Informatics

SHACL validation of evolving RDF graphs

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Logic and Computation

eingereicht von

Dominic Jäger, BSc
Matrikelnummer 01634025

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Prof. Magdalena Ortiz
Mitwirkung: Dr.^{in} techn. Shqiponja Ahmetaj

Wien, 27. April 2023

Dominic Jäger | Magdalena Ortiz

Informatics

SHACL validation of evolving RDF graphs

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Logic and Computation

by

Dominic Jäger, BSc
Registration Number 01634025

to the Faculty of Informatics

at the TU Wien

Advisor: Prof. Magdalena Ortiz
Assistance: Dr.in techn. Shqiponja Ahmetaj

Vienna, 27th April, 2023

Dominic Jäger	Magdalena Ortiz

Erklärung zur Verfassung der Arbeit

Dominic Jäger, BSc

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, 27. April 2023

Dominic Jäger

Danksagung

Ich bedanke mich bei meinen Betreuerinnen Prof. Magdalena Ortiz und Dr.^{in} Shqiponja Ahmetaj für die Unterstützung beim Erstellen der Arbeit, die vielen interessanten Diskussionen und die zahlreichen hilfreichen Rückmeldungen. Durch diesen Prozess konnte ich sehr viel lernen.

Ich danke meiner Familie dafür, während der gesamten Studienzeit auf ihre Unterstützung gezählt haben zu können.

Daten für Experimente durch Projekt Yago und U.S. National Library of Medicine.

Acknowledgements

I would like to thank my advisors Prof. Magdalena Ortiz and Dr.^{in} Shqiponja Ahmetaj for supporting me while writing the thesis, the numerous interesting discussions and the helpful feedback. I could learn very much thanks to this.

I would like to thank my family for supporting me during the whole time of my studies.

Experiment data courtesy of Yago project and the U.S. National Library of Medicine.

Kurzfassung

Es gibt die Vision eines Internets, in dem enorme Mengen von Daten verknüpft sind und verwendet und ausgetauscht werden können. Um diese Vision wahr werden zu lassen, veröffentlichte das World Wide Web Consortium (W3C) den RDF-Standard zum Beschreiben von Informationen im Internet. Aufgrund vieler vorteilhafter Eigenschaften hat RDF weltweite Bekanntheit erlangt und sich zu einer Kerntechnologie des semantischen Internets entwickelt.

Das Prüfen der Korrektheit und Vollständigkeit von Daten ist dabei zu einer wesentlichen Herausforderung geworden. Beispiele dafür liegen in der Sicherheit für Industrieanwendungen oder beim Datenaustausch für Geschäftsanwendungen. Um diese Herausforderung zu meistern, hat das W3C den SHACL-Standard vorgestellt. Da aber SHACL eine relativ junge Sprache ist, sind viele andere Anwendungsfälle sind noch nicht betrachtet worden.

Darunter ist die Auswirkung von Änderungen auf den Datengraph für die Validierung in Betracht zu ziehen. Insbesondere betrachten wir die Situation, dass ein validierter Datengraph gegeben ist und wir wissen wollen, ob er valide bleibt, wenn wir Änderungen am Graph vornehmen. Das Problem dabei ist, dass Änderungen auf – potenziell sehr großen – Teilen des Graphen und eine Re-Validierung eine große Menge von Ressourcen benötigen kann. Außerdem ist es nützlich zu wissen, ob eine Reihe von Änderungen zu einem gültigen Graph führt, falls die Rückkehr zum ursprünglichen Graph schwierig ist, oder falls wir im System nicht die Berechtigung haben, Änderungen durchzuführen. Deshalb wollen wir einen Weg finden zu prüfen, ob eine Reihe von Änderungen einen gültigen Graphen liefert, bei dem das tatsächliche Durchführen der Änderungen nicht notwendig ist. Wir nennen dies *statische Validierung unter Änderungen*.

Um diese Herausforderung zu bewältigen, entwickeln wir zuerst eine formale Sprache zum Beschreiben von Änderungen an Datengraphen. Der Hauptbeitrag ist eine Reduktion des statischen Validierungsproblems zu SHACL-Validierung. Diese Reduktion nimmt eine Folge von Änderungen und eine Menge von SHACL-Bedingungen und transformiert sie in eine neue Menge von Bedingungen. Wir zeigen, dass diese neue Menge von Bedingungen den originalen Datengraphen genau dann und nur dann validiert, wenn der veränderte Datengraph die originalen Bedingungen validiert. Um diese Reduktion zu ermöglichen, werden wir außerdem existierende SHACL-Formalisierungen aus der Literatur erweitern. Abschließend zeigen wir eine prototypische Implementierung der Technik, die uns hilft, ihr Potenzial zu verstehen.

Abstract

Imagine a web of data and services in which huge amounts of data are linked and can be used and exchanged. To fulfill this vision, the World Wide Web Consortium (W3C) presented the RDF standard to describe information in the Web. Due to multiple favorable properties, RDF gained a world-wide audience and developed into a core technology for the Semantic Web.

As RDF is a very flexible language, checking the correctness and completeness of data graphs is a key challenge. Examples for this are safety for industry applications or integration between various enterprise applications. To solve this challenge, the W3C proposed the SHACL standard. As SHACL is a relatively young language, several use-cases have not yet been addressed.

Among them is to take the effect of updates on the graph into account for validation. Specifically, we look at the situation where we have a validated graph and want to know if it remains valid when performing updates on the graph data. The problem is that performing updates on – potentially large – parts of the graph and then re-validating it can require vast resources. Furthermore, it is beneficial to know if a set of updates leads to a valid graph in case that returning to the initial state is difficult, or that we do not have sufficient permissions in a system to perform the updates. Therefore, we want to find a way to verify that a set of updates yields a valid graph that avoids actually performing the updates on the graph. We call this *static validation under updates*.

To tackle this challenge, we develop a suitable formal update language for describing updates on data graphs. The main contribution is a reduction of the static validation problem to SHACL validation. This reduction takes a sequence of actions and a set of SHACL constraints and transforms them into a new set of constraints. We prove that this new set of constraints validates the original graph if and only if the updated data graph validates the original constraints. This technique requires extending SHACL formalizations that have already been considered in the literature. Finally, we provide a proof-of-concept implementation of the technique that enables us to understand its feasibility and potential.

Contents

CHAPTER 1

Introduction

In this chapter we give some motivation for the thesis and state the main problem that we want to solve. Then we explain what our contributions to this research area are and what related work has already been done.

Motivation

Imagine a web of data and services in which huge amounts of data are linked and can be used and exchanged. This is the vision of the World Wide Web Consortium (W3C) [36], an organization that works on standards for the web [1].

One tool to achieve this is the *Resource Description Framework* (RDF). It is a W3C recommendation to describe information on the web, and a core technology for the *Semantic Web* [7]. RDF has become attractive to researchers as well as practitioners, and gained a world-wide audience. [7].

In this framework, data is represented using subject-predicate-object triples, such as $\text{loves}(jacob, charlie)$ [17]. In other words, we say that things have properties with particular values and it turns out that this is a way to describe data that is suitable for most machine-processed data [9]. RDF triples form directed graphs, which we call *data graphs*. Figure 1.1 shows an example for a data graph. The definition with triples gives RDF multiple favorable properties. Among them is that it is easy to compose multiple

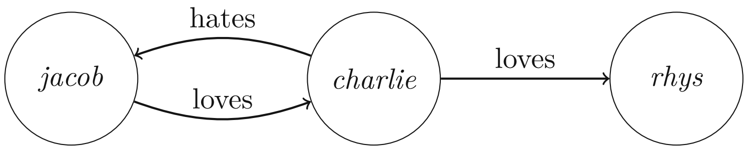


Figure 1.1: An example RDF graph

RDF graphs into a larger one. It is easy to integrate various sources of data, and existing data models may evolve to meet new requirements. Specifically, this allows to add more triples to the graph with ease [27].

One natural problem is to query RDF data [7]. We can do so using the SPARQL query language. SPARQL supports queries with features such as aggregation, negation and subqueries [22]. One way to use this language is via online services that provide answers by processing a given SPARQL query.

While RDF is a very flexible language [27], checking the correctness and completeness of data graphs is a key challenge [15]. Safety and performance are requirements of particular importance for industry and enterprise applications [27]. One specific situation where this is important is application integration, where different software components produce and consume data and must smoothly work together. A standard for this may reduce both complexity and cost [35]. Today, huge amounts of data get – partly automatically – generated and exposed on the Web [27]. As a consequence, RDF databases may be of considerable size, that is, in the order of millions or billions of triples. It is still desirable to check various properties, and to do this in regular intervals. Thus, it must be possible to perform the validation in a reasonable amount of time [35].

To solve this, the W3C published the Shapes Constraint Language (SHACL) recommendation in 2017 [26]. SHACL allows describing the form that a data graph must adhere to using *shapes* and *constraints*. Shapes are parts of constraints, and they can be represented as graphs as well. As an example, we have the shape $\exists \text{loves}.\top$ as part of the constraint $\mathit{isDevoted} \leftrightarrow \exists \text{loves}.\top$. This constraint says that *isDevoted* holds for an individual if and only if there is at least one other individual that it loves. In Figure 1.1, the constraint holds for *jacob* and *charlie*, but not for *rhys*.

Problem Statement and Research Question

SHACL is a relatively young language. Thus, validators for it are currently in development and many challenges and use-cases have not yet been addressed.

Among them is to take the effect of updates on the graph into account for validation. One relevant situation is that we have a validated graph and want to know if it remains valid when performing updates on the graph data. The problem is that performing updates on – potentially large – parts of the graph and then re-validating it can require vast resources. Furthermore, it is beneficial to know if a set of updates leads to a valid graph in case that returning to the initial state is difficult, or that we do not have sufficient permissions in a system to perform the updates.

Therefore, we want to find a way to verify that a set of updates yields a valid graph that avoids actually performing the updates on the graph. We call this *static validation under updates*.

The main goal of this thesis is to study the validation of SHACL constraints over RDF graphs that are subjects of updates.

The research questions are stated and explained in the following.

RQ1: How can we suitably describe updates on RDF graphs? Can we build an adequate update language, building on proposals from the literature?

We need syntax and semantics for an update language, that can express modifications on RDF data graphs. We want a language that supports addition and removal of classes, as well as properties. In other words, it should be able to add unary and binary atoms to the data graph. We want to concatenate multiple actions and to use variables.

RQ2: Is it possible to reduce validation under updates to validation over a static graph without performing the updates? Moreover, can such a validation be realized using existing validators?

We want a technique that can solve the static validation problem. In other words, we want to find a way to check if a set of updates of graph data results in a valid set of constraints that avoids modifying the data graph. This technique should capture the effects of the updates that we define in the first research question.

We want to have proof of the correctness of the reduction. This means that we need to be sure that the original graph validates the transformed set of constraints if and only if the updated data graph validates the original constraints. We want to have examples for this static validation technique.

Several validator implementations for different fragments of W3C SHACL Core already exist. We want to reuse one of these existing validators, and know if they are a sufficient tool for the reduced problem.

RQ3: Which features of SHACL can be supported in this setting that are not covered by existing formalizations, but that can increase the applicability of the approach?

The SHACL W3C recommendation contains multiple features that are not addressed in existing formalizations [26] [3] [15]. However, it is desirable to have a formalization that can express further important situations. In particular, this could include comparing things and being able to reason about numbers, dates, and strings. We want to have examples that demonstrate such situations and underline the usefulness of possible extensions of existing formalizations.

RQ4: Can we gain experimental data that indicates the effectiveness of the static validation technique? Which additional artifacts do we need to get this data?

To be able to apply the static validation technique from Research Question 2 in practice, we want to have a prototype implementation of the transformation technique. Furthermore, we want to be able to get experimental data that shows whether the static validation technique is useful in practice or not. We want to have proof-of-concept implementations of other relevant artifacts, that we need to get the experimental data as well. The prototype implementation should reuse existing SHACL validators as far as possible.

Contributions

Solving the stated problem and answering the research question requires addressing multiple problems, and we do so by contributing various new ideas.

Inspired by an action language for description logics [2], we **create an update language to formalize changes in data graphs**. Our update language, however, uses SHACL syntax and semantics. It allows adding and removing classes as well as properties, and we reason about sequences of actions.

As the main contribution, we **define a transformation that allows to reduce static validation to SHACL validation** after an idea used in the context of description logics [2]. This transformation takes a set of SHACL constraints and transforms them into a different set of constraints that take the actions into account. Again, our transformation definition uses SHACL instead of description logics. We prove its correctness, which means that we show that the original graph validates the transformed set of constraints if and only if the updated data graph validates the original constraints. In addition, we demonstrate the static validation technique in various examples.

One specific part of the transformation definition requires us to **extend the syntax and semantics of existing SHACL formalizations**. We do this by making path expressions more powerful. In addition, we add predicates and literals that bring our formalization closer to the W3C SHACL recommendation. We show in multiple examples what this extended formalization can express.

Furthermore, we provide a prototype implementation of the transformation based on the Apache Jena project, and a program that can parse actions from our update language and apply them to a graph. Finally, we experiment with different existing RDF data sets of various sizes to show the effectiveness of our approach.

Related work

The W3C has recommendations for syntax [17] and semantics [23] for RDF. The SHACL standard for graph validation has been published by the W3C in 2017 [26]. As an introduction to SHACL, there is also a book available [27]. Suggestions for improvements of SHACL are currently in discussion [24].

Some parts of the SHACL standard are not yet properly defined. For example, the standard leaves the problem of recursive constraints open [26]. To find a remedy for this, different logic-based approaches to formalize the SHACL standard have been presented.

Formal semantics for a restricted set of SHACL that handle arbitrary recursion and include abstract syntax for SHACL was suggested in 2018 [15]. These semantics are called supported-model semantics [4]. An approach that uses ideas from Answer Set Programming has been introduced a few years later and is called stable model semantics [4]. These approaches allow devising better algorithms and to compare the expressiveness of SHACL to other languages.

Another topic that is left open in the SHACL standard is the explanation of violated constraints [26]. There has been work based on logic-based abduction and database repairs to improve this. The main idea is to use a collection of additions and deletions that can repair the graph [3]. Further research has been done concerning RDF graphs that are only exposed via a SPARQL endpoint. Specifically, algorithms have been proposed that allow performing SHACL validation by using SPARQL queries [16].

One important language that SHACL is compared against is *Shape Expressions (ShEx)*. Both languages have their roots in the same W3C working group and share the goal of describing RDF data for validation. However, they considerably differ in the way to achieve this vision. SHACL describes rules that must be fulfilled and reports violations, while ShEx describes the RDF graph more directly & compactly and reports the validated nodes. SHACL therefore is similar to validation of XML using Schematron. Finally, ShEx is a community group effort and not officially a W3C recommendation, whereas SHACL Core is an official recommendation [27].

The idea of describing constraints on data graphs in the presence of updates on the graph has already been studied before SHACL became an official recommendation in 2017. The question is if a set of constraints expressed using description logics is still fulfilled after varying updates have been performed on the graph data. To answer such questions, a technique that modifies description logic constraints to simulate updates on graph-structured data has been proposed [12]. It has been shown that such problems can be reduced to satisfiability of knowledge bases, and that satisfiability of knowledge bases for an important description logic is NExpTime-complete [2]. This simulation technique is significant for the thesis and a cornerstone for the update language and the technique to capture the action effects that we plan to implement ourselves. It is pivotal to note that validation of SHACL constraints under updates has not been studied yet.

The computational complexity of various problems around SHACL has been studied as well. SHACL satisfiability is the problem „Is there a graph that validates the constraints?“ The satisfiability problem is undecidable for full SHACL. The problem if validation of one set of constraints implies validation of another set of constraints is called SHACL containment and, for full SHACL, undecidable as well [31] [30]. While validation of some fragments of SHACL is tractable, validation of a fixed graph using full SHACL is NP-complete, too [15]. [28] translate the W3C SHACL specification into description logics and show soundness and completeness for restricted parts of it.

Several SHACL validator implementations are available. However, their capabilities are significantly different. One implementation originates from a paper that uses SPARQL to validate SHACL constraints [16]. It supports only a limited subset of the SHACL W3C recommendation. Among the missing features are at the time of writing classes, which we need to use heavily [13]. The Python implementation [34] of Trav-SHACL [20] does not support sh:or, which we need. A different implementation implements the W3C recommendation using pure JavaScript [33]. The implementation that we will use is the validator of the Apache Jena project. This validator supports all SHACL core constraint and also SPARQL-based constraint components [6] [5].

Thesis structure

Chapter 2 In this chapter, we outline important notions of the RDF, SPARQL and SHACL W3C recommendations.

Chapter 3 As existing SHACL formalizations are not expressible enough to perform the static validation under updates, we use this chapter to present syntax & semantics for an extended SHACL formalization.

Chapter 4 Here, we first present syntax & semantics for an update language that formalizes updates on data graphs. This language is based on SHACL. Thereafter, we propose a transformation of constraints to capture the effects of those actions. In addition, we give examples about what this formalization can express.

Chapter 5 In this chapter, we prove that the transformation of constraints is correct, and we provide examples for the technique.

Chapter 6 This chapter contains details about the implementation of the transformation and experimental results based on two different data sets.

Chapter 7 The final chapter contains a summary about the results and ideas for future work.

CHAPTER 2

Preliminaries

This chapter contains an explanation of the basic notions that are relevant for the subsequent chapters. Specifically, this includes details about the Resource Description Framework (RDF), SPARQL Protocol and RDF Query Language (SPARQL) and the Shapes Constraint Language (SHACL). Furthermore, it explains some more advanced ideas from the W3C recommendations that are relevant for or rather act as motivation for later parts of this thesis.

2.1 RDF and SPARQL

The Resource Description Framework (RDF) has been designed to represent information on the internet [17]. The first drafts turned into an actual recommendation by W3C in 1999. [27] An RDF graph is a set of triples like loves $(jacob, charlie)$. Each triple has a subject, a predicate, and an object and can be visualized as in Figure 2.1 [17]. All three parts of the triple are Universal Resource identifiers (URIs), which usually look like links on a web page [9]. An example for a URI is `http://www.w3.org/1999/02/22-rdf-syntax-ns#type`. Within the thesis, we often abstract away from this. Figure 1.1 from the previous chapter shows an example for a graph that can be formed with the abstraction. The W3C recommendation uses the Turtle syntax to present RDF graphs [26]. An example for a data graph in Turtle syntax can be seen in Listing 2.1.

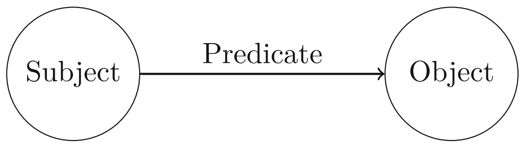


Figure 2.1: Visualization of an RDF triple

Listing 2.1: A data graph in Turtle syntax [8]

```
<#green-goblin>
    rel:enemyOf <#spiderman> ;
    a foaf:Person ;    # in the context of the Marvel universe
    foaf:name "Green Goblin" .
```

Data types are used for RDF *literals* and are represented by IRIs. While various data type definitions are allowed, there is a list of recommended data types in the W3C recommendation. This list includes common types like string, boolean, integer, float and dateTime [17].

The SPARQL Protocol and RDF Query Language (SPARQL) is a query language for RDF data [19]. Example 1 demonstrates what a SPARQL query and its result look like.

Example 1 ([22])**.** Suppose we have the data from Listing 2.2 and the query from Listing 2.3.

Listing 2.2: A data graph about a book

```
    <http://ex.com/book1> <http://ex.com/title> "SPARQL Tutorial" .
```

Listing 2.3: SPARQL query to find book titles

```
    SELECT ?title
    WHERE
    {
      <http://ex.com/book1> <http://ex.com/title> ?title .
    }
```

Then the result is the following table:

title
„SPARQL tutorial“

SPARQL queries can be performed at SPARQL endpoints. One such endpoint is `https://query.wikidata.org/`. A slightly more complex example for a SPARQL query that can be performed at that endpoint can be seen in Listing 2.4. It selects instances (wdt:P31) or subclasses (wdt:P279) and coordinates (wdt:P625) of hospitals (wd:Q16917) in Germany (wd:Q183). Syntactic details like the property path syntax that is used in line 2 of Listing 2.4 are defined and more thoroughly explained in the W3C SPARQL recommendation [22].

Listing 2.4: A SPARQL query about hospitals in Germany.

```
SELECT DISTINCT * WHERE {
  ?item wdt:P31/wdt:P279* wd:Q16917;
        wdt:P17 wd:Q183;
        wdt:P625 ?geo .
}
```

Listings 2.5 and 2.6 show a feature that will act as motivation for a new idea in chapter 3. The query in Listing 2.5 selects hospitals, but filters out those that are in Germany. Note that the same ?item appears in lines 2 and 4.

Listing 2.5: SPARQL query for hospitals that are not in Germany using filter

```
1 SELECT DISTINCT * WHERE {
2   ?item wdt:P31/wdt:P279* wd:Q16917;
3         wdt:P625 ?geo .
4         FILTER NOT EXISTS { ?item wdt:P17 wd:Q183 }
5 }
```

The query in Listing 2.6 gives the same result as the query in Listing 2.5 using the minus operator.

Listing 2.6: SPARQL query for hospitals that are not in Germany using minus

```
SELECT DISTINCT * WHERE {
  ?item wdt:P31/wdt:P279* wd:Q16917;
        wdt:P625 ?geo .
  MINUS {
  ?item wdt:P31/wdt:P279* wd:Q16917;
        wdt:P17 wd:Q183;
        wdt:P625 ?geo .
  }
}
```

2.2 Shapes Constraint Language

Various definitions from this thesis and specifically from Chapter 3 are inspired by ideas from different documents related to the W3C. This section explains parts of the W3C Shapes Constraint Language (SHACL) recommendation [17] that are of specific relevance for us.

Listing 2.7 shows two syntactic details from the SHACL recommendation that are of special importance for us.

Listing 2.7: An example shape graph [26]

```
 1 ex:PersonShape
 2         a sh:NodeShape ;
 3         sh:targetClass ex:Person ;
 4         sh:property [
 5                 sh:path ex:ssn ;
 6                 sh:maxCount 1 ;
 7                 sh:datatype xsd:string ;
 8         ] ;
 9         sh:property [
10                 sh:path ex:worksFor ;
11                 sh:class ex:Company ;
12                 sh:nodeKind sh:IRI ;
13         ] ;
```

First, using the token „a“ as a predicate is a shorthand for rdf:type in the Turtle syntax [8]. Declaring rdf:type means that something is an instance of a class [10]. We will introduce classes in Chapter 3

Second, it is interesting to compare the definition of targets between the W3C SHACL recommendation and the formalization from Chapter 3. In Listing 2.7, the target nodes are defined in line 3 using sh:targetClass. This means that the definition of targets happens in the shapes graph. In Chapter 3, the constraints will be one set and the targets will be another set.

2.2.1 Core Constraint Components

SHACL has different *Core constraint components*. Their goal is to capture the most important use cases and requirements, while keeping the size of the language reasonable. They include i.a. constraint components about the types of value nodes, cardinality or logical constraint components. [26]

From the core constraint components, we first show some *shape-based constraint components*. The keyword sh:path in shapes graphs is of particular importance.

> A *node shape* is a shape in the shapes graph that is not the subject of a triple with sh:path as its predicate. It is recommended, but not required, for a node shape to be declared as a SHACL instance of sh:NodeShape [26].

Allowing to omit the declaration will turn out to be useful for the implementation. A node shape declaration can be seen in line 3 of Listing 2.7.

> A *property shape* is a shape in the shapes graph that is the subject of a triple that has sh:path as its predicate. A shape has at most one value for sh:path. Each value of sh:path in a shape must be a well-formed SHACL

property path. It is recommended, but not required, for a property shape to be declared as a SHACL instance of sh:PropertyShape. SHACL instances of sh:PropertyShape have one value for the property sh:path [26].

We can see two occurrences of sh:path in Listing 2.7, and thus two property shapes, even though the explicit declaration as sh:PropertyShape is missing.

Most constraint components use *value nodes*. Their definition has two cases [26]:

Node Shapes For each node shape, the value node is a set with the focus node as the single member.

Property Shape The value nodes are the set of nodes in the data graph that can be reached from the focus node with the value of the sh:path.

sh:property asserts that each value node has a given property shape [26] and it specifically does so in Listing 2.7. Note that the brackets in Listing 2.7 denote blank nodes, or to be precise a blankNodePropertyList [8].

The second important types of constraints, that are part of the W3C SHACL recommendation core constraints, are *value type constraint components*. These constraint components restrict the types of value nodes. There are three different types of them [26]:

1. sh:class: Each value node is a SHACL instance of a given type. The values are IRIs. These can be compared to the classes of the SHACL formalization of Chapter 3.
2. sh:datatype: Each value node is of a given datatype.
3. sh:nodeKind: Node kinds are IRI, blank node and literal. This component declares that each value node is of a given node kind.

2.2.2 Node Expressions

Some features of *node expression* serve as motivation for the formalization of SHACL in Chapter 3 and updates in Section 4.1. Node expressions are a feature that has been proposed in the W3C working group note „SHACL Advanced Features“ from 2017. They are not part of the SHACL W3C recommendation from 2017 [26]. Working group notes are draft documents and must be considered work in progress [24]. There is a more recent revision of this document called „SHACL Advanced Features 1.1“. It is a draft community group report from 2021 [25].

Node expressions are declared as RDF nodes in a shapes graph. The idea of these expressions is to start at the focus node and then obtain a set of nodes from it. Node expressions can be combined. Therefore, we could, for example, first get all values of

Expressions	Syntax	Semantics
Focus Node	sh:this	The list consisting of the current focus node.
Path	Blank node with sh:path	The values of a given property path.
Minus	Blank node with sh:minus and sh:nodes	The input nodes except those that are in another "minus" list
Intersection	Blank node with sh:intersection	The intersection of two or more input node lists.
Filter	Blank node with sh: filterShape	The sub-list of the input nodes that conform to a given shape.

Table 2.1: Syntax and semantics for selected node expressions [25]

Node Expressions	Focus Node	Constant Term	Exists	If	Filter	Function	Path	Intersection	Union	Minus	Distinct	Min	Max	Sum	Group Concat	OrderBy	Limit	Offset	SPARQL ASK	SPARQL SELECT
AF	✓	✓			✓	✓	✓	✓	✓											
AF 1.1	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 2.2: Comparison of available node expressions in SHACL Advanced Features Working Group Note [26] and Draft Community Group Report [25]

a property of a focus node and then restrict the result set according to a second node expressions. Later, the result sets can be used for expression constraints and SHACL rules, which are both features that are – just like node expressions – in the „SHACL Advanced Features“ working group note [24] but not in the SHACL recommendation [26]. Table 2.1 contains syntax and semantics for some node expressions that most closely resemble ideas that the formalization from Chapter 3 adds on top of existing formalizations. Table 2.2 contains an exhaustive comparison of the available node expressions in the two versions of the proposal.

Listing 2.8 contains a shapes graph example for a minus expression. By definition „sh:minus returns all values of the property sh:children except those that are also values of ex:sons.“ [25]. The value of a property refers to the object of the triple [26]. Consequently, it's the objects of the triples that are produced in Listing 2.8. This fits together with the motivation that we stated for node expressions. Generally, dollar signs $ and question marks ? in SPARQL code denote query variables, and it is possible to select a subset of these query variables as output [22]. Additionally, in the proposal the $ denotes the current focus which produces a list of RDF nodes [25]. We can compare this with the definition of focus nodes in table 2.1. Therefore, Listing 2.9 contains a SPARQL expression of which the result is comparable to the node expression of Listing 2.8.

Listing 2.8: Shapes graph for a minus node expression [25]

```
[
    sh:nodes [ sh:path ex:children ] ;
    sh:minus [ sh:path ex:sons ] ;
] .
```

Listing 2.9: Comparable SPARQL expression for Listing 2.8 [25]

```
{
    $this ex:children ?result .
    MINUS {
        $this ex:sons ?result .
    }
}
```

From the definitions we see that sh:and is as a core constraint component concerned with the validation of *shapes* [26, sh:and] while sh: intersection is as a *node expressions* concerned with selecting a set of nodes. In other words, minus expressions, intersection expressions and generally node expressions exist to derive a set of nodes from a given focus node, whereas sh:and exists for the core validation task.

It is possible to create even more advanced node expressions. For example, Listing 2.10 contains a node expression that creates a conditional.

Listing 2.10: Example for an if node expressions [25]

```
[
    sh:if [ sh:exists [ sh:path ex:spouse ] ] ;
    sh:then "married" ;
    sh:else "not married" ;
] .
```

Finally, node expressions are used in SHACL rules, which allow update on graphs. One example is to infer for instances of ex:Rectangle for which ex:width equals ex:height that they are also instances of ex:Square [24]. These features are an additional motivation for our formalization of updates.

CHAPTER 3

Extended SHACL formalization

In this chapter, we introduce syntax and semantics for our SHACL formalization. We take the formalization from [3] and extend the possibilities to express properties and predicates. At the end of the chapter, we demonstrate in a few examples how these new constructs can be used and what the formalization is able to express.

3.1 Syntax

Let $\mathbb{I}$ be the set of *individuals*. It is inspired by the Internationalized Resource Identifiers (IRIs) in RDF [26]. In figures of graphs, such as for example Figure 3.1, we will denote individuals with circles. Let $\mathbb{L}$ the set of *literals*. Literals denote i.a. numbers, dates, and strings. Thus, literals possess a natural data type. Section 3.3.2 contains two examples for this. In figures, for example Figure 6.4, we will denote literals with rectangles. Let $\mathbb{C}$ be the set of *classes*. Individuals can belong to classes. In figures, we denote classes with rectangles on top of individuals. This is denoted by *unary atoms* of the form $\mathrm{B}(c)$ where $\mathrm{B} \in \mathbb{C}$ and $c \in \mathbb{I}$. Let $\mathbb{O}$ be the set of *object property names*. Object properties connect individuals to other individuals. This is denoted by *binary atoms* of the form $\mathrm{p}(c, d)$ where $\mathrm{p} \in \mathbb{O}$ and $c, d \in \mathbb{I}$. Let $\mathbb{D}$ be the set of *data property names*. Data properties connect individuals to literals. This is denoted by *binary atoms* of the form $\mathrm{d}(c, l)$ where $\mathrm{d} \in \mathbb{D}$, $c \in \mathbb{I}$ and $l \in \mathbb{L}$. A *data graph* G is a finite set of unary and binary atoms. [3]

Example 2. This is a valid data graph:

$$G = \{\mathrm{Human}(joanne), \mathrm{worksFor}(joanne, acme), \mathrm{age}(joanne, 52)\}$$

with $joanne, acme \in \mathbb{I}$, $52 \in \mathbb{L}$, $\mathrm{Human} \in \mathbb{C}$, $\mathrm{worksFor} \in \mathbb{O}$, and $\mathrm{age} \in \mathbb{D}$.

Let $\mathbb{S}$ be the set of *shape names* and $\mathbb{V}$ be the set of *variables*. The sets $\mathbb{I}$, $\mathbb{L}$, $\mathbb{C}$, $\mathbb{O}$, $\mathbb{D}$, $\mathbb{S}$ and $\mathbb{V}$ are countably infinite and mutually disjoint. It is especially important to distinguish between shape names and classes. A *shape atom* is an expression of the form $s(a)$ with $s \in \mathbb{S}$ and $a \in \mathbb{I}$. Then atoms $\mathrm{B}\,(c)$ differ from shape atoms $s(a)$ because $\mathbb{S}$ is disjoint from $\mathbb{C}$.

We define *(basic) object properties* r and *object path expressions* E according to the following syntax:

$$\begin{aligned} \mathrm{r}, \mathrm{r}' &::= \mathrm{p} \mid (c, d) \mid \mathrm{r}^- \mid \mathrm{r} \cup \mathrm{r}' \mid \mathrm{r} \cap \mathrm{r}' \mid \mathrm{r} \setminus \mathrm{r}' \\ E, E' &::= \mathrm{r} \mid E \cup E' \mid E \cdot E' \mid E^* \end{aligned}$$

where $\mathrm{p} \in \mathbb{O}$, and $c, d \in \mathbb{I}$. We call p^- the *inverse object property name* of p and r^- the *inverse basic object property* of r. We define the *structural complexity* of a path expression as the number of operators $*, \cdot, \cup$ that construct it. This does not count the potentially existing operators $\cup$, $\cap$ and $\setminus$ within each basic object property r.

We define *shape expressions* (or simply *shapes*) according to the following syntax [3, p. 13]:

$$\phi, \phi' ::= \top \mid \mathrm{B} \mid t \mid \phi \wedge \phi' \mid \neg\phi \mid \geq_n E.\phi \mid \mathrm{EQ}\,(E, E') \mid \circledast\,(\mathrm{d}, \psi) \mid s$$

where $\mathrm{B} \in \mathbb{C}$, $t \in \mathbb{I} \cup \mathbb{V}$, $\mathrm{d} \in \mathbb{D}$, $\psi \in \mathbb{D} \cup \mathbb{L}$, $s \in \mathbb{S}$, n is a positive integer, E and E' are path expressions, and $\circledast$ is a binary predicate . Section 3.3 explains the motivation behind the new expression $\circledast\,(\mathrm{d}, \psi)$. Following description logics, we call a shape expression *concept expression* or simply *concept* if it does not contain any shape names $s \in \mathbb{S}$. We call a shape (or concept) *ground* if $t \in \mathbb{I}$. We may use infix notation for predicates like $<$ and $\geq$.

Example 3. These are two valid shape expressions:

$$x \wedge \mathrm{Human} \wedge \geq_2 ((\mathrm{attends} \cup \mathrm{listensTo}) \cdot \mathrm{teachedAt})\,.\,(\mathrm{University} \vee \mathrm{College})$$
$$\geq_3 (\mathrm{dislikes} \setminus \mathrm{hates})\,.\,(\mathrm{Adult} \wedge \mathrm{Serious}) \wedge \mathrm{EQ}\,(\mathrm{admires} \cup \mathrm{loves}, \mathrm{likes}) \wedge (\mathrm{age} > 17)$$

Allowing set intersection & difference in object properties is a new contribution that is required for the transformation from Section 4.2. Like [3, p. 13], we will write $\phi \vee \phi'$ instead of $\neg(\neg\phi \wedge \neg\phi')$, $\phi \rightarrow \phi'$ instead of $\neg\phi \vee \phi'$ or rather $\neg(\phi \wedge \neg\phi')$, $\exists E.\phi$ instead of $\geq_1 E.\phi$, $\forall E.\phi$ instead of $\neg \geq_1 E.\neg\phi$.

A *constraint* is an expression $s \leftrightarrow \phi$ where $s \in \mathbb{S}$ and ϕ is a shape expression. When a shape atom $s(a)$ is designated as *target* then the shape s must be validated at individual a. A *shape document* is a pair (C, T) where C is a set of constraints and T is a set of targets. Each shape name appearing in C appears exactly once on the left-hand side of a constraint in C [3, p. 13]. The following is an example for a constraint

$$\mathit{AdultShape} \leftrightarrow \mathrm{Person} \wedge (\mathrm{age} > 18)$$

$$\begin{aligned}
\llbracket \top \rrbracket^I &= V(I) & \llbracket \neg\phi \rrbracket^I &= V(I) \setminus \llbracket \phi \rrbracket^I \\
\llbracket a \rrbracket^I &= \{a\} & \llbracket \phi_1 \wedge \phi_2 \rrbracket^I &= \llbracket \phi_1 \rrbracket^I \cap \llbracket \phi_2 \rrbracket^I \\
\llbracket \mathrm{B} \rrbracket^I &= \{a \mid \mathrm{B}(a) \in I\} & \llbracket \phi_1 \vee \phi_2 \rrbracket^I &= \llbracket \phi_1 \rrbracket^I \cup \llbracket \phi_2 \rrbracket^I \\
\llbracket s \rrbracket^I &= \{a \mid s(a) \in I\}
\end{aligned}$$

$$\llbracket \geq_n E.\phi \rrbracket^I = \left\{ a \mid \left| \left\{ (a,b) \in \llbracket E \rrbracket^I \text{ and } b \in \llbracket \phi \rrbracket^I \right\} \right| \geq n \right\}$$

$$\llbracket \mathrm{EQ}\,(E, E') \rrbracket^I = \left\{ a \mid \forall b : (a,b) \in \llbracket E \rrbracket^I \text{ iff } (a,b) \in \llbracket E' \rrbracket^I \right\}$$

$$\llbracket \circledast\,(\mathrm{d}, \psi) \rrbracket^I = \begin{cases} \left\{ a \mid \forall l \in \mathbb{L} \left((a,l) \in \llbracket \mathrm{d} \rrbracket^I \implies \circledast\,(l,\psi) \right) \right\} & \text{if } \psi \in \mathbb{L} \\ \left\{ a \mid \forall l, l' \in \mathbb{L} \left((a,l) \in \llbracket \mathrm{d} \rrbracket^I \wedge (a,l') \in \llbracket \psi \rrbracket^I \implies \circledast\,(l,l') \right) \right\} & \text{if } \psi \in \mathbb{D} \end{cases}$$

Table 3.1: Unary part of definition of the evaluation function $\llbracket \cdot \rrbracket^I$ [3, p. 14] [4, p. 1572] with I an assignment, $a, b \in \mathbb{I}$, $l \in \mathbb{L}$, $\mathrm{B} \in \mathbb{C}$, $l \in \mathbb{L}$, $\mathrm{d} \in \mathbb{D}$, $s \in \mathbb{S}$, ϕ, ϕ_1, ϕ_2 shapes, E, E' object path expressions, $\circledast$ a predicate, $\psi \in \mathbb{D} \cup \mathbb{L}$

3.2 Semantics

Like [3], we follow the semantic from [15] which is known as supported-model semantic [4]. Evaluation is done with a function $\llbracket \cdot \rrbracket^I$ that maps shape expressions ϕ to a set of individuals, and it maps basic object properties r and path expressions E to a set of pairs of individuals or literals. The definition of $\llbracket \cdot \rrbracket^I$ can be seen in Tables 3.1 and 3.2. By this definition, the first shape expression in Example 3 describes some human that listens to or attends something (like a lecture) that is taught at a university or college. Section 4 contains more examples that illustrate the function $\llbracket \cdot \rrbracket^I$. For instance, Example 21 demonstrates how shapes of the form $\mathrm{EQ}(E, E')$ are evaluated under actions. We denote with $V(I)$ a finite subset of $\mathbb{I}$ that contains the individuals of G. This means that even if $G = \emptyset$, then some individuals may still be present in $V(I)$. We will motivate and refine this definition in Section 4.3, as further information about updates on graphs are required for it.

Let L be a set of shape atoms such that a occurs in $V(I)$ for each $s(a) \in L$. Then an *assignment* for a data graph G is defined as set $I = G \cup L$. An assignment I for G is a *model* of a constraint $s \leftrightarrow \phi$ if $\llbracket \phi \rrbracket^I = s^I$, for which we write $I \models s \leftrightarrow \phi$. Furthermore, I is a model of C if it is a model for all constraints in C, for which we write $I \models C$. The graph G *validates* (C, T) if and only if there exists an assignment $I = G \cup L$ for G such that (i) I is a model of C, and (ii) $T \subseteq L$ [3, p. 13]. Examples 6 and 9 demonstrate validation of a shape document.

Considering the convention in Section 3.1, we can furthermore evaluate $\llbracket \forall E.\phi \rrbracket^I = \{a \mid \forall b : (a,b) \in \llbracket E \rrbracket^I \text{ implies } b \in \llbracket \phi \rrbracket^I\}$ [4]. The $\circ$ denotes the usual composition of relations. This composition of relations is – as part of the evaluation of selected path expressions – demonstrated in Example 4.

$$\begin{aligned}
[\![(a,b)]\!]^I &= \{(a,b)\} & [\![E \cup E']\!]^I &= [\![E]\!]^I \cup [\![E']\!]^I \\
[\![\mathrm{p}]\!]^I &= \{(a,b) \mid \mathrm{p}\,(a,b) \in I\} & [\![E \cap E']\!]^I &= [\![E]\!]^I \cap [\![E']\!]^I \\
[\![\mathrm{p}^-]\!]^I &= \{(a,b) \mid \mathrm{p}\,(b,a) \in I\} & [\![E \setminus E']\!]^I &= [\![E]\!]^I \setminus [\![E']\!]^I \\
[\![\mathrm{d}]\!]^I &= \{(a,l) \mid \mathrm{d}\,(a,l) \in I\} & [\![E \cdot E']\!]^I &= [\![E]\!]^I \circ [\![E']\!]^I \\
[\![E^*]\!]^I &= \{(a,a) \mid a \in V(I)\} \cup [\![E]\!]^I \cup [\![E \cdot E]\!]^I \cup \dots
\end{aligned}$$

Table 3.2: Binary part of definition of the evaluation function $[\![\cdot]\!]^I$ [3, p. 14] [4, p. 1572] with I an assignment, $a, b \in \mathbb{I}$, $l \in \mathbb{L}$, $\mathrm{p} \in \mathbb{O}$, $\mathrm{d} \in \mathbb{D}$, $s \in \mathbb{S}$, E, E' object path expressions.

Example 4. Let $G = \{\text{likes}\,(a,b)\,, \text{likes}\,(b,c)\,, \text{loves}\,(a,c)\}$. Then by definition

$$\begin{aligned}
[\![\text{likes} \cup \text{loves}^-]\!]^G &= \{(a,b),(b,c)\} \cup \{(c,a)\} \\
[\![\text{likes} \cdot \text{loves}]\!]^G &= \{(a,b),(b,c)\} \circ \{(a,c)\} = \{\} \\
[\![\text{likes} \cdot \text{likes}]\!]^G &= \{(a,b),(b,c)\} \circ \{(a,b),(b,c)\} = \{(a,c)\} \\
[\![[\![\text{likes} \cdot \text{likes}]\!]^G \cdot \text{likes}]\!]^G &= \{(a,c)\} \circ \{(a,b),(b,c)\} = \{\} \\
[\![\text{likes}^*]\!]^G &= \{(a,a),(b,b),(c,c)\} \cup \{(a,b),(b,c)\} \cup \\
&\quad\ \{(a,c)\} \cup \{\} \cup \dots
\end{aligned}$$

We continue with an observation that will be useful later on. Let $a \in G$ be an individual, s a shape name and $I = G \cup L$ an assignment. It holds $a \in [\![s]\!]^I$ if and only if $s(a) \in L$ for any s. This is because G does not contain any shape names s or shape atoms $s(a)$ by definition, whereas L contains only shape atoms $s(a)$. Consider Example 5 for an illustration.

Example 5. At some points, we can replace $I = G \cup L$ with just L or G. Let $\mathit{AdultShape} \leftrightarrow \mathit{ReasonableShape} \wedge \text{Adult}$. Then $[\![\mathit{AdultShape}]\!]^I = \{a \mid \mathit{AdultShape}(a) \in I\} = \{a \mid \mathit{AdultShape}(a) \in L\}$. Furthermore,

$$\begin{aligned}
[\![\mathit{ReasonableShape} \wedge \text{Adult}]\!]^I &= [\![\mathit{ReasonableShape}]\!]^I \cap [\![\text{Adult}]\!]^I \\
&= \{a \mid \mathit{ReasonableShape}(a) \in I\} \cap \{a \mid \text{Adult}\,(a) \in I\} \\
&= \{a \mid \mathit{ReasonableShape}(a) \in L\} \cap \{a \mid \text{Adult}\,(a) \in G\}.
\end{aligned}$$

3.3 Examples

3.3.1 Properties

It is possible to do various property related operations using existing SHACL formalizations. Consider the data graph

$$\left\{\begin{matrix} \text{fliesIn}\,(\mathit{bird}, \mathit{sky})\,, & \text{swimsIn}\,(\mathit{fish}, \mathit{sea})\,, & \text{climbsIn}\,(\mathit{ape}, \mathit{tree}) \\ \text{movesIn}\,(\mathit{bird}, \mathit{sky})\,, & \text{movesIn}\,(\mathit{fish}, \mathit{sea})\,, & \text{movesIn}\,(\mathit{ape}, \mathit{tree}) \end{matrix}\right\}$$

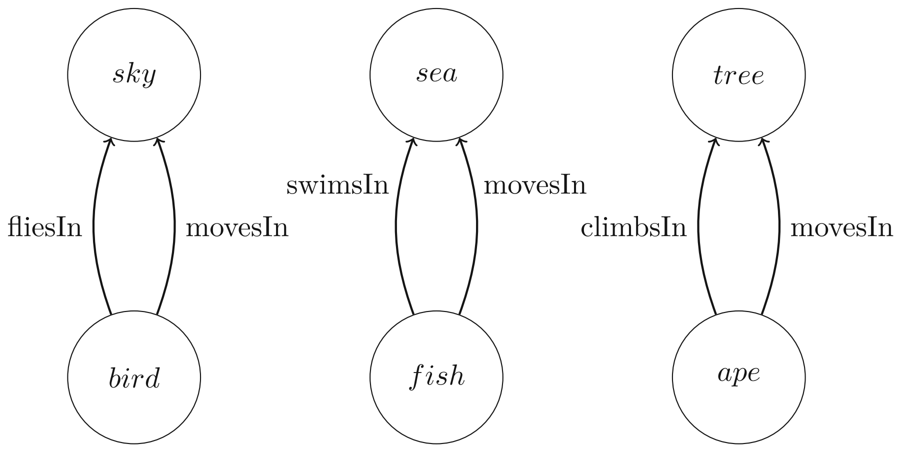


Figure 3.1: Moving animals demonstrating property evaluation

which is also displayed in Figure 3.1. The shapes in Equations 3.1 up to 3.6 are evaluated using this data graph. Two examples that have already been possible to express and evaluate are Equations 3.1 and 3.2.

$$[\![\exists \text{movesIn}.\top \wedge \neg\exists \text{fliesIn}.\top]\!]^G = \{fish, ape\} \tag{3.1}$$

$$[\![\exists \text{movesIn}^-.\top \wedge \neg\exists \text{fliesIn}^-.\top]\!]^G = \{sea, tree\} \tag{3.2}$$

As introduction, we evaluate a part of them with the semantics from existing literature [3]: $\neg\exists \text{fliesIn}^-.\top$ is evaluated as $[\![\text{fliesIn}]\!]^G = \{(bird, sky)\}$ then $[\![\text{fliesIn}^-]\!]^G = \{(sky, bird)\}$ then $[\![\exists \text{fliesIn}^-.\top]\!]^G = \{sky\}$ and finally $[\![\neg\exists \text{fliesIn}^-.\top]\!]^G = \{sea, tree, bird, fish, ape\}$.

$$[\![\exists\,(\text{movesIn} \cap \text{fliesIn}).\top]\!]^G = \{bird\} \tag{3.3}$$

$$[\![\exists\,(\text{movesIn} \cap \text{fliesIn})^-.\top]\!]^G = \{sky\} \tag{3.4}$$

$$[\![\exists\,(\text{movesIn} \setminus \text{fliesIn}).\top]\!]^G = \{fish, ape\} \tag{3.5}$$

$$[\![\exists\,(\text{movesIn} \setminus \text{fliesIn})^-.\top]\!]^G = \{sea, tree\} \tag{3.6}$$

Then for the shape in Equation 3.3 we get $[\![\text{movesIn}]\!]^G = \{(bird, sky), (fish, sea), (ape, tree)\}$ and $[\![\text{movesIn} \cap \text{fliesIn}]\!]^G = \{(bird, sky)\}$ and finally $[\![\exists\,(\text{movesIn} \cap \text{fliesIn}).\top]\!]^G = \{bird\}$. For the shape in Equation 3.5 we get $[\![\exists\,(\text{movesIn} \setminus \text{fliesIn}).\top]\!]^G = \{fish, ape\}$. Then using the inversion operator gives situations like in Equations 3.4 and 3.6. Evaluating shapes of the form $E \setminus E'$ and $E \cap E'$ like in Equations 3.3 to 3.6 is a new contribution.

3.3.2 Literals and predicates

Consider the data graph in Listing 3.1 and the shapes graph in Listing 3.2. In this example, we see that numbers are used and that comparisons with the property maxInclusive happen. Motivated by the fact that the SHACL w3C recommendation allows literals and data types [26], we also want to treat numbers as a more concrete thing than an IRI.

Listing 3.1: Data graph with literals

```
<https://example.com/emily> a <http://schema.org/Person> ;
        <http://schema.org/salary> "3000"^^xsd:integer ;
        <http://schema.org/salary> "4000"^^xsd:integer ;
        <http://schema.org/age> "55"^^xsd:integer ;
        <http://schema.org/name> "Emily"^^xsd:string.
```

Listing 3.2: Shapes graph about salary and age

```
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix schema: <http://schema.org/> .

schema:Person
  a rdfs:Class , sh:NodeShape ;
  sh:property
    [
      sh:path schema:salary ;
      sh:minCount 2 ;
      sh:maxInclusive 3500 ;
    ],
    [
      sh:path schema:age ;
      sh:minCount 1 ;
    ] ;
.
```

We propose a few data types that are natural in this context:

Integers With this data type, it is possible to express equality and inequality between numbers. Therefore, we need the predicates $<$ and $=$.

Strings The second data type is about strings and inspired by the String-based constraint components in the SHACL recommendation [26]. With the „maxlength“ (minlength) predicate of this data type, it is possible to express if some integer is an upper (lower) limit for the string length. The „pattern“ predicate shows if a literal matches a regular expression.

The evaluation $[\![\circledast(\mathrm{d},\psi)]\!]^I$ is done in two parts. The first case is with $\psi \in \mathbb{L}$ a literal. The intention of this is to enable comparison of data with a literal value that is chosen in the shape. The evaluation in this case is $\{a \mid \forall l \in \mathbb{L}((a,l) \in [\![\mathrm{d}]\!]^I \implies \circledast(l,\psi))\}$. Such evaluations are demonstrated in Examples 6, 7, 8, 9, and 10.

Example 6. With the new syntax, we can find all persons with age higher than 18. Let $G = \{\mathrm{Person}\,(robert)\,, \mathrm{age}\,(robert, 19)\,, \mathrm{Person}\,(mia)\,, \mathrm{age}\,(connor, 17)\,, \mathrm{age}\,(connor, 20)\}$.

We have $[\![(\text{age} > 18)]\!]^I = \left\{c \mid \forall l \in \mathbb{L}\left((c,l) \in [\![\text{age}]\!]^I \implies (l > 18)\right)\right\} = \{robert, mia\}$ where $mia \in [\![(\text{age} > 18)]\!]^I$ because of vacuous truth.

We demonstrate how this can be used within a complete evaluation process. Let (C,T) be a shape document with $C = \{\mathit{AdultShape} \leftrightarrow \text{Person} \wedge (\text{age} > 18)\}$ and $T = \{\mathit{AdultShape}(robert)\}$. Let $I = G \cup L$ where $L = \{\mathit{AdultShape}(robert), \mathit{AdultShape}(mia)\}$ be an assignment. Then $[\![\mathit{AdultShape}]\!]^I = \{robert, mia\}$ and $[\![\text{Person} \wedge (\text{age} > 18)]\!]^I = [\![\text{Person}]\!]^I \cap [\![\text{age} > 18]\!]^I = \{robert, mia\} \cap \{robert, mia\}$. It follows that G validates (C,T) because I is an assignment for G such that I is a model of C and $T \subseteq L$.

Example 7. Let $G = \{\text{ssn}\,(lily, 123)\,, \text{salary}\,(lily, 5000)\,, \text{ssn}\,(james, 456)\}$ be a data graph. Then the set $[\![\text{salary}]\!]^I = \{(lily, 5000)\}$ has a single pair as element. The statement $(lily, 5000) \in [\![\text{salary}]\!]^I \wedge (5000 > 3500)$ is true. For all other $l' \in \mathbb{L}$ holds $(lily, l') \notin [\![\text{salary}]\!]^I$. Thus $lily \in [\![(\text{salary} > 3500)]\!]^I$. For all other individuals $a \in V(G) \setminus \{lily\}$ and literals $l \in \mathbb{L}$ holds that $(a,l) \notin [\![\text{salary}]\!]^I$. Thus, $a \in [\![(\text{salary} > 3500)]\!]^I$ for all those, too. Consequently, $[\![(\text{salary} > 3500)]\!]^I = V(G)$.

Example 8. We want to express that each individual salary of a person is at most 4500. Let

$$G_2 = \begin{Bmatrix} \text{salary}\,(lauren, 3000)\,, \text{salary}\,(lauren, 4000)\,, \\ \text{salary}\,(ethan, 2000)\,, \text{salary}\,(ethan, 5000)\,, \text{salary}\,(lily, 3500) \end{Bmatrix}$$

We evaluate $[\![\text{salary} \leq 4500]\!]^I = \{lauren, lily\}$. Note that we cannot express something like „there are at least two salaries“ with this property.

We can also consider some SHACL string-based constraint components. Example 9 is inspired by the sh:minLength property from the W3C recommendation [26].

Example 9. We want to express that a password has length at least 12 [26]. Let

$$G = \{\text{password}\,(user1, length7)\,, \text{password}\,(user2, length13abcde)\}.$$

Then, by definition $[\![\text{minlength}\,(\mathit{password}, \mathit{12})]\!]^I = \{c \mid \forall l \in \mathbb{L}((c,l) \in [\![\text{password}]\!]^I \implies \text{minlength}\,(l, \mathit{12}))\} = \{user2\}$.

Let (C,T) be a shape document with $C = \{\mathit{StrongShape} \leftrightarrow \text{minlength}(\text{password}, 12)\}$ and $T = \{\mathit{StrongShape}(user2)\}$. Let $I = G \cup L$ where $L = \{\mathit{StrongShape}(user2)\}$ is an assignment. Then $[\![\mathit{StrongShape}]\!]^I = \{user2\}$ and $[\![\text{minlength}(\text{password}, 12)]\!]^I = \{user2\}$. It follows that G validates (C,T) because I is an assignment for G such that I is a model of C and $T \subseteq L$.

We can consider an idea that is similar to SHACL property pair constraint components. Example 10 is inspired by sh:equals property from the W3C recommendation [26].

Example 10. We want to express that first name and given name are the same [27]. Let

$$G = \begin{Bmatrix} \text{firstName}\,(person1, alice)\,, \text{givenName}\,(person1, alice)\,, \\ \text{firstName}\,(person2, bob)\,, \text{givenName}\,(person2, robert) \end{Bmatrix}$$

and let „eq“ be a binary predicate. Then $[\![\mathrm{eq}\,(\mathit{firstname}, \mathit{givenname})]\!]^I = \{c \mid \forall l, l' \in \mathbb{L}((c,l) \in [\![\mathrm{firstName}]\!]^I \wedge (c,l') \in [\![\mathrm{givenName}]\!]^I \implies \mathrm{eq}\,(l,l'))\} = \{\mathit{person}1\}$ as it does not hold that $\mathrm{eq}\,(\mathit{bob}, \mathit{robert})$

The second case is with $\psi \in \mathbb{D}$ a data property. The purpose of this is to enable comparison of one value with a value that is given in a different part of the data (and not the shapes). The evaluation for this case is $\Big\{a \mid \forall l, l' \in \mathbb{L}\Big((a,l) \in [\![\mathrm{d}]\!]^I \wedge (a,l') \in [\![\psi]\!]^I \implies \circledast\,(l,l')\Big)\Big\}$. Such a case is demonstrated in Example 11.

Example 11. We can make use of a second data property instead of a literal. We want to express that the salary is smaller than some maximum salary. Let

$$G = \left\{\begin{matrix} \mathrm{salary}\,(\mathit{lauren}, 3000)\,, \mathrm{salary}\,(\mathit{lauren}, 4000)\,, \\ \mathrm{maxSalary}\,(\mathit{lauren}, 5000)\,, \mathrm{maxSalary}\,(\mathit{lauren}, 6000)\,, \end{matrix}\right\}$$

then $[\![(\mathrm{salary} \leq \mathrm{maxSalary})]\!]^I = \{\mathit{lauren}\}$ because we can use the semantics like this

$$\forall l, l' \in \mathbb{L}\Big((\mathit{lauren}, l) \in [\![\mathrm{salary}]\!]^I \wedge (\mathit{lauren}, l') \in [\![\mathrm{maxSalary}]\!]^I \implies (l \leq l')\Big)$$

3.4 Complexity

The combination of conjunction, negation and existential quantification makes regular SHACL validation NP-complete in the size of the graph and constraints. Forbidding negation would be sufficient to regain tractability [15].

[15] allows full SPARQL property paths as defined in the SPARQL recommendation [22]. SPARQL property paths contain negation as well as concatenation, and the Kleene star [22]. This makes no difference for the complexity results, as lower bound proofs can be done without using property paths [14].

From this, we see that our extension does not change the definitions that determine the complexity in [15]. Thus, SHACL validation is NP-complete for our formalization as well.

What if we stepped away from a fixed graph? This corresponds to the SHACL satisfiability problem from [30], which is also called static verification problem [12]. Concatenation and Kleene star are not allowed in actions of the forms $\alpha = \mathrm{p} \oplus \mathrm{r}$ and $\alpha = \mathrm{p} \ominus \mathrm{r}$. Therefore, when only these actions are used, satisfiability remains decidable, but not tractable [12]. However, we also have actions of the forms $\mathrm{A} \oplus \phi_c$ and $\mathrm{A} \ominus \phi_c$ with ϕ_c concepts. Both $\geq_n E.\phi$ and $\mathrm{EQ}\,(E, E')$ with E path expressions are allowed in concepts. As path expressions have concatenation and Kleene star, it follows that SHACL satisfiability is undecidable for such actions [12].

CHAPTER 4

Update Language and Static Validation

4.1 Update Language

The following update language expresses changes in data graphs. We may call these updates *actions*, to distinguish them from other updates or rather modifications that we will introduce in Section 4.2. The syntax idea is after [2] and [11].

Definition 1 ([2])**.** *Basic actions* β and *(complex) actions* α are defined by the following grammar:

$$\beta ::= (\mathrm{A} \oplus \phi_c) \mid (\mathrm{A} \ominus \phi_c) \mid (\mathrm{p} \oplus \mathrm{r}) \mid (\mathrm{p} \ominus \mathrm{r})$$
$$\alpha ::= \epsilon \mid (\beta \cdot \alpha)$$

where $\mathrm{A} \in \mathbb{C}, \mathrm{p} \in \mathbb{O}$, ϕ_c are non-ground concepts, r are non-ground basic object properties, and ϵ denotes the *empty* action. ϕ_c and r may take individuals only from $V(I)$. Function composition denotes successive action execution.

Example 12. Basic actions allow expressing updates like, for example, $\alpha_1 = \mathrm{Human} \oplus \mathrm{Student}$, $\alpha_2 = \mathrm{Adult} \ominus (\mathrm{age} < 18)$, $\alpha_3 = \mathrm{studiesFor} \oplus \mathrm{takesExam}$, $\alpha_4 = \mathrm{poor} \oplus (\mathrm{salary} < \mathrm{minimumSalary})$, $\alpha_5 = \mathrm{enrolledIn} \oplus (x, y)$, $\alpha_6 = \mathrm{Student} \oplus x$. The following action is a complex action: $\alpha_7 = \mathrm{enrolledIn} \oplus (x, y) \cdot \mathrm{enrolledIn} \ominus (x, 1375)$.

The semantics idea is similar to [2] and [11]. The first step to capture the effects of actions is grounding the actions.

Definition 2. A *substitution* σ is a function from $\mathbb{V}$ to $\mathbb{I}$. For an action α we denote with $\sigma(\alpha)$ the result of replacing every occurrence of variable x by the individual $\sigma(x)$. An action is *ground* if it has no variables.

We can see an example for a substitution in Example 19. Suppose in the following that a substitution σ is given for all $t, t_1, t_2 \in \mathbb{V}$. Then concepts ϕ_c and basic object properties r are evaluated using Table 3.1 and 3.2.

Definition 3. Let G be a data graph, α be a ground (complex) action and S_α a mapping from graphs to graphs. The result of applying α or rather the mapping S_α to G is the data graph G^α such that

$$\begin{aligned}
G^{\epsilon} &= S_{\epsilon}(G) = G \\
G^{(\mathrm{A}\oplus\phi_c)\cdot\alpha} &= S_{(\mathrm{A}\oplus\phi_c)\cdot\alpha}(G) = S_\alpha\left(G \cup \left\{\mathrm{A}\left(v\right) \mid v \in [\![\phi_c]\!]^G\right\}\right) \\
G^{(\mathrm{A}\ominus\phi_c)\cdot\alpha} &= S_{(\mathrm{A}\ominus\phi_c)\cdot\alpha}(G) = S_\alpha\left(G \setminus \left\{\mathrm{A}\left(v\right) \mid v \in [\![\phi_c]\!]^G\right\}\right) \\
G^{(\mathrm{p}\oplus\mathrm{r})\cdot\alpha} &= S_{(\mathrm{p}\oplus\mathrm{r})\cdot\alpha}(G) = S_\alpha\left(G \cup \left\{\mathrm{p}\left(a,b\right) \mid (a,b) \in [\![\mathrm{r}]\!]^G\right\}\right) \\
G^{(\mathrm{p}\ominus\mathrm{r})\cdot\alpha} &= S_{(\mathrm{p}\ominus\mathrm{r})\cdot\alpha}(G) = S_\alpha\left(G \setminus \left\{\mathrm{p}\left(a,b\right) \mid (a,b) \in [\![\mathrm{r}]\!]^G\right\}\right)
\end{aligned}$$

We may write I for the assignment $I = G \cup L$ where G is the original data graph. Furthermore, we may write I^α for the assignment $I^\alpha = G^\alpha \cup L$ where G^α is the result of applying the mapping S_α to G for action α. It is important to observe that L is equal in I and I^α.

Example 13. With $G = \{\mathrm{ssn}_2\left(a, 2341\right), \mathrm{ssn}_3\left(a, 2341\right)\}$ and $\alpha = \mathrm{ssn}_1 \oplus \mathrm{ssn}_2 \cap \mathrm{ssn}_3$ we get that $G^\alpha = G \cup \{\mathrm{ssn}_1\left(a, 2341\right)\}$ because $[\![\mathrm{ssn}_2 \cap \mathrm{ssn}_3]\!]^I = \{(a, 2341)\} \cap \{(a, 2341)\}$

4.2 Static validation

The update language from Section 4.1 with actions α formalizes changes on data graphs. In contrast, the following definitions capture the effects of those updates by modifying the constraints. The distinction of updates on the data graphs and updates on the constraints will be of particular importance for the transformation implementation in Chapter 6.

We need a transformation $\mathrm{TR}_\alpha^\leftrightarrow$ that takes a set of constraints C and actions α and rewrites them into a new set of constraints C^α, which we denote as $\mathrm{TR}_\alpha^\leftrightarrow(C) = C^\alpha$. For this transformation, it must hold that

$$G^\alpha \text{ validates } (C, T) \iff G \text{ is valid on } (C^\alpha, T)$$

Definition 4 ([2])**.** Given a constraint $s \leftrightarrow \phi$ we write $s \leftrightarrow \phi_{\phi_1 \leftarrow \phi_2}$ to denote the constraint where each occurrence of ϕ_1 is replaced by ϕ_2. Let α be a ground (complex) action. Suppose that α and ϕ take individuals only from $V(I)$. We define a mapping $\mathrm{TR}_\alpha^\leftrightarrow$ from constraints to constraints

$$\mathrm{TR}_\alpha^\leftrightarrow(s \leftrightarrow \phi) = s \leftrightarrow \mathrm{TR}_\alpha(\phi)$$

and the main mapping TR_α from shape expressions to shape expressions

$$\begin{aligned}
\mathrm{TR}_\epsilon(\phi) &= \phi\\
\mathrm{TR}_{(\mathrm{A}\oplus\phi_c)\cdot\alpha}(\phi) &= (\mathrm{TR}_\alpha(\phi))_{\mathrm{A}\leftarrow\mathrm{A}\vee\phi_c}\\
\mathrm{TR}_{(\mathrm{A}\ominus\phi_c)\cdot\alpha}(\phi) &= (\mathrm{TR}_\alpha(\phi))_{\mathrm{A}\leftarrow\mathrm{A}\wedge\neg\phi_c}\\
\mathrm{TR}_{(\mathrm{p}\oplus\mathrm{r})\cdot\alpha}(\phi) &= (\mathrm{TR}_\alpha(\phi))_{\mathrm{p}\leftarrow\mathrm{p}\cup\mathrm{r}}\\
\mathrm{TR}_{(\mathrm{p}\ominus\mathrm{r})\cdot\alpha}(\phi) &= (\mathrm{TR}_\alpha(\phi))_{\mathrm{p}\leftarrow\mathrm{p}\setminus\mathrm{r}}
\end{aligned}$$

where a is a constant, and we use $A \vee \phi_c$ for $\neg(\neg A \wedge \neg\phi_c)$.

Given a set C of constraints, we write $\mathrm{TR}^{\leftrightarrow}_\alpha(C)$ to denote the set of constraints where $\mathrm{TR}^{\leftrightarrow}_\alpha$ is applied to each constraint $s \leftrightarrow \phi \in C$. Similarly, we write $C_{\phi_1\leftarrow\phi_2}$ to denote the set of constraints where each occurrence of ϕ_1 is replaced by ϕ_2 in each constraint. Furthermore, we write ϕ_α to denote $\mathrm{TR}_\alpha(\phi)$. As shorthand, we will write E_α to denote the same for path expressions with $\mathrm{TR}_\alpha(E)$ defined analogously to $\mathrm{TR}_\alpha(\phi)$.

Example 14 is a basic demonstration of capturing the effects of actions.

Example 14. Suppose we have

$$\begin{aligned}
G &= \{\mathrm{Student}(ann)\} & T &= \{Person(ann)\}\\
\alpha &= \mathrm{Human} \oplus \mathrm{Student} & C &= \{Person \leftrightarrow \mathrm{Human}\}
\end{aligned}$$

Then

$$\begin{aligned}
G^\alpha &= G \cup \left\{\mathrm{Human}(v) \mid v \in [\![\mathrm{Student}]\!]^G\right\} = \{\mathrm{Student}(ann), \mathrm{Human}(ann)\}\\
C^\alpha &= \{Person \leftrightarrow \mathrm{Human} \vee \mathrm{Student}\}
\end{aligned}$$

It holds that G validates (C^α, T) if and only if G^α validates (C, T).

In Example 15, we can quickly see that the role of the left and right part of actions significantly differs.

Example 15. Suppose we have

$$G = \{\mathrm{Professor}(Olivia), \mathrm{Postdoc}(Liam), \mathrm{Employee}(Olivia), \mathrm{Employee}(Liam)\}$$

Then

$$\begin{aligned}
G^{\mathrm{Postdoc}\ominus\mathrm{Employee}} &= G \setminus \left\{\mathrm{Postdoc}(v) \mid v \in [\![\mathrm{Employee}]\!]^G\right\}\\
&= \{\mathrm{Professor}(Olivia), \mathrm{Employee}(Olivia), \mathrm{Employee}(Liam)\}\\
G^{\mathrm{Employee}\ominus\mathrm{Postdoc}} &= G \setminus \left\{\mathrm{Employee}(v) \mid v \in [\![\mathrm{Postdoc}]\!]^G\right\}\\
&= \{\mathrm{Professor}(Olivia), \mathrm{Employee}(Olivia), \mathrm{Postdoc}(Liam)\}
\end{aligned}$$

After having seen the relevance of the two parts of actions in Example 15, we consider the impact of the order within a sequence of actions in Example 16. This new example is an extension of Example 14. It is crucial to notice that the transformation or rather the substitutions are in reversed order with respect to the actions.

Example 16. Suppose that we have

$$G = \{\mathrm{Student}\,(charlie)\,, \mathrm{Human}\,(george)\} \qquad T = \{s(charlie), s(george)\}$$
$$\alpha = \mathrm{Human} \oplus \mathrm{Student} \cdot \mathrm{Mammal} \oplus \mathrm{Human} \qquad C = \{s \leftrightarrow \mathrm{Human} \wedge \mathrm{Mammal}\}$$

We abbreviate Human. . . H, Mammal. . . M, Student. . . S, *charlie*. . . *c*, *george*. . . *g*. Then

$$\begin{aligned} G^\alpha &= S_{(\mathrm{H}\oplus\mathrm{S})\cdot\mathrm{M}\oplus\mathrm{H}} = S_{\mathrm{M}\oplus\mathrm{H}}(G \cup \{\mathrm{H}(v) \mid v \in [\![\mathrm{S}]\!]^G) = \{\mathrm{S}\,(c)\,, \mathrm{H}\,(c)\,, \mathrm{M}\,(c)\,, \mathrm{H}\,(g)\,, \mathrm{M}\,(g)\} \\ C^\alpha &= \{\mathrm{TR}_{(\mathrm{H}\oplus\mathrm{S})\cdot\mathrm{M}\oplus\mathrm{H}}(\mathrm{H}\wedge\mathrm{M})\} = \{(\mathrm{TR}_{\mathrm{M}\oplus\mathrm{H}}(\mathrm{H}\wedge\mathrm{M}))_{\mathrm{H}\leftarrow\mathrm{H}\vee\mathrm{S}}\} \\ &= \{((\mathrm{H}\wedge\mathrm{M})_{\mathrm{M}\leftarrow\mathrm{M}\vee\mathrm{H}})_{\mathrm{H}\leftarrow\mathrm{H}\vee\mathrm{S}}\} = \{\mathrm{H}\vee\mathrm{S}\} \end{aligned}$$

Then G validates (C^α, T) and also G^α validates (C, T).

In contrast, if we had defined that the substitution were in the same order as the actions we would get $((\mathrm{H}\wedge\mathrm{M})_{\mathrm{H}\leftarrow\mathrm{H}\vee\mathrm{S}})_{\mathrm{M}\leftarrow\mathrm{M}\vee\mathrm{H}} = \mathrm{H}\vee(\mathrm{M}\wedge\mathrm{S})$ and then G would not validate (C^α, T) while G^α would still validate (C, T).

Example 17. This example demonstrates that the transformation requires the inverse to be defined not only on object property names, but also on basic object properties. Let $G = \{\mathrm{q}_1\,(a,b)\,, \mathrm{q}_2\,(b,a)\}$. Then the action $\alpha = \mathrm{p} \oplus (\mathrm{q}_1^- \cap \mathrm{q}_2)$ gives $G^\alpha = G \cup \{\mathrm{p}\,(b,a)\}$ as $[\![\mathrm{q}_1^- \cap \mathrm{q}_2]\!]^G = \{(b,a)\} \cap \{(b,a)\}$. Let $\phi = \exists \mathrm{p}^-.\top$ be a shape expression. Directly applying the transformation gives $\mathrm{TR}_\alpha(\phi) = \exists(\mathrm{p} \cup (\mathrm{q}_1^- \cap \mathrm{q}_2))^-.\top$.

Example 18 shows how properties in an action work. It is an extension of Figure 1.1.

Example 18. Let

$$G = \{\mathrm{q}\,(a,b)\,, \mathrm{p}\,(b,a)\,, \mathrm{q}\,(b,c)\} \qquad T = \{s(a), s(b), s(c)\}$$
$$\alpha = \mathrm{p} \oplus \mathrm{q} \qquad C = \{s \leftrightarrow \geq_1 \mathrm{q}.\top\}$$
$$G^\alpha = G \cup \{\mathrm{p}\,(a,b)\,, \mathrm{p}\,(b,c)\}$$

G is displayed in Figure 4.1 and G^α in Figure 4.2. To find C^α we transform

$$C^\alpha = \{\mathrm{TR}_\alpha^\leftrightarrow\,(s \leftrightarrow \geq_1 \mathrm{q}.\top)\} = \{s \leftrightarrow \mathrm{TR}_\alpha\,(\geq_1 \mathrm{q}.\top)\} = \{s \leftrightarrow \geq_1 (\mathrm{q} \cup \mathrm{p})\,.\top\}$$

Then it holds that G does not validate (C^α, T) because of c, but also G^α does not validate (C, T) because of c.

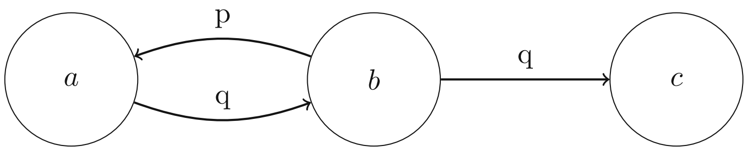


Figure 4.1: Original graph G for Example 18

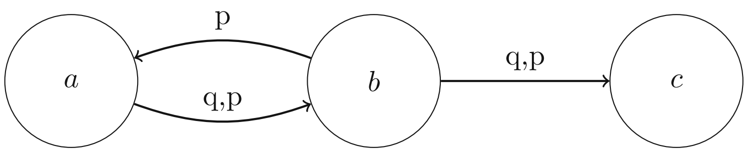
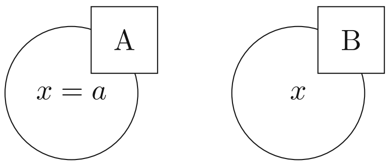
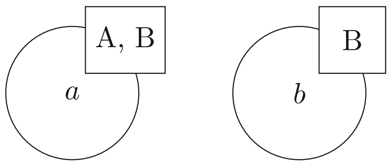


Figure 4.2: Updated graph G^α for Example 18



(a) Original graph G



(b) Updated graph G^α

Figure 4.3: Graphs for Example 19

4.3 Examples

In this section, we demonstrate the static validation technique with several examples.

Example 19 shows how a variable can be used.

Example 19. The unary atom $\mathrm{B}(b)$ is already in G and choosing $\sigma(x) = b$ would attempt to add it another time. Thus, choose $\sigma(x) = a$.

$$\begin{aligned} G &= \{\mathrm{A}(a), \mathrm{B}(b)\} & T &= \{s(a)\} \\ \alpha &= \mathrm{B} \oplus x & C &= \{s \leftrightarrow \mathrm{A} \wedge \mathrm{B}\} \\ G^\alpha &= G \cup \{\mathrm{B}(a)\} \end{aligned}$$

G is displayed in Figure 4.3a and G^α in Figure 4.3b. Then

$$C^\alpha = \{\mathrm{TR}^{\leftrightarrow}_{\mathrm{B}\oplus a}(s \leftrightarrow \mathrm{A} \wedge \mathrm{B})\} = \{s \leftrightarrow \mathrm{TR}_{\mathrm{B}\oplus a}(\mathrm{A} \wedge \mathrm{B})\} = \{s \leftrightarrow \mathrm{A} \wedge (\mathrm{B} \vee a)\}$$

We show that G validates (C^α, T). Consider the assignment $I = G \cup L$ with $L = T$. Then $T \subseteq L$ and $[\![s]\!]^I = \{a\}$. Furthermore, $[\![\mathrm{A} \wedge (\mathrm{B} \vee a)]\!]^I = \{a\} \cap (\{b\} \cup \{a\}) = \{a\}$. Then $I \models C^\alpha$ because $[\![s]\!]^I = [\![\mathrm{A} \wedge (\mathrm{B} \vee a)]\!]^I$. We show that G^α validates (C, T). Consider the assignment $I^\alpha = G^\alpha \cup L^\alpha$ with $L^\alpha = T$. Then $T \subseteq L^\alpha$ and $[\![s]\!]^{I^\alpha} = \{a\}$. Furthermore, $[\![\mathrm{A} \wedge \mathrm{B}]\!]^{I^\alpha} = \{a\} \cap \{a, b\} = \{a\}$. Then $I^\alpha \models C$ because $[\![s]\!]^I = [\![\mathrm{A} \wedge \mathrm{B}]\!]^I$.

Example 20 demonstrates adding a property name to a pair of individuals.

Figure 4.4: Original graph G for Example 20.

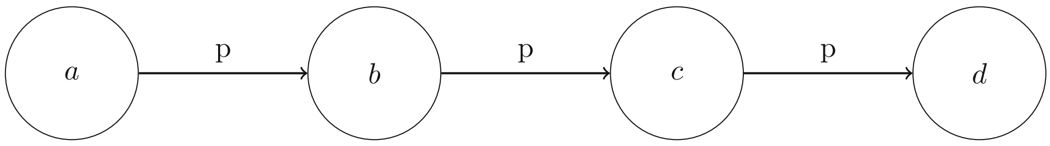


Figure 4.5: Updated graph G^α for Example 20.

Example 20. Let

$$\begin{aligned} G &= \{\mathrm{p}\,(a,b)\,, \mathrm{p}\,(c,d)\} & T &= \{s(a), s(b), s(c)\} \\ \alpha &= \mathrm{p} \oplus (b,c) & C &= \{s \leftrightarrow \geq_1 \mathrm{p}.\top\} \\ G^\alpha &= G \cup \{\mathrm{p}\,(b,c)\} \end{aligned}$$

G is displayed in Figure 4.4 and G^α in Figure 4.5. To find C^α we transform

$$C^\alpha = \{\mathrm{TR}_\alpha^\leftrightarrow (s \leftrightarrow \geq_1 \mathrm{p}.\top)\} = \{s \leftrightarrow \mathrm{TR}_\alpha (\geq_1 \mathrm{p}.\top)\} = \{s \leftrightarrow \geq_1 (\mathrm{p} \cup (b,c)) \,.\top\}$$

G validates (C^α, T) because there is the assignment $I = G \cup L$ with $L = T$ such that $I \models C^\alpha$ because $s(b)$ holds as C^α has $\mathrm{p} \cup (b,c)$ as property. In addition, G^α validates (C,T) because there is the assignment $I = G \cup L$ with $L = T$ such that $I \models C$.

Example 21 demonstrates how shapes of the form $\mathrm{EQ}(E, E')$ are evaluated under actions on constants.

Example 21. Let $\alpha = \mathrm{loves} \oplus ((c,d) \cup (e,f))$, L arbitrary and $I^\alpha = G^\alpha \cup L$.

$$\begin{aligned} G &= \{\mathrm{loves}\,(a,b)\,, \mathrm{likes}\,(a,b)\,, \mathrm{likes}\,(c,d)\,, \mathrm{likes}\,(e,f)\} \\ G^\alpha &= G \cup \{\mathrm{loves}\,(c,d)\,, \mathrm{loves}\,(e,f)\} \\ \phi &= \mathrm{EQ}\,(\mathrm{loves}, \mathrm{likes}) \\ \phi_{\mathrm{p} \leftarrow \mathrm{p} \cup \mathrm{r}} &= \mathrm{EQ}\,(\mathrm{loves} \cup (c,d) \cup (e,f)\,, \mathrm{likes}) \end{aligned}$$

In Example 21, p is loves and $(c,d) \cup (e,f)$ is r. Evaluating on the original assignment I with the original shape gives just $[\![\phi]\!]^I = \{a\}$. However, evaluating the original shape on the updated assignment I^α gives $[\![\phi]\!]^{I^\alpha} = \{a, c, e\}$. Then, evaluating the transformed shape on the original assignment also gives $[\![\phi_{\mathrm{p} \leftarrow \mathrm{p} \cup \mathrm{r}}]\!]^I = \{a, c, e\}$.

We now consider the motivation for the definition of $V(I)$. One common definition for $V(I)$ or rather $V(G)$ is to denote the set of individuals appearing in G [3]. Thus, our definition of $V(I)$ should at least include all those individuals. However, actions can add new individuals to the set of individuals occurring in G, and they can also remove individuals from it. This can be seen in Example 22. Constraints may contain individuals that are not present in the original data graph G in addition.

Example 22. Let $G = \{\mathrm{A}\,(a)\}$, $\alpha_1 = \mathrm{A} \ominus a$, and $\alpha_2 = \mathrm{A} \oplus b$. Then $G^{\alpha_1} = \{\}$ and $G^{\alpha_2} = \{\mathrm{A}\,(a)\,, \mathrm{A}\,(b)\}$. After each action, the set of individuals of the new graph differs from the set of individuals of the original graph.

We want to be able to express constraints that take the updates into account as well. In other words, we do not restrict that every individual that occurs in C or T must also occur in G. [28] notices that the W3C SHACL recommendation [26] does not define if explicitly enumerated target nodes that are missing in the data graph should result in an error. Example 23 demonstrates what we can express without the restriction and the motivation for the definition of $V(I)$.

Example 23. Let $G = \{\mathrm{A}\,(a)\}$, $\alpha = \mathrm{A} \oplus b$, $C = \{s \leftrightarrow \top\}$, and $T = \{s(b)\}$. By the definition of the transformation holds $C^\alpha = C$. By action definition holds $G^\alpha = \{\mathrm{A}\,(a)\,, \mathrm{B}\,(b)\}$.

What if we had defined $V(I)$ to be just the individuals in G? Certainly, $s(b) \in L$ is required for G to validate (C^α, T) because $T \subseteq L$ must hold. However, b cannot be an element of $[\![\top_\alpha]\!]^I = V(I) = \{a\}$ with this definition.

In contrast, if we allow the nodes of the actions and constraints to silently be present in $V(I)$ the validation works as intended. First, G validates (C^α, T) because there is assignment $I = G \cup \{s(a), s(b)\}$ such that $[\![\top_\alpha]\!]^I = V(I) = [\![s]\!]^I = \{a, b\}$. Second, G^α validates (C, T) because there is assignment $J = G^\alpha \cup \{s(a), s(b)\}$ such that $[\![\top]\!]^J = V(J) = [\![s]\!]^J = \{a, b\}$. Thus, with the new definition of $V(I)$, our goal that G^α validates (C, T) if and only if G validates (C^α, T) is fulfilled in this example.

CHAPTER 5

Correctness

Our goal is to prove that G^α validates (C, T) if and only if G validates (C^α, T). To achieve this, we will first show some basic properties that will facilitate the later proofs. Then, Lemma 2 will be an essential result about arbitrary shape expressions under a single update. In Theorem 1 we will step from basic actions β that perform only one update to complex actions α that allow a sequence of updates. Finally, Theorem 2 will be the final result.

We start with an observation about basic object properties that we will use as base case for Lemma 1. Roughly put, the observation is that our goal that G^α validates (C, T) if and only if G validates (C^α, T) holds for basic object properties under one specific action on properties. Recall that we said that we may write I^α for the assignment $I^\alpha = G^\alpha \cup L$ where G^α is the result of applying the mapping S_α to G for action α in Section 4.1.

Claim 1. *Let* r' *be a basic object property,* $\beta = \mathrm{p} \oplus \mathrm{r}$ *a basic action,* I *an assignment. Then* $[\![\mathrm{r}']\!]^{I^\beta} = [\![\mathrm{r}'_{\mathrm{p} \leftarrow \mathrm{p} \cup \mathrm{r}}]\!]^{I}$.

Proof. Proof by induction over the structure of r'.

Base case For the base case we consider r' to be an object property name symbol, the inverse of an object property name symbol, or a pair of individuals.

- Suppose that r' is an object property name symbol. Then either $\mathrm{r}' = \mathrm{p}$ or $\mathrm{r}' \neq \mathrm{p}$.

  Suppose $\mathrm{r}' = \mathrm{p}$. Then $[\![\mathrm{r}']\!]^{I^\beta} = [\![\mathrm{p}]\!]^{I^\beta}$. The evaluation function $[\![\cdot]\!]^{I^\beta}$ maps basic object properties like r' to pairs of individuals. Then by assumption of I^β we can separate $[\![\mathrm{p}]\!]^{I^\beta} = \{(a, b) \mid \mathrm{p}\,(a, b) \in I^\beta\}$ into the union of two sets of pairs of individuals, namely $\{(a, b) \mid \mathrm{p}\,(a, b) \in I^\beta\} = \{(a, b) \mid \mathrm{p}\,(a, b) \in G\} \cup \{(a, b) \mid (a, b) \in [\![\mathrm{r}]\!]^I\}$. By definition $\{(a, b) \mid \mathrm{p}\,(a, b) \in G\} = [\![\mathrm{p}]\!]^G = [\![\mathrm{p}]\!]^I$ and $\{(a, b) \mid$

$(a,b) \in [\![\mathrm{r}]\!]^I\} = [\![\mathrm{r}]\!]^I$. By the semantics in Table 3.2 holds $[\![\mathrm{p}]\!]^I \cup [\![\mathrm{r}]\!]^I = [\![\mathrm{p} \cup \mathrm{r}]\!]^I$. As $\mathrm{p} \cup \mathrm{r} = \mathrm{p}_{\mathrm{p} \leftarrow \mathrm{p} \cup \mathrm{r}} = (\mathrm{r}')_{\mathrm{p} \leftarrow \mathrm{p} \cup \mathrm{r}}$ we get $[\![\mathrm{p} \cup \mathrm{r}]\!]^I = [\![(\mathrm{r}')_{\mathrm{p} \leftarrow \mathrm{p} \cup \mathrm{r}}]\!]^I$ and thus in total $[\![\mathrm{r}']\!]^{I^\beta} = [\![(\mathrm{r}')_{\mathrm{p} \leftarrow \mathrm{p} \cup \mathrm{r}}]\!]^I$ as desired.

If $\mathrm{r}' \neq \mathrm{p}$ then r' is still some basic object property, say $\mathrm{r}' = \mathrm{q}$, because we are in the base case. Then $\mathrm{q}_{\mathrm{p} \leftarrow \mathrm{p} \cup \mathrm{r}} = \mathrm{q}$. Then by the semantics in Table 3.2 $[\![\mathrm{q}_{\mathrm{p} \leftarrow \mathrm{p} \cup \mathrm{r}}]\!]^I = [\![\mathrm{q}]\!]^I$. Thus, it remains to show that $[\![\mathrm{q}]\!]^{I^\beta} = [\![\mathrm{q}]\!]^I$. By semantics of actions holds $G^\beta = G \cup \{\mathrm{p}\,(a,b) \mid (a,b) \in [\![\mathrm{r}]\!]^G\}$. This means that the action only adds binary atoms that have p as predicate. Thus $\{(a,b) \mid \mathrm{q}\,(a,b) \in G\} = \{(a,b) \mid \mathrm{q}\,(a,b) \in G^\beta\}$ and consequently by definition of $[\![\cdot]\!]^I$ and assignments holds $[\![\mathrm{q}]\!]^I = [\![\mathrm{q}]\!]^{I^\beta}$. Then by replacing our shorthand q by r' and our previous result we have again $[\![\mathrm{r}']\!]^{I^\beta} = [\![(\mathrm{r}')_{\mathrm{p} \leftarrow \mathrm{p} \cup \mathrm{r}}]\!]^I$ as desired.

- Suppose that r' is the inverse of an object property name symbol. Then there is some object property name $\mathrm{q} \in \mathbb{O}$ such that $\mathrm{r}' = \mathrm{q}^-$. Then either $\mathrm{q} = \mathrm{p}$ or $\mathrm{q} \neq \mathrm{p}$.

  Suppose $\mathrm{q} = \mathrm{p}$. Then $[\![\mathrm{r}']\!]^{I^\beta} = [\![\mathrm{q}^-]\!]^{I^\beta} = [\![\mathrm{p}^-]\!]^{I^\beta} = \{(a,b) \mid \mathrm{q}\,(b,a) \in I^\beta\}$. The evaluation function $[\![\cdot]\!]^{I^\beta}$ maps basic object properties like r' to pairs of individuals. Then by assumption of I^β we can separate the last set into the union of two sets of pairs of individuals, namely $\{(a,b) \mid \mathrm{q}\,(b,a) \in I^\beta\} = \{(a,b) \mid \mathrm{q}\,(b,a) \in G\} \cup \{(a,b) \mid (b,a) \in [\![\mathrm{r}]\!]^I\}$. By definition of $[\![\cdot]\!]^I$ holds $\{(a,b) \mid \mathrm{q}\,(b,a) \in G\} = [\![\mathrm{p}^-]\!]^G = [\![\mathrm{p}^-]\!]^I$. As basic object properties are trivial cases of object path expressions, it follows that we can evaluate a basic object property r^- like an inverted object property name p^- in Table 3.2. Then $\{(a,b) \mid (b,a) \in [\![\mathrm{r}]\!]^I\} = [\![\mathrm{r}^-]\!]^I$. Then in addition, $[\![\mathrm{p}^-]\!]^I \cup [\![\mathrm{r}^-]\!]^I = [\![\mathrm{p}^- \cup \mathrm{r}^-]\!]^I$. By assumption holds that $\mathrm{r}' = \mathrm{p}^-$ and consequently $[\![(\mathrm{r}')_{\mathrm{p} \leftarrow \mathrm{p} \cup \mathrm{r}}]\!]^I = [\![(\mathrm{p}^-)_{\mathrm{p} \leftarrow \mathrm{p} \cup \mathrm{r}}]\!]^I$. From the definition of $\leftarrow$ follows $(\mathrm{p}^-)_{\mathrm{p} \leftarrow \mathrm{p} \cup \mathrm{r}} = (\mathrm{p}_{\mathrm{p} \leftarrow \mathrm{p} \cup \mathrm{r}})^- = (\mathrm{p} \cup \mathrm{r})^-$ and then $(\mathrm{p} \cup \mathrm{r})^- = \mathrm{p}^- \cup \mathrm{r}^-$. From this follows, that $[\![(\mathrm{r}')_{\mathrm{p} \leftarrow \mathrm{p} \cup \mathrm{r}}]\!]^I = [\![\mathrm{p}^- \cup \mathