in the head may occur in $C_{\mathsf{generated}}$. This can be rewritten as disjunction or with conjunction and negation if it is required in normal form.

3.2 Extended Algorithm for Full SHACL

To determine whether the validation of the targets is affected by inconsistencies, it is necessary to check whether the targets are associated with so-called *odd cycles*. An odd cycle is a cyclic dependency with an odd number of negated predicates, in the database-literature it is well-known that this causes inconsistencies [LZ04]. In the following, first the notion of *dangerous* shape names is introduced and then the magic shapes algorithm is extended to consider these *dangerous* shapes.

3.2.1 Dangerous Shapes

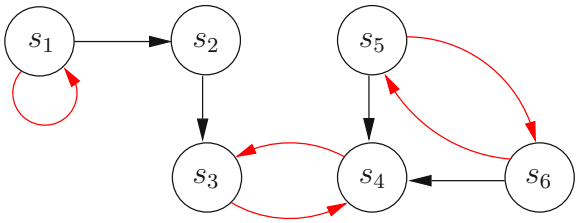
Dangerous shape names are shape names involved with odd cycles and can therefore lead to inconsistencies during validation of a data graph. But before defining dangerous shape names a definition for odd cycles and an example to illustrate them are provided below.

Definition 25 (Odd cycle)**.** An edge (s, s') of the shape dependency graph DG_C of C is *marked* if $s \leftarrow \neg s'$. An *odd cycle* in DG_C is a cycle, where an odd number of edges are marked.

Example 26. This example illustrates the shape dependency graph and its odd cycles for the set of constraints C with the following constraints:

$$s_1 \leftarrow \neg s_1 \wedge s_2 \qquad s_2 \leftarrow s_3$$
$$s_3 \leftarrow \exists r.\neg s_4 \qquad s_4 \leftarrow \exists r.\neg s_3$$
$$s_5 \leftarrow s_4 \wedge \neg s_6 \qquad s_6 \leftarrow s_4 \wedge \neg s_5$$

The corresponding shape dependency graph with the marked edges in red is shown below. It has three cycles, but only the cycle involving s_1 has an odd number of marked edges.

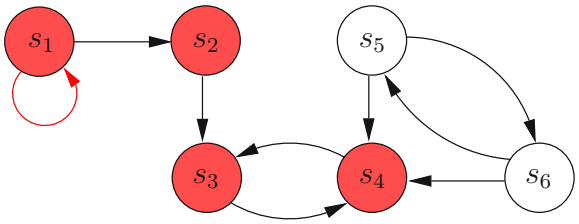


The following definition specifies when a shape name is identified as dangerous, which is the case if the shape name occurs directly in an odd cycle or is associated with an odd cycle in the dependency graph.

Definition 27 (Dangerous shape name)**.** Let DG_C be a shape dependency graph of a set of constraints C. A shape name s is *dangerous* if

(i) s occurs in an odd cycle of DG_C, or
(ii) there exists an edge (s', s) in DG_C where s' is dangerous

Example 28. Consider the set of shape constraints C from Example 26. This set has one odd cycle, which involves the shape name s_1. Since s_1 is dangerous and depends on s_2, s_2 depends on s_3 and s_3 on s_4, the shape names s_1 to s_4 become dangerous. The figure below shows the shape dependency graph with its dangerous shapes in red.



The following example demonstrates that the magic shapes algorithm for the full SHACL language must take dangerous shape names into account in order to preserve cautious-completeness and brave-soundness for consistent inputs. The completeness property means that if a data graph G validates a shapes graph (C, T), then G also validates (C_{magic}, T) and soundness means if G validates (C_{magic}, T), then G validates (C, T).

Example 29. Consider a data graph $G = \{r(a, a)\}$ and a document (C, T) where C is the set of constraints from Example 26 and the target set $T = \{s_4(a)\}$. Then there exists two supported models that also coincide with the stable models, which are

$$I_1 = \{r(a, a), s_4(a), s_5(a)\} \text{ and}$$
$$I_2 = \{r(a, a), s_4(a), s_6(a)\}.$$

Since the target set is a subset of both models, the data graph is both bravely and cautiously valid.
Now to compare this result of the validation with the output of the magic shapes algorithm

from Section 3.1, the set of magic constraints C_{magic} is computed. The reachable set contains the shape names s_3 and s_4. Consequently, the module of the shapes graph is the set $C_T = \{s_3 \leftarrow \exists r.\neg s_4, s_4 \leftarrow \exists r.\neg s_3\}$ and the resulting magic shapes graph C_{magic} consists of the constraints below:

$$\begin{aligned}
\mathsf{magic_}s_4 &\leftarrow a \\
\mathsf{magic_}s_4 &\leftarrow \exists r^{-}\mathsf{magic_}s_3 \\
\mathsf{magic_}s_3 &\leftarrow \exists r^{-}\mathsf{magic_}s_4 \\
s_3 &\leftarrow \mathsf{magic_}s_3 \wedge \exists r.\neg s_4 \\
s_4 &\leftarrow \mathsf{magic_}s_4 \wedge \exists r.\neg s_3
\end{aligned}$$

This transformed set of shape constraints likewise has two supported (and stable) models:

$$\begin{aligned}
I_1' &= \{r(a,a), \mathsf{magic_}s_3(a), \mathsf{magic_}s_4(a), s_4(a)\} \text{ and} \\
I_2' &= \{r(a,a), \mathsf{magic_}s_3(a), \mathsf{magic_}s_4(a), s_3(a)\}.
\end{aligned}$$

Different from the supported (or stable) models of G and C, not both models of G and C_{magic} contain the target $s_4(a)$ and thus G is also bravely valid against (C_{magic}, T), but not cautiously.
There are also cases were G bravely validates (C_{magic}, T), but not cautiously. For instance, consider again the graph G and the shapes graph (C, T'), with the target set $T' = \{s_3(a)\}$. Since the data graph and the set of constraints have not changed, the supported (or stable) models I_1 and I_2 remain the same. The target set T' is neither contained in I_1 nor in I_2, therefore G is neither bravely nor cautiously valid against the shapes graph (C, T'). When applying the magic shapes algorithm on the shapes graph (C, T'), the output C'_{magic} contains the same constraints as above, except for the *magic seed*, which is $\mathsf{magic\_}s_3 \leftarrow a$ due to the change of the target. Then the supported (or stable) models of $(C'_{\mathsf{magic}}, T')$ are:

$$\begin{aligned}
I_3' &= \{r(a,a), \mathsf{magic_}s_3(a), \mathsf{magic_}s_4(a), s_4(a)\} \text{ and} \\
I_4' &= \{r(a,a), \mathsf{magic_}s_3(a), \mathsf{magic_}s_4(a), s_3(a)\}.
\end{aligned}$$

Although the target set T' is not a subset of I_1 and I_2, it is contained in model I_4', which means that $(C'_{\mathsf{magic}}, T')$ bravely validates the graph G and (C, T') does not.

As Example 29 illustrates, the magic shapes algorithm from Section 3.1 is not cautious-complete and brave-sound for the full SHACL language. In order to maintain cautious-completeness and brave-soundness for the unrestricted language the magic shapes algorithm is extended to take dangerous shapes into account. For this purpose Definition 19 for reachable sets is extended by item (iii) below. The extended definition for reachable shape names takes into account not only the dependency from the head to the body, but also the reverse direction for dangerous shape names in the body.

Definition 30 (Extended Reachable Set, Module)**.** Given a shapes graph (C, T), $exReach(C, T)$ is the set of *extended reachable shape names* in C from shape names in T such that:

(i) if $s(a) \in T$, then $s \in exReach(C, T)$,

(ii) if $s(a) \in T$ and there exists an edge from s to s' in DG_C, then $s' \in exReach(C, T)$, and

(iii) if $s \in exReach(C, T)$, s' is dangerous and there exists an edge from s' to s in DG_C, then $s' \in exReach(C, T)$.

The set of constraints $C_T = \{s \leftarrow \phi \in C \mid s \in exReach(C, T)\}$ from C is called a *module* of C w.r.t. T.

By extending the reachable set, the module C_T is consequently enhanced by constraints with a dangerous shape name in the head. These are then processed by `Generate`, but this alone is not enough. Therefore the function `Generate` is expanded by a second part (see Algoritm 3.3 lines 9-16), which generates magic shapes also for the body to head direction of constraints with dangerous shape names. The following example demonstrates the extended version of the magic shapes algorithm.

Algorithm 3.3: Extended Generation

```
1  function Generate(s ← φ) is
2      if φ =≥n E.s' then
3          return magic_s' ←≥n E⁻.magic_s;
4      else
5          foreach shape name s ≠ s' in φ do
6              return magic_s' ← magic_s;
7          end
8      end
9      if s is dangerous then
10         if φ =≥n E.s' then
11             return magic_s ←≥n E.magic_s';
12         else
13             foreach shape name s ≠ s' in φ do
14                 return magic_s ← magic_s_i;
15             end
16         end
17 end
```

Example 31. Consider $G = \{r(a, a)\}$ and (C, T) from Example 29. The shapes graph has s_1 to s_4 as dangerous shape names and consequently the extended reachable set contains s_1 to s_4 (note that this does not necessarily have to be the same set). Accordingly the module is $C_T = \{s_1 \leftarrow \neg s_1 \wedge s_2, s_2 \leftarrow s_3, s_3 \leftarrow \exists r.\neg s_4, s_4 \leftarrow \exists r.\neg s_3\}$ based on which the magic shapes algorithm yields the constraints listed in column C_{magic} of the table

below. (The highlighted rows show the constraints added by the extended version of procedure `Generate` (see Algorithm 3.3)).

C			C_{magic}		
			magic_s_4	$\leftarrow$	a
			magic_s_2	$\leftarrow$	magic_s_1
			magic_s_1	$\leftarrow$	magic_s_2
			magic_s_3	$\leftarrow$	magic_s_2
			magic_s_2	$\leftarrow$	magic_s_3
			magic_s_4	$\leftarrow$	$\exists r^-.\mathsf{magic}\_s_3$
			magic_s_3	$\leftarrow$	$\exists r.\mathsf{magic}\_s_4$
			magic_s_3	$\leftarrow$	$\exists r^-.\mathsf{magic}\_s_4$
			magic_s_4	$\leftarrow$	$\exists r.\mathsf{magic}\_s_3$
s_1	$\leftarrow$	$\neg s_1 \wedge s_2$	s_1	$\leftarrow$	$\mathsf{magic}\_s_1 \wedge \neg s_1 \wedge s_2$
s_2	$\leftarrow$	s_3	s_2	$\leftarrow$	$\mathsf{magic}\_s_2 \wedge s_3$
s_3	$\leftarrow$	$\exists r.\neg s_4$	s_3	$\leftarrow$	$\mathsf{magic}\_s_3 \wedge \exists r.\neg s_4$
s_4	$\leftarrow$	$\exists r.\neg s_3$	s_4	$\leftarrow$	$\mathsf{magic}\_s_4 \wedge \exists r.\neg s_3$
s_5	$\leftarrow$	$s_4 \wedge \neg s_6$			
s_6	$\leftarrow$	$s_4 \wedge \neg s_5$			

The shapes graph (C_{magic}, T) has a unique stable model

$$I = \{r(a,a), \mathsf{magic}\_s_1, \mathsf{magic}\_s_2, \mathsf{magic}\_s_3, \mathsf{magic}\_s_4, s_4(a)\}.$$

Since the target T is contained in I the data graph both bravely and cautiously validates (C_{magic}, T) as it validates (C, T).

CHAPTER 4

Correctness

This chapter provides a correctness proof for the magic shapes algorithm. For a given shapes graph (C, T), the algorithm described in Chapter 3 outputs a new shapes graph (C_{magic}, T) where C_{magic} is the union of C_{generate} and C_{modified}. By C_{generate} the set of constraints of the form $\texttt{magic\_}s \leftarrow \phi$ and by C_{modified} the modified constraints from C are denoted. The correctness proofs guarantee that on consistent C validation of a data graph G against (C, T) and (C_{magic}, T) coincides for supported- and stable model semantics. Otherwise, the algorithm ensures cautious-completeness and brave-soundness of validation. In order to show these statements the following lemma is provided first:

Lemma 32. Let G be a data graph and let (C, T) be a shapes graph, where C is in the positive fragment. Then, there exists a unique assignment A, such that $G \cup A$ is an unique stable model of C.

Proof. By a simple inspection of the constraints in C can be observed that the claim holds. In particular, we observe that the constraints in C are positive, that is they do not contain negation, and hence a stable model must exist, which is unique. □

From Lemma 32 it follows that there exists an unique stable model for C_{generate}, since procedure `Generate` does not produce constraints of the form (NF4) and hence C_{generate} falls in the positive fragment. Note that there may be more *supported* models for a set C_{generate}, but there is a unique one that has a level assignment. The level assignment assures minimality of a supported model $I = G \cup M$ of C_{generate}, that means for any other supported model with assignment M' it holds that $M \subset M'$. In particular, the stable model of C_{generate} produces an overestimation of the necessary atoms to validate the target(s) against the set of constraints C for both the stable and supported model semantics. These atoms are marked with a 'magic' prefix and in this way preselected for the actual validation by C_{modified}.

Ground Constraints

In order to show the correctness of the magic shapes algorithm, we show that the magic shape constraint C_{magic} defines some kind of *module*, that is *independent* from the rest of the constraints in C. For this purpose we introduce the notion of *ground constraints*. Intuitively ground constraints instantiate shape constraints at a node v or a pair of nodes (v, v').

Definition 33 (Ground Constraints)**.** A *ground constraint* is a tuple $(s \leftarrow \phi, v)$, if $s \leftarrow \phi$ is a constraint of the form (NF1)-(NF5) and a tuple $(s \leftarrow \phi, (v, v'))$, if $s \leftarrow \phi$ is a constraint of the form (NF6), where v, v' are nodes from $\mathbf{N}$. We say $(s \leftarrow \phi, v)$ (or $(s \leftarrow \phi, (v, v'))$) is the *grounding* of $s \leftarrow \phi$ w.r.t. v (or (v, v')).

In the following we provide the definition for *grounding* a set of shape constraints with respect to a data graph.

Definition 34 (Grounding)**.** Given a data graph G and a set of constraints C, the *grounding* C_{gr} of C w.r.t. G is a set of ground constraints obtained for each $s \leftarrow \phi \in C$, such that

- if $s \leftarrow \phi$ is of the form (NF1)-(NF5), then $(s \leftarrow \phi, v)$ is in C_{gr} for each $v \in V(G)$,
- if $s \leftarrow \phi$ is of the form (NF6), then $(s \leftarrow \phi, (v, v'))$ is in C_{gr} for each $v, v' \in V(G)$.

Next the definition for supported- and stable models for ground constraints is given. This allows the validation of ground constraints.

Definition 35 (Supported-, Stable Model of Ground Constraints)**.** A decorated graph I is a supported model of a set of ground constraints C_{gr} if $[\![s]\!]^I = [\![\phi_s]\!]^I$ for each shape name s occurring in C_{gr} and for each $v \in [\![s]\!]^I$ there exists in C_{gr} a ground constraint $(s \leftarrow \phi, v)$ or $(s \leftarrow \phi, (v, v'))$. I is a stable model of C_{gr} if it is a supported model of C_{gr}, and there exists a level assignment such that $level(\phi_s, v) < level(s, v)$ for all $s(v) \in I$.

From this definition the following lemma immediately holds, since the grounding of a set of shape constraints C with respect to a data graph G instantiates every constraints in C with every node in $V(G)$.

Lemma 36. Let G be a data graph and let C be a set of constraints. Then I is a supported (or stable) model of C iff I is a supported (or stable) model of C_{gr}.

Independent Shape Constraints

In this section we define the notion of *independence*, i.e. a set of ground constraints is independent of another set of ground constraints. This definition is based on the notion for ground ASP programs in [EGM97] by Eiter et al. and splitting sets in [LT99] by Lifschitz et al.

Definition 37 (Shape Domain, Independent Constraints)**.** Given a set of ground constraints C_{gr}, the *shape domain* of C_{gr}, written as $\mathsf{SDom}(C_{gr})$, is the set of shape atoms $s(v)$ such that one of the following holds:

- $(s \leftarrow \phi, v)$ is in C_{gr}, or
- $(s' \leftarrow \phi, v)$ is in C_{gr} and s occurs in ϕ, or
- $(s \leftarrow \phi, (v, v'))$ is in C_{gr}, or
- $(s' \leftarrow \phi, (v', v))$ is in C_{gr} and s occurs in ϕ.

Let C_1 and C_2 be sets of ground constraints. Then C_1 is *independent* of C_2 if it is not the case that there is a shape name s of a ground constraint $(s \leftarrow \phi, v)$ or of the form $(s \leftarrow \phi, (v, v'))$ in C_2 such that $s(v)$ occurs in $\mathsf{SDom}(C_1)$.

This definition states that C_1 is independent of C_2 whenever the grounded shape name in the head of any constraint in C_2 does not occur at all in the ground constraints of C_1. This has the effect that the constraints in C_1 may have an impact on the constraints of C_2 but not the other way around.

Theorem 38. Let G be a data graph and let C_{gr} be the grounding of C w.r.t. G. Moreover, let C_1 and C_2 be such that $C_{gr} = C_1 \cup C_2$ and C_1 is independent of C_2. Let $M = \mathsf{SDom}(C_1)$. Then, for every supported (or stable) model $I = G \cup A$ of C_{gr} it is the case that $G \cup (A \cap M)$ is a supported (or stable) model of C_1.

Proof. First we show the claim for the supported model semantics and then for supported models with a level assignment. Let $I = G \cup A$ be a supported model of C_{gr} and let $I_1 = G \cup (A \cap M)$. We need to show that I_1 is a supported model of C_1, that is: 1) $[\![s]\!]^{I_1} = [\![\phi_s]\!]^{I_1}$ for each shape name s occurring in C_1, and 2) for each $v \in [\![s]\!]^{I_1}$ there exists a ground constraint $(s \leftarrow \phi, v)$ or $(s \leftarrow \phi, (v, v'))$ in C_1.

To show 1), first note that since C_1 is independent from C_2, it is the case for each node $v \in [\![s]\!]^{I_1}$ that all the ground constraints $(s \leftarrow \phi, v)$ or $(s \leftarrow \phi, (v, v'))$ that appear in C_{gr} appear also in C_1. Thus there is a unique ϕ_s that is the definition of s in both C_{gr} and C_1, which is in fact also its definition in C. Now, let s be an arbitrary shape name in C_1 and let v be an arbitrary node in $V(G)$. It is left to show that $v \in [\![s]\!]^{I_1}$ if and only if $v \in [\![\phi_s]\!]^{I_1}$. For $(\Rightarrow)$, let $s(v) \in I_1$. This implies that $s(v) \in A$ and $s(v) \in M$. The first implies that $v \in [\![\phi_s]\!]^{I}$. Let ϕ be an arbitrary disjunct in ϕ_s such that $v \in [\![\phi]\!]^{I}$. It is left to show that $v \in [\![\phi]\!]^{I_1}$. The claim trivially holds for ϕ as in (NF1), (NF2) or (NF3). If ϕ is of the form $\neg s'$, as in (NF4), then $s'(v)$ must not be in I and hence it is not in A, which implies that $v \in [\![\neg s']\!]^{I_1}$. If ϕ is a conjunction of shape names $s_1, \ldots, s_n$, as in (NF4), then each $s_i(v)$ is in A. Since $(s \leftarrow s_1 \wedge \cdots \wedge s_n, v)$ is in C_{gr}, then it must also be in C_1, which implies that each $s_i(v)$ is also in M, and hence also in I_1. Finally, for ϕ of the form $\geq_n E.s'$, as in (NF6), reasoning as in the previous case, there must be at least n

nodes v_i with an E-path from v such that $s'(v_i)$ in A. Since the grounding is with respect to every node in $V(G)$, for each such v_i there is a ground constraint $(s \leftarrow \geq_n E.s', (v, v_i))$ in C_{gr}, and since C_1 is independent of C_2, each such ground constraint is also in C_1. Hence, by the definition of the shape domain of C_1, $s'(v_i)$ is in M for each such node v_i. From these observations follows that $s'(v_i)$ is in $A \cap M$ for each v_i, which shows that v satisfies $\geq_n E.s'$ in I_1. For ($\Leftarrow$), $v \in [\![\phi_s]\!]^{I_1}$ implies $v \in [\![\phi_s]\!]^I$. Since I is a supported model of C_{gr}, it follows that $v \in [\![s]\!]^I$. That is, there is some ground constraint $(s \leftarrow \phi, v)$ or $(s \leftarrow \phi, (v, v'))$ that appears in C_{gr}, and hence, it also appears in C_1. It follows that $s(v)$ is also in M, and hence also in I_1.

To show 2), note that $s(v)$ is in M, so there is some constraint with s in the head or the body and v in the corresponding position, as in Definition 37. Note that it cannot occur in a body only, since there must be some constraint with s in the head and grounded with v in either C_1 or C_2, but as C_1 is independent from C_2, it cannot be in C_2.

For the stable model semantics, it is clear that if I has a level assignment, then $I_1 \subseteq I$ also has a level assignment. □

Independent Magic Shape Constraints

In the following we show that the supported or stable models of C_{magic} are also supported or stable models of the grounding of the module C_T with the nodes in the stable model of C_{generate}, i.e. the nodes with a "magic" prefix, and vice versa.

Lemma 39. Let G be a data graph, let (C, T) be a shapes graph and let $G \cup M$ be the stable model of C_{generate}. For each shape name s such that `magic_s` is in M let $N_s = \{v \mid \texttt{magic\_}s(v) \in M\}$. Let $C_{T,gr}$ be the set of ground constraints obtained by grounding every constraint $s \leftarrow \phi$ in C_T as follows:

- if $s \leftarrow \phi$ is of the form (NF1) to (NF5), then $(s \leftarrow \phi, v) \in C_{T,gr}$ for each $v \in N_s$,
- if $s \leftarrow \phi$ is of the form (NF6), then $(s \leftarrow \phi, (v, v')) \in C_{T,gr}$ for each v, v' with $v \in N_s$ and $v' \in N_{s'}$,

Then, the following hold under supported and stable model semantics:

(1) if I is a model of C_{magic}, then $I \setminus M$ is a model of $C_{T,gr}$, and

(2) if I is a model of $C_{T,gr}$, then $I \cup M$ is a model of C_{magic}.

Proof. For (1), let I be a supported model of C_{magic} and let s be an arbitrary shape name occurring in the constraints of C_{modified}. Then, it is the case that $[\![s]\!]^I = [\![\phi_s \wedge \texttt{magic\_}s]\!]^I$ and $[\![s]\!]^I \subseteq N_s$. Clearly, $s \leftarrow \phi_s$ is also the definition of s in C_T and by construction, for each $v \in N_s$ and for each constraint $s \leftarrow \phi$ in C_T, there is a ground constraint $(s \leftarrow \phi, v)$ or $(s \leftarrow \phi, (v, v'))$ in $C_{T,gr}$. By the definition of models of ground constraints, it follows

that $I \setminus M$ is also a model of $C_{T,gr}$. For the stable model semantics, the existence of a level assignment for $I \setminus M$ in item (1) is trivial.

For (2), let I be a model of $C_{T,gr}$. Then, $[\![s]\!]^I = [\![\phi_s]\!]^I$ and $[\![s]\!]^I \subseteq N_s$. Since N_s contains all the nodes appearing in atoms over $\texttt{magic\_}s$ in M, follows that $I \cup M$ is a model for C_{generate} and for C_{modified}. It is left to show that if $I = G \cup A$ has a level assignment, then $I \cup M$ has a level assignment. Since $G \cup M$ is the stable model of C_{generate}, there is a level assignment for the atoms with $\texttt{magic}$ as prefix such that $level(\phi_{\mathsf{magic}}, v) < level(\texttt{magic\_}s, v)$ for every $\texttt{magic\_}s(v) \in M$, with $\texttt{magic\_}s \leftarrow \phi_{\mathsf{magic}}$ a constraint in C_{generate}. Let L_G be an assignment, where the atoms from G are assigned integer 0, and let ℓ be the integer of the highest ranked shape atom in L_G. Furthermore, let L' be a level assignment for I. We define L_M which assigns to each shape atom α in A an integer $i + \ell$, where i is the level assigned to the atom α in L'. Clearly, the level assignment composed of L_G and L_M is as desired. □

The next lemma states that the ground constraints $C_{T,gr}$ obtained from C_{magic} are independent from the rest of the grounding C_{gr}.

Lemma 40. Let G be a data graph and let C_{gr} be the grounding of C w.r.t. G. Then, $C_{T,gr}$ is independent of $C_{gr} \setminus C_{T,gr}$.

Proof. Towards a contradiction, let $(s \leftarrow \phi, v)$ or $(s \leftarrow \phi, (v, v'))$ be in $C_{gr} \setminus C_{T,gr}$ and assume that $s(v)$ occurs in $\mathsf{SDom}(C_{T,gr})$. By Lemma 39 follows that $\texttt{magic\_}s(v)$ is in M, where $G \cup M$ is the stable model of C_{generate}. From that follows that the constraint $s \leftarrow \phi$ appears in C_T and by Lemma 39 there exists a ground constraint $(s \leftarrow \phi, v)$ or $(s \leftarrow \phi, (v, v'))$ in $C_{T,gr}$, thus deriving a contradiction to the initial assumption. □

Now we present the main lemma, that puts all the previous definitions and lemmas together to show that each supported (or stable) model of C can be modified, such that they become a supported (or stable) model of C_{magic}. And for consistent C it shows that the supported (or stable) model of C_{magic} can also be modified to obtain a supported (or stable) model for C.

Lemma 41. Let G be a data graph and let (C, T) a shapes graph. Assume $G \cup M$ is the stable model of C_{generate} and $M' = \{s(v) \mid \texttt{magic\_}s(v) \in M\}$. Then, the following hold under supported- and stable model semantics:

(1) if $I = G \cup A$ is a model of C, then $I' = G \cup M \cup A'$ is a model of C_{magic}, where $A' = A \cap M'$, and

(2) if C is consistent with G and $I' = G \cup M \cup A'$ is a model of C_{magic}, then there exists an A such that $I = G \cup A$ is a model of C and $A' = A \cap M'$.

Proof. For item (1), we again show the claim first for supported models and then for models with a level assignment. By Lemma 36 it follows that I is a model of the grounding of C w.r.t. G, i.e. C_{gr}. By Lemma 40 follows that $C_{T,gr}$ is independent of $C_{gr} \setminus C_{T,gr}$, where C_T is the module of C w.r.t. T, and $C_{T,gr}$ is the grounding of C_T w.r.t. the stable model $G \cup M$ of C_{generate} as defined in Lemma 39. This observation and Theorem 38, imply that $G \cup (A \cap M')$ is a supported model of $C_{T,gr}$, which together with Lemma 39, item (2), imply that $I' = G \cup (A \cap M') \cup M$ is a supported model of C_{magic}. For the stable model semantics, it is left to show that I' has a level assignment if I has a level assignment. For that we argue analogously to the proof of Lemma 39 item (2), that is we can combine the level assignments for $G \cup M$ and I into a level assignment for I'.

For item (2), similarly as above, we first show the claim for supported models and argue about the existence of a level assignment. Let $I' = G \cup M \cup A'$ be a supported model of C_{magic}. By Lemma 39, item (1), follows that $I' \setminus M$, i.e. $G \cup A'$ is a supported model of $C_{T,gr}$. Since $C_{T,gr}$ is independent from $C_{gr} \setminus C_{T,gr}$, we know that for every supported model $I = G \cup A$ of C_{gr} it is the case that $G \cup (A \cap M')$ is a supported model of $C_{T,gr}$, from which follows that $A' = A \cap M'$. It is left to show that such an A exists, which we argue similarly as in the proof of Theorem 3.4 in [FGL07]: in a nutshell, we show that there is no constraint participating in an odd cycle in $C_{gr} \setminus C_{T,gr}$ that 'kills' the assignment A' of $C_{T,gr}$.

To this aim, let $C_{odd,gr}$ denote the set of all ground constraints $(s \leftarrow \phi, v)$ or $(s \leftarrow \phi, (v, v'))$ in $C_{gr} \setminus C_{T,gr}$, where s is a dangerous shape name. This means that $C_{odd,gr}$ contains all ground constraints from $C_{gr} \setminus C_{T,gr}$ with a dangerous shape name s in the head that participates either in an odd cycle, is reachable from an odd cycle, or reaches an odd cycle (see Definition 27 for dangerous shape names). It follows that the shape domain $\mathsf{SDom}(C_{odd,gr}) \cap \mathsf{SDom}((C_{gr} \setminus C_{T,gr}) \setminus C_{odd,gr}) = \emptyset$. Moreover, we show that $\mathsf{SDom}(C_{odd,gr}) \cap \mathsf{SDom}(C_{T,gr}) = \emptyset$. By Definition 37, there is no ground constraint $(s \leftarrow \phi, v)$ or $(s \leftarrow \phi, (v, v'))$ in $C_{odd,gr}$ with $s(v) \in \mathsf{SDom}(C_{T,gr})$ since $C_{T,gr}$ is independent of $C_{gr} \setminus C_{T,gr}$ and $C_{odd,gr}$ is contained in $C_{gr} \setminus C_{T,gr}$. Assume towards a contradiction that there is a ground constraint $(s \leftarrow \phi, v)$ or $(s \leftarrow \phi, (v, v'))$ in $C_{odd,gr}$ with s' in ϕ of the constraint $s \leftarrow \phi$ such that $s'(v) \in \mathsf{SDom}(C_{T,gr})$. First, we observe that since s of such a constraint $(s \leftarrow \phi, v)$ or $(s \leftarrow \phi, (v, v'))$ in C_{gr} is dangerous, then $s \leftarrow \phi$ must also also be dangerous in C, and by definition, the shape name s' occurring in ϕ is also dangerous. Since $s'(v) \in \mathsf{SDom}(C_{T,gr})$ by Definition 37, there is a constraint such that at least one of the following holds:

- $(s'' \leftarrow \phi'', v)$ is in $C_{T,gr}$ and $s' = s''$, or
- $(s'' \leftarrow \phi'', v)$ is in $C_{T,gr}$ and s' occurs in ϕ'', or
- $(s'' \leftarrow \phi'', (v, v'))$ is in $C_{T,gr}$ and $s' = s''$, or
- $(s'' \leftarrow \phi'', (v', v))$ is in $C_{T,gr}$ and s'' occurs in ϕ''.

Thus, in $s'' \leftarrow \phi''$ either s'' is dangerous or it contains a dangerous shape name in ϕ''. Hence, by Definition 30, $s \leftarrow \phi$ is present in the module C_T and by Definition 33, $C_{T,gr}$ contains the ground constraint $(s \leftarrow \phi, v)$ or $(s \leftarrow \phi, (v', v))$ and is therefore not in $C_{odd,gr}$, deriving a contradiction.

From the above observations it follows that the shape domain of $C_{odd,gr}$ has no intersection with the shape domain of the rest of the constraints in C_{gr}. Since C_{gr} is consistent, it follows that there exists a supported model of $C_{odd,gr}$. Let $G \cup A_{odd}$ be that supported model. Note that the rest of the set of ground constraints, that is $(C_{gr} \setminus C_{T,gr}) \setminus C_{odd,gr}$ does not contain any odd cycles. It is known from answer set programming that programs that do not contain any odd cycles are consistent [Dun92], which means they admit a supported model. This result can be lifted to SHACL by results from Andresel et al. in [ACO+20], which showed that a shapes graph can be translated into a logic program with negation that preserves validation under the stable and supported model semantics. These transformation does not change the parity of negation in cyclic dependencies of shape names. Hence, a shape name s appears in a cycle with an odd number of negative edges if and only if the corresponding unary atom over s appears in a cycle with an odd number of negative edges in the translated logic program. So, since $(C_{gr} \setminus C_{T,gr}) \setminus C_{odd}$ has no odd cycles, we know it has a supported model. Let $G \cup A_{rest}$ be that supported model. It then follows that $G \cup A_{rest} \cup A' \cup A_{odd}$ is a supported model of C_{gr}. By Lemma 36 it holds that $I = G \cup A$ where $A = A' \cup A_{rest} \cup A_{odd}$ is a supported model of C. Neither A_{odd} nor A_{rest} have any atoms from M', so $A' = A \cap M'$. This completes the proof for the supported model semantics.

For the stable model semantics, it remains to argue that there is a level assignment for I. We know that there are level assignments for each of $G \cup A_{rest}$, $G \cup A_{odd}$ and $G \cup A'$. Since A_{rest} and A_{odd} are disjoint, and they are both disjoint from A', we can combine them into a level assignment for I. □

Intuitively, the above theorem states that every stable model of C can be converted into a stable model of C_{magic} by restricting the set of shape atoms to those defined by the magic part of C_{magic}. And every stable model of C_{magic} can be extended into a stable model of C, after removing the magic shape atoms. Moreover, Theorem 41 implies the following corollary.

Corollary 42. Let G be a data graph and let (C, T) be a shapes graph. Then, C is consistent with G implies that C_{magic} is consistent with G.

Finally, the relationship between a set of shape constraints C and the produced set of magic shape constraints C_{magic} for a given target set T is establish.

Theorem 43. Let G be a data graph and let (C, T) be a shapes graph. Then, the following hold:

(1) if G *bravely* validates (C, T), then G *bravely* validates (C_{magic}, T),

(2) if G *cautiously* validates (C_{magic}, T), then G *cautiously* validates (C, T), and

(3) if C is consistent with G, then G validates (C_{magic}, T) iff G validates (C, T), under both *brave* and *cautious* validation.

Proof. For (1), suppose $I = G \cup A$ is a stable model of C and $T \subseteq A$. Observe that for every such atom there exists a constraint of the form $\texttt{magic\_}s \leftarrow v$ in C_{magic}, and particularly in C_{generate}. Hence, $\texttt{magic\_}s(v)$ is in the stable model $G \cup M$ of C_{generate} for each $s(v) \in T$ and by definition it follows that $s(v) \in M'$. From the above observations and by Lemma 41 item (1) it follows that $I' = G \cup M \cup A'$, where $A' = A \cap M'$, is a stable model of C_{magic} and since $T \subseteq M$ and $T \subseteq A$ it follows that $T \subseteq A'$, therefore it holds that G bravely validates (C_{magic}, T).

For (2), the claim trivially holds for the case where C is not consistent with G. Otherwise, suppose G cautiously validates C_{magic}. Then, either (a) C_{magic} is not consistent with G, which by Corollary 42 implies that C is not consistent with G, and thus the claim trivially follows, or (b) C_{magic} is consistent with G and I' is a stable model of C_{magic} and $T \subseteq A$ for each I'. For the latter, let $I' = G \cup M \cup A'$ be an arbitrary stable model of C_{magic} such that $T \subseteq A'$. By Lemma 41 item (2) it is the case for consistent C that there is a stable model $I = G \cup A$ where $A' = A \cap M'$, $G \cup M$ is the stable model of C_{generate} and $M' = \{s(v) \mid \texttt{magic\_}s(v) \in M\}$. Since $A' \subseteq A$ it follows that $T \subseteq A$.

For (3), C is consistent with G implies C_{magic} is consistent with G. It suffices to show (4) if G *bravely* validates (C_{magic}, T), then G bravely validates (C, T), and (5) if G *cautiously* validates (C, T), then G cautiously validates (C_{magic}, T). Then, (4), (5) together with (1) and (2) show the correctness of the claim. For (4), suppose $I' = G \cup M \cup A$ is a stable model of C_{magic}. Then, arguing as above by Theorem 41, item (2), let a shape assignment A be such that $I = G \cup A$ is a stable model of C, where $A' = A \cap M'$, $G \cup M$ is the stable model of C_{generate} and $M' = \{s(v) \mid \texttt{magic\_}s(v) \in M\}$. By assumption $T \subseteq A'$, and since $A' \subseteq A$, it follows that I contains T and hence I is a stable model of C, which shows the claim. For (5), towards a contradiction assume $I' = G \cup M \cup A'$ is a stable model of C_{magic} such that $s(v) \notin A'$, for some shape atom $s(v) \in T$. Then, let a shape assignment A be such that $A \cap M = \emptyset$ and $I = G \cup A$ is a stable model of C, where $A' = A \cap M'$, $G \cup M$ is the stable model of C_{generate} and $M' = \{s(v) \mid \texttt{magic\_}s(v) \in M\}$. Such a shape assignment must exist by Theorem 41 item (2). By assumption $s(v) \notin A'$, and $s(v)$ cannot be in A either since $s(v) \in M'$ because $\texttt{magic\_}s(v) \in M$. It follows that I is not a stable model of C, which contradicts the assumption that G cautiously validates (C, T).

The proof for supported model semantics is identical, since the essential Lemma 41 also holds for the supported model semantics. □

For the positive fragment of SHACL, brave and cautious validation coincide as Lemma 32 shows that there exists exactly one stable model for C in the positive fragment.

CHAPTER 5

Inconsistency-tolerant Validation

The previous chapters indicate that there are cases were a data graph might not be consistent with a shapes graph (C, T), but it may nevertheless be consistent with the output shapes graph (C_{magic}, T) of the magic shapes algorithm. This suggests that the parts of the data- and shapes graph causing the inconsistency are not relevant for validating the targets. As the size of RDF data graphs increases, the likelihood of such situations occurring also increases. The 2-valued semantics considered in this thesis require a total assignment i.e. a node of the data graph is neither assigned to a shape or its negation. Until here a shape atom $s(v)$ is considered as negated if it does not occur in the assignment. In [CRS18] Corman et al. separate the definition for *assignment* and *total assignment*. Further, they define validation based on the notion of a *faithful assignment* or *strictly-faithful assignment*. The evaluation of 3-valued assignments in [ACO$^+$20] corresponds to the notion of a strictly-faithful assignment, whereas the inconsistency-tolerant validation in this chapter is based on the notion of faithful assignment. Hence, 3-valued assignments have been proposed, but no good validation algorithm exists. The magic shapes algorithm can be deployed to decide for a 3-valued model. To this aim we first introduce a new definition for assignment, which makes negated atoms explicit to allow shape atoms to be *undefined*.

Definition 44 (Literal assignment)**.** A *literal assignment* A for a graph G is a set of shape atoms of the form $s(v)$ or $\neg s(v)$ where $s \in \boldsymbol{S}$ and $v \in V(G)$.

A shape atom $s(v) \in A$ is called *undefined* if for a shape name s occurring in C and a node $v \in V(G)$ it is the case that $\{s(v), \neg s(v)\} \cap A = \emptyset$.

In the following 3-valued models for the supported and the stable model semantics are defined based on the literal assignment.

Definition 45 (3-valued supported model)**.** Assume a set of shape constraints C and a literal assignment A for G. Then $I = G \cup A$ is called a *3-valued supported model* of C if for each $s \leftarrow \phi_s \in C$ and for each node $v \in V(G)$

(i) if $s(v) \in I$, then $v \in [\![\phi]\!]^I$, and

(ii) if $\neg s(v) \in I$, then $v \notin [\![\phi]\!]^I$.

Definition 46 (3-valued stable model)**.** Assume a set of shape constraints C and a literal assignment A for G. Then $I = G \cup A$ is called a *3-valued stable model* of a set of constraints C if

(i) I is a 3-valued supported model of C

(ii) there exists a level assignment for I such that for all $s(v) \in I, level(\phi, v) < level(s, v)$, with $s \leftarrow \phi$ the constraints in C.

Definition 47 (Validation)**.** A graph G is *valid* against a document (C, T) under *3-valued supported* (or *3-valued stable model*) semantics if there exists a literal assignment A such that

(i) $G \cup A$ is a 3-valued supported resp. stable model of C, and

(ii) $T \subseteq A$

Note that every 2-valued supported- (or stable) model $G \cup A$ of a set of constraint C can be translated into a 3-valued supported (resp. 3-valued stable) model $G \cup A'$. In order to do so let A' extend A by the negated version of each shape atom $s(v) \notin A$ with s a shape name appearing in C and v a node in $V(G)$.

The magic shapes algorithm provides an efficient method to validate a shapes graph under the 3-valued semantics. The models of the magic shapes graph can be seen as 3-valued models, where shape atoms relevant for validating the targets are identified by C_{generate}. The shape atoms of the model for C_{generate} are assigned either true or false and all remaining atoms are left undefined, since they are irrelevant for the validation of the targets.

Lemma 48. Let G be a data graph, let C be a shapes graph, and let $I' = G \cup M \cup A'$ be a supported (or stable) model of C_{magic}, where $G \cup M$ is the stable model of C_{generate}. Let $M' = \{s(v) \mid \texttt{magic\_}s(v) \in M\}$. Then, $I = G \cup A$ is a 3-valued supported (or stable) model of C, where

$$A = A' \cup \{\neg s(v) \mid s(v) \in M' \setminus A'\}$$

Proof. In order to show that $I = G \cup A$ is a 3-valued supported model we need to show that (i) if $s(v) \in I$, then $v \in [\![\phi]\!]^I$, and (ii) if $\neg s(v) \in I$, then $v \notin [\![\phi]\!]^I$. Let $I' = G \cup M \cup A'$

be a supported model of C_{magic}, it follows that $[\![s]\!]^{I'} = [\![\texttt{magic\_}s]\!]^{I'} \cap [\![\phi]\!]^{I'}$ for each constraint $s \leftarrow \texttt{magic\_}s \wedge \phi \in C_{\mathsf{magic}}$. Note that the evaluation of shape expressions is not affected by additional negated shape atoms (see Table 2.1), hence the equality $[\![\phi]\!]^{I} = [\![\phi]\!]^{I'}$ holds. For positive shape atoms $s(v) \in I$, we know that $s(v) \in A'$, which implies that $v \in [\![\phi]\!]^{I}$. For the negated shape atoms of the form $\neg s(v) \in A$, we know that $s(v)$ is in M' but not in A', which implies that $v \notin [\![\phi]\!]^{I}$.

To prove it for the 3-valued stable model semantics it is left to show that I has a level assignment such that for all $s(v) \in I$ it holds that $level(\phi_s, a) < level(s, a)$. Let I' be a stable model of C_{magic}, then it has a level assignment where $level(\texttt{magic\_}s \wedge \phi_s) = max(\{level(\texttt{magic\_}s), level(\phi_s)\}) < level(s, v)$ for all $s(v) \in I'$. From this observation follows that $level(\phi_s) < level(s, v)$ for all $s(v) \in I'$. By construction of I every $s(v)$ in I is also in I', therefore for all $s(v) \in I$ it holds that $level(\phi_s) < level(s, v)$. □

Theorem 49. Let G be a data graph, let (C, T) be a shapes graph. Then, if G validates (C_{magic}, T) under the supported (or stable) model semantics, then G validates (C, T) under the 3-valued supported (or stable) model semantics.

Proof. Let $I = G \cup M \cup A'$ be a stable model of C_{magic} and $T \subseteq A$. Then G is valid against a shapes graph (C, T) under the 3-valued supported (or stable) model semantics if (i) I is a 3-valued supported- or stable model and (ii) $T \subseteq A$. By Theorem 48 (i) holds, since $G \cup A$ is a 3-valued supported model (or stable model) of C, where $A = A' \cup \{\neg s(v) \mid s(v) \in M' \setminus A'\}$ and $M' = \{s(v) \mid \texttt{magic\_}s(v) \in M\}$, and (ii) holds by assumption. □

The converse does not hold as can be seen by the following counter example. This is the case because the magic shapes technique provides a stronger inconsistency-tolerant validation than arbitrary 3-valued models.

Example 50. Consider a shapes graph (C, T) and a data graph G defined as follows:

$$C = \{s \leftarrow s' \wedge \neg s, s' \leftarrow \exists r.\top\}$$
$$T = \{s'(a)\}$$
$$G = \{r(a, b)\}$$

Then $G \cup \{s'(a)\}$ is a 3-valued supported (or stable) model of C that validates the target T. But it is not a 2-valued supported (or stable) model of C_{magic}, because shape name s is involved in an odd cycle and is therefore dangerous. Since s' occurs in the body of $s \leftarrow s' \wedge \neg s$, the constraint is also contained in C_T by Definition 30. Based on this the magic output contains the constraint $s \leftarrow \mathsf{magic\_}s \wedge s' \wedge \neg s$. For the data and shapes graph there is no consistent way to assign or not assign node a to shape name s, hence there exists no supported (or stable) model for C_{magic}.

CHAPTER 6

Implementation & Experiments

To conduct experiments and test the performance and advantages of the approach, we implemented a prototype of the magic shapes algorithm. The prototype receives as input a shapes graph and produces as output a new magic shapes graph, more details on the implementation are provided in the first section of this chapter. In a second section the evaluation of the experiments are discussed. For the experiments the SHACL-ASP validator performed once on the original shapes graph and once on the magic variant. The source code of the prototype and the files used for the experiments are available at the following link: `https://github.com/biziloehnert/magicSHACL`.

6.1 Prototype

Our prototype takes as input a shapes graph either in abstract or Turtle syntax and produces a magic shapes graph, again in abstract or Turtle syntax. Later in this section we give more details about the different conversion of syntaxes. For the implementation of the different parsers and the generation of the magic shapes graph, we applied the *model-driven engineering* (MDE) approach [Ken02]. The model of this approach is not to be confused with the previously defined supported- or stable models. The MDE approach allows software development based on a domain model also called *meta-model*. In case of the MAGICSHAPES prototype the meta-model is an abstract representation of the SHACL language. In order to apply the MDE approach the *eclipse model framework* (EMF) was used[1]. Figure 6.1 shows the complete meta-model as UML diagram and Figure 6.2 provides an excerpt meta-model with the components of the SHACL language. The prototype is able to parse and generate different syntax, i.e. Turtle and an abstract shapes graph representation. For parsing the project Xtext[2] was deployed and for generating

[1] `https://www.eclipse.org/modeling/emf/`
[2] `https://www.eclipse.org/Xtext/`

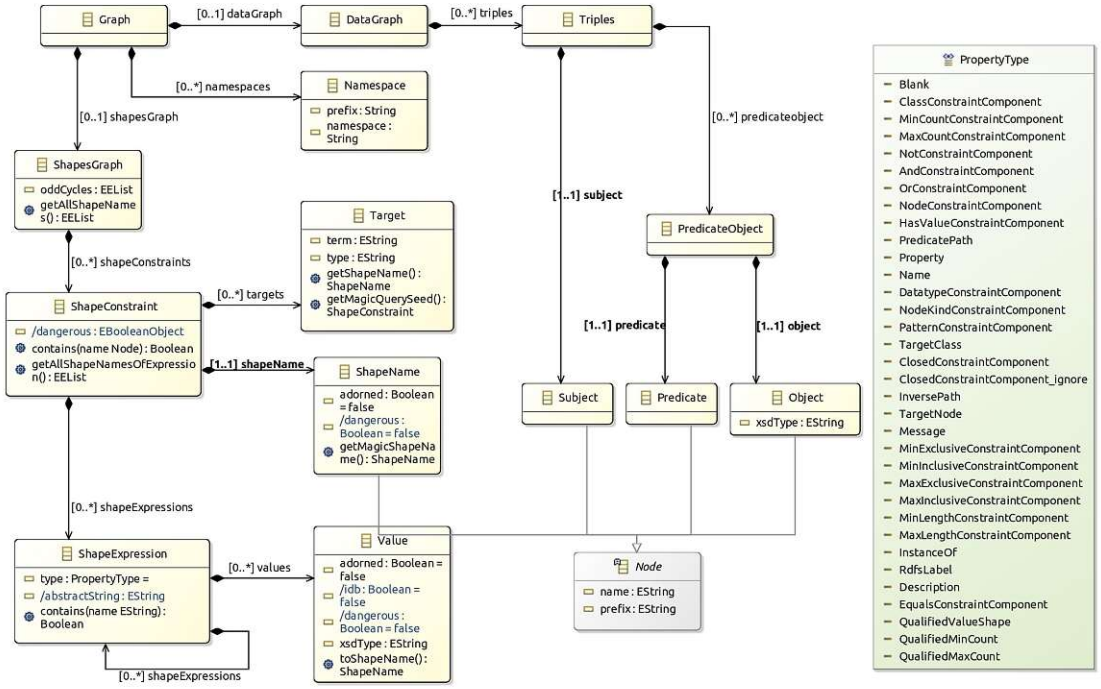


Figure 6.1: Meta-model as UML diagram used to implement the prototype by means of MDE

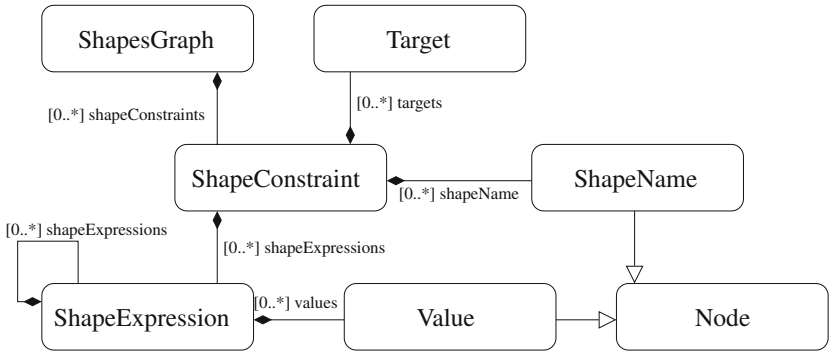


Figure 6.2: Excerpt of the meta-model used to implement the prototype

Xtend[3], both are part of the *eclipse modeling project*[4] (like EMF) and based on the meta-model shown in 6.2. To enable the parsing of shapes graphs in Turtle or abstract syntax, Xtext is configured with the grammar of each language, which is similar to the *Extended Backus-Naur Form* (EBNF). A transition from EBNF to Xtext is provided

[3] `https://www.eclipse.org/xtend/`
[4] `https://projects.eclipse.org/projects/modeling`

```
ShapesGraph = {ShapeConstraint}
ShapeConstraint = ShapeName (Target "?" | ":-" ShapeExpression ";")
ShapeName = UNICODE
Target = "(" UNICODE ")"
ShapeExpression = PropertyType {Value} | Value {PropertyType Value}
PropertyType = "MIN"|"MAX"|"AND"|"OR"|"NOT"|"SOME"|"SOME ^"
Value = UNICODE
UNICODE = [^]{"a".."z"|"A".."Z"|"0".."9"|"_"|":"|"#"|"/"|"."|"-"|"”"|"{"|"}"|"\\"|"$"}
```

Table 6.1: EBNF of the abstract syntax for SHACL

in [Yue14]. Table 6.1 shows the grammar of the abstract syntax for SHACL in EBNF. The grammar is based on the elements of the meta-model in Figure 6.2. Besides the uncomplicated implementation of a parser, another advantage of MDE is the generation of files (or code), based on the meta-model. By means of this mechanism the following transformations are performed:

- if the input shapes graph (C, T) is in Turtle syntax, then the prototype produces
  $\rightarrow$ a normalized shapes graph (C', T) in abstract syntax, and
  $\rightarrow$ a magic shapes graph (C_{magic}, T) in Turtle syntax
- if the input shapes graph (C, T) is in abstract syntax, then it produces
  $\rightarrow$ a magic shapes graph (C_{magic}, T) in abstract syntax

The transformation from a shapes graph in abstract syntax to one in Turtle syntax has not yet been implemented, since the SHACL-ASP validator only accepts abstract syntax and thus merely shapes graphs in abstract syntax were needed for the experiments. The reverse direction was implemented, as for the experiments shapes graphs from [CFRS19b] were expanded and these are available in Turtle syntax. Nevertheless, the framework provides the mechanisms and thus allows the list of transformations to be extended if necessary.

6.2 Evaluation

The experiments were performed on a Linux server with a 24 core Intel Xeon CPU running at 2.20 GHz and 264 GB of RAM. We used the DLV based SHACL-ASP validator and an Apache Jena TDB[5] as an RDF triple store to retrieve the relevant triples from the data graph, which are transformed into ASP facts.

Data Graph

The data graph G for the experiments was obtained from DBPedia (latest-core, version 2021-12)[6]. More precisely, the datasets "PersonData", "Instance Types", "Labels",

[5] https://jena.apache.org/documentation/tdb/
[6] https://www.dbpedia.org/resources/latest-core/

$$
\begin{aligned}
C_1 = \{\quad & \mathsf{Actor} \leftarrow \mathsf{Person} \wedge (\exists \mathit{starring}^{-}.\mathsf{Movie} \vee \exists \mathit{occupation}.\mathit{actor}),\\
& \mathsf{Director} \leftarrow \mathsf{Person} \wedge \exists \mathit{director}^{-}.\mathsf{Movie},\\
& \mathsf{Movie} \leftarrow \geq \mathit{imdbId}.\top \wedge \leq \mathit{imdbId}.\top \wedge \exists \mathit{starring}^{-}.\mathsf{Actor},\\
& \mathsf{Location} \leftarrow \exists \mathit{country}.\top,\\
& \mathsf{TranslatedMovie} \leftarrow \mathsf{Movie} \wedge \geq_2 \mathit{language}.\top,\\
& \mathsf{Anarchy} \leftarrow \mathsf{Location} \wedge \neg \exists \mathit{leaderTitle}.\top,\\
& \mathsf{AwardWinning} \leftarrow \exists \mathit{award}.\top,\\
& \mathsf{Famous} \leftarrow \exists \mathit{knownFor}.\top,\\
& \mathsf{LivingLanguage} \leftarrow \exists \mathit{spokenIn}.\mathsf{Location},\\
& \mathsf{Person} \leftarrow \geq \mathit{birthPlace}.\mathsf{Location} \wedge \leq \mathit{birthPlace}.\mathsf{Location},\\
& \mathsf{Parent} \leftarrow \mathsf{Person} \wedge \exists \mathit{child}.\top,\\
& \mathsf{Musician} \leftarrow \mathsf{Person} \wedge \exists \mathit{instrument}.\top,\\
& \mathsf{Employee} \leftarrow \exists \mathit{employer}.\top,\\
& \mathsf{WorkingPerson} \leftarrow \mathsf{Person} \wedge \mathsf{Employee},\\
& \mathsf{WorkingParent} \leftarrow \mathsf{Parent} \wedge \mathsf{WorkingPerson},\\
& \mathsf{WorkingClass} \leftarrow \mathsf{Person} \wedge \exists \mathit{child}^{-}.\mathsf{WorkingParent}\}\\
C_2 = \quad & (C_1 \setminus \{\mathsf{Employee} \leftarrow \exists \mathit{employer}.\top\}) \cup \{\mathsf{Employee} \leftarrow \exists \mathit{employer}.\mathsf{Employee}\}
\end{aligned}
$$

Table 6.2: Sets of constraints C_1 and C_2 used in the experiments.

"Mappingbased Literals" and "Mappingbased Objects" were used. The data sets are in Turtle syntax and contain about 119 million triples (15.4 GB). These data sets are stored as triples in an Apache Jena TDB, from where the data can be accessed by means of SPARQL queries. The SHACL-ASP validator retrieves triples from the data graph G and transforms them into corresponding ASP facts. This ASP facts form a sub-graph of G, which we denote as G_{asp}. The data graph G_{asp} for the experiments of this thesis contains about 7 million facts and 4 million constants i.e. nodes in the graph. To be able to evaluate the inconsistency-tolerant semantics a few additional triples were added to the data graph, i.e. a cycle of employers $\mathit{employer}(\mathit{bill}, \mathit{bob})$, $\mathit{employer}(\mathit{bob}, \mathit{jim})$ and $\mathit{employer}(\mathit{jim}, \mathit{bill})$. All three nodes *bill*, *bob* and *jim* are of type (class) *Person* and have a *birthPlace*.

Shapes Graphs

For the experiments two set of shape constraints were created from the domain of persons and their professions motivated by those in [CFRS19b]. Table 6.2 shows the sets of constraints C_1 and C_2 in abstract syntax, both containing 16 shape constraints. The set C_2 contains a cycle (recursion) in the shape Employee, this can cause inconsistencies if the data graph has cycles involving the relation *employer*. The sets of constraints were combined with different targets in order to obtain 40 different shapes graphs. The targets are either node targets $s(v)$ or *class targets* $s(C)$, that allow to specify focus nodes via class names. Such a target definition is a short-cut for node targets $s(v)$, where v has *type* C in the data graph G, i.e. there exists a triple of the form v `rdf:type` C. The

Targets		C_1			C_2		
T	nodes	t	t_{magic}	$\lvert C_{\mathsf{magic}}\rvert$	t	t_{magic}	$\lvert C_{\mathsf{magic}}\rvert$
Musician(*mozart*)	1	783s	243s	6	860s	231s	6
Musician(*Actor*)	4872	958s	253s	6	957s	240s	6
Musician(*Person*)	2267445	894s	410s	6	936s	407s	6
Actor(*cameron*)	1	768s	384s	8	894s	404s	8
Actor(*Actor*)	4872	905s	450s	8	929s	441s	8
Actor(*Person*)	2267445	975s	620s	8	933s	642s	8
Movie(*Film*)	143121	882s	474s	8	999s	448s	8
TranslatedMovie(*Film*)	143121	849s	491s	12	905s	508s	12
Employee(*bill*)	1	868s	42s	2	936s	45s	4
Employee(*mark*)	1	785s	39s	2	831s	43s	4
Employee(*Actor*)	4872	825s	42s	2	977s	44s	4
Employee(*Person*)	2267445	868s	86s	2	880s	145s	4
WorkingPerson(*bill*)	1	846s	241s	8	981s	259s	10
WorkingPerson(*mark*)	1	795s	234s	8	879s	249s	10
WorkingPerson(*Actor*)	4872	864s	250s	8	997s	294s	10
WorkingPerson(*Person*)	2267445	906s	410s	8	970s	488s	10
WorkingClass(*bill*)	1	734s	258s	14	890s	283s	16
WorkingClass(*mark*)	1	822s	291s	14	942s	285s	16
WorkingClass(*Actor*)	4872	906s	278s	14	979s	313s	16
WorkingClass(*Person*)	2267445	887s	445s	14	985s	588s	16

Table 6.3: Results of experiments with the SHACL-ASP validator.

number of focus nodes ranges from 1 to over 2 million nodes, see the second column in Table 6.4 and 6.3.

Results

Table 6.3 summarizes the main results of the conducted experiments. Column t and t_{magic} shows the average execution time for running the validation five times on the input shapes graph and five times on its magic version. The results display that the magic shapes algorithm significantly improves the performance of the SHACL-ASP validator in the test cases by at least 30% and in some cases by as much as 95% (e.g. where shape name Employee is targeted). The columns $\lvert C_{\mathsf{magic}}\rvert$ give the number of shape constraints in the magic versions of C_1 and C_2. In most cases this number is reduced for this test cases, except for the shapes graphs with C_2 and WorkingClass as target. Nevertheless the execution times are also reduced in these cases. Note that the magic shapes graph might count more shape constraints than the input shapes graph, since the procedure `Generate` adds *magic* shape constraints.

Table 6.4 provides more details about the validation. In the second column again the

Targets		C_1		C_2	
T	nodes	$\text{valid}_{2,3}$	N_{magic}	valid_3	N_{magic}
Musician($mozart$)	1	0	1	0	1
Musician($Actor$)	4872	0	6908	0	6908
Musician($Person$)	2267445	5248	2396852	5248	2396852
Actor($cameron$)	1	1	3	1	3
Actor($Actor$)	4872	614	253606	614	253606
Actor($Person$)	2267445	16645	2567841	16645	2567841
Movie($Film$)	143121	2449	302030	2449	302030
TranslatedMovie($Film$)	143121	5	303021	5	303021
Employee($bill$)	1	1	1	nm	3
Employee($mark$)	1	0	1	0	1
Employee($Actor$)	4872	6	4872	6	4879
Employee($Person$)	2267445	9290	2267445	nm	2273847
WorkingPerson($bill$)	1	1	2	nm	2
WorkingPerson($mark$)	1	0	2	0	2
WorkingPerson($Actor$)	4872	2	6908	2	6915
WorkingPerson($Person$)	2267445	2959	2396852	nm	2403194
WorkingClass($bill$)	1	0	2	0	2
WorkingClass($mark$)	1	1	7	1	7
WorkingClass($Actor$)	4872	0	7055	0	7056
WorkingClass($Person$)	2267445	40	2396883	40	2397188

Table 6.4: Details about the experiments with the SHACL-ASP validator.

number of focus nodes are given. The column '$\text{valid}_{2,3}$' shows the number of valid targets for C_1 under the 2- and 3-valued semantics, i.e. how many of the focus nodes validate to true. In case of C_1 this number is the same for the input shapes graph and the magic variant, since Theorem 43 states equivalence for consistent shapes graphs. Indeed, C_1 also contains recursion and negation, but they do not interact with each other and therefore C_1 is always consistent. This is not the case for C_2, it contains a dangerous shape name, namely Employee as there is a marked self-loop resulting in an odd cycle in the dependency graph of C_2. Because of the *employer* cycle in the data graph G_{asp} between *bill*, *bob* and *jim*, no consistent way of assigning Employee to these nodes can be found, which makes it inconsistent. Thus, there exists no 2-valued (supported and stable) model for these shapes graphs. However, the magic shapes algorithm allows to find a 3-valued model for 16 out of the 20 shapes graphs, since the targets are not related to the inconsistency, see column 'valid_3'. The magic version of the shapes graphs with targets containing the shape names Musician, Actor, Movie and TranslatedMovie are positive (do not contain negation) and are equivalent for C_1 and C_2. This is reflected by the numbers in the columns $|C_{\text{magic}}|$ of Table 6.3 and N_{magic} of Table 6.4. N_{magic} are the number of nodes that are recognizes as relevant by the magic shapes algorithm, i.e.

	C, G_{asp}	$C_{\mathsf{magic}}, G_{\mathsf{magic}}$	N_{magic}	$\lvert G_{\mathsf{magic}}\rvert$
$(C_1, \{\mathsf{Actor}(\mathit{cameron})\})$	768s	0.13s	3	461
$(C_1, \{\mathsf{WorkingClass}(\mathit{mark})\})$	822s	1.87s	7	26069
$(C_2, \{\mathsf{WorkingClass}(\mathit{mark})\})$	942s	2.31s	7	26069

Table 6.5: Experiments on reduced data graphs. The times in the first column are for validating the original shapes graph over G_{asp}, and in the second column for the magic shapes graph over the reduced data graph.

the nodes that occur in $V(M)$, where $G \cup M$ is the stable model of C_{generate}. Comparing N_{magic} to the focus nodes, it can be observed that it increased only by a few nodes. For the targets with shape names Employee, WorkingPerson and WorkingClass N_{magic} of C_2 increases compared to C_1. The reason is that there is an additional magic shape constraint magic_Employee $\leftarrow$ *employer*.magic_Employee in C_{generate}.

Finally, in shapes graphs where the target is not connected to recursion the magic shapes algorithm can eliminate recursion completely, e.g. targets with the shape name Musician. In other cases, the algorithm gets rid of the recursion associated with negation, reducing the complexity of validation (e.g. shapes graphs with C_2 and targets involving the shape names Musician, Actor, Movie and TranslatedMovie). This allows to follow a validation approach optimized for a fragment of SHACL (see [CFRS19b, FRV21]), as implemented by the validators SHACL2SPARQL and TRAV-SHACL[7].

Reduced data graph

Column N_{magic} shows the number of *nodes* relevant for validating the target. Consequently, these nodes with their properties are enough for a correct validation. Unfortunately, the SHACL-ASP validator still loads all 7 million facts for validation and does not leverage the data graph recognized as relevant, which is several orders of magnitude smaller. To show the potential this smaller data graph holds, G_{asp} was manually reduced to a smaller data graph for three test cases, such that only triples remain that contain a node from those in N_{magic}, this smaller data graph is called G_{magic}. Table 6.5 provides the results of validating the reduced data graph G_{magic} against the magic shapes graphs. For the restricted data graph the improvement of the execution time is even greater, the validation takes in the range of seconds (see column $C_{\mathsf{magic}}, G_{\mathsf{magic}}$) compared to $750s - 940s$ for validation of C, G_{asp}. This indicates that SHACL-ASP could be significantly improved by using the procedure `Generate` (i.e. the set of constraints C_{generate}) of the magic shapes algorithm and based on that querying only for the relevant part of the data graph from the triple store.

[7]It was not possible to run the magic shapes graphs with these validators because the parser seems to be incomplete and does not support shape constraints like $s \leftarrow s'$.

CHAPTER 7

Conclusion

In this thesis we examined whether the ideas underlying the magic sets transformation can be applied to SHACL shapes graphs and be used to improve the validation of unrestricted SHACL. The answer to this research question is:

> "*When the ideas underlying the magic sets transformation from logic programming are applied to SHACL shapes graphs, they facilitate a more effective and robust validation of shapes graphs, where the interaction of recursion and negation is not restricted.*"

To come to this conclusion we proposed the **magic shapes algorithm** for SHACL, which is based on the magic sets technique for non-ground Datalog$^\neg$ programs [FGL07]. The algorithm takes as input a shapes graph (C, T) and returns a new *magic* shapes graph (C_{magic}, T). The magic shapes graph optimizes the validation of the targets once by removing irrelevant constraints and furthermore by identifying the relevant part for validating the data graph. This enables a target oriented validation independent from the validation approach and even in the presence of unrestricted recursion and negation.

Further, we showed the **correctness** of the magic shapes algorithm. The main theorem states that in cases where the data graph is consistent with C, the validation results of (C_{magic}, T) are equivalent to the results of (C, T).

For the cases where the data graph is inconsistent with (C, T), we discussed how the magic shapes algorithm provides an effective way to perform **inconsistent-tolerant validation**.

In order to **evaluate the empirical evidence** of the approach a prototype implementing the magic shapes algorithm was developed. Experiments were performed with the SHACL-ASP validator, the only prototype that supports the validation of full SHACL, i.e. unrestricted interaction of recursion and negation. The experiments' results show that

the validation with the magic version of the shapes graph is significantly faster than with the input shapes graph, we discuss the details in the following section.

Discussion

The results of the experiments show that the magic shapes algorithm improves the execution times of the SHACL-ASP validator by at least 30% for our test cases. For the consistent test cases the magic shapes graph validates the same amount of targets as the input shapes graph. We showed that this is the case for consistent shapes graphs in Theorem 43. Further, for the test cases with an inconsistent input shapes graphs the magic shapes graph could still be validated in 16 out of 20 cases. This is the case if the inconsistency is not related to the target and prohibits to find a model for the whole shapes graph. The 2-valued models of the magic shapes graphs correspond to 3-valued models of the input shapes graphs, we show this in Theorem 49. For the experiments, we executed the shapes- and magic shapes graphs all on the same data graph. But with the C_{generate} constraints of the magic shapes graph it would be possible to identify the relevant shape atoms and reduce the data graph to these and their properties. For three test cases that have a single target node we manually reduced the data graph and achieved validation times around few seconds, compared to 750-900s for the input shapes graph. This could be a starting point for future research.

Outlook

Future work could explore how the magic shapes algorithm improves the other approach (besides stable model semantics from [ACO$^+$20]) that supports unrestricted recursion and negation from [CFRS19a].

Another research direction could be to improve the validators by retrieving the relevant data graph on-the-fly. The experiments we have conducted with the reduced data graph show that this is promising for further optimization of the existing validators. Since some prototypes, e.g. SHACL2SPARQL, retrieve facts from a data graph via SPARQL queries, it would be an option to find a suitable translation from the constraints in C_{generate} to SPARQL in such a way that SPARQL directly retrieves only the relevant part of the data graph from the endpoint to validate the targets.

In this thesis we focused on the magic sets transformation algorithm from [FGL07], but there are several variation of the magic sets technique. Thus, in future the different magic sets techniques could be compared and it could be investigated whether the advantages can also be applied to the SHACL language.

Bibliography

[ACO+20] Medina Andresel, Julien Corman, Magdalena Ortiz, Juan L Reutter, Ognjen Savkovic, and Mantas Simkus. Stable model semantics for recursive SHACL. In *Proceedings of The Web Conference 2020*, pages 1570–1580, 2020.

[AFGL12] Mario Alviano, Wolfgang Faber, Gianluigi Greco, and Nicola Leone. Magic sets for disjunctive datalog programs. *Artificial Intelligence*, 187:156–192, 2012.

[BMSU85] Francois Bancilhon, David Maier, Yehoshua Sagiv, and Jeffrey D Ullman. Magic sets and other strange ways to implement logic programs. In *Proceedings of the fifth ACM SIGACT-SIGMOD symposium on Principles of database systems*, pages 1–15, 1985.

[CFRS19a] Julien Corman, Fernando Florenzano, Juan L Reutter, and Ognjen Savkovic. SHACL2SPARQL: Validating a SPARQL endpoint against recursive SHACL constraints. In *ISWC Satellites*, pages 165–168, 2019.

[CFRS19b] Julien Corman, Fernando Florenzano, Juan L. Reutter, and Ognjen Savkovic. Validating SHACL constraints over a SPARQL endpoint. In *ISWC*. Springer, 2019.

[CRS18] Julien Corman, Juan L Reutter, and Ognjen Savković. Semantics and validation of recursive SHACL. In *International Semantic Web Conference*, pages 318–336. Springer, 2018.

[Dun92] Phan Minh Dung. On the relations between stable and well-founded semantics of logic programs. *Theor. Comput. Sci.*, 105(1):7–25, 1992.

[EGM97] Thomas Eiter, Georg Gottlob, and Heikki Mannila. Disjunctive datalog. 22(3):364–418, 1997.

[FGL07] Wolfgang Faber, Gianluigi Greco, and Nicola Leone. Magic sets and their application to data integration. *Journal of Computer and System Sciences*, 73(4):584–609, 2007.

[FRV21] Mónica Figuera, Philipp D Rohde, and Maria-Esther Vidal. Trav-SHACL: Efficiently validating networks of SHACL constraints. *arXiv preprint arXiv:2101.07136*, 2021.

[HO09] Wendy Hall and Kieron O'Hara. *Semantic Web*, pages 8084–8104. Springer New York, New York, NY, 2009.

[Ken02] Stuart Kent. Model driven engineering. In *International conference on integrated formal methods*, pages 286–298. Springer, 2002.

[LT99] Vladimir Lifschitz and Hudson Turner. Splitting a logic program. 01 1999.

[LZ04] Fangzhen Lin and Xishun Zhao. On odd and even cycles in normal logic programs. In *AAAI*, pages 80–85, 2004.

[Yue14] Jianan Yue. Transition from ebnf to xtext. *Alternation*, 1:1, 2014.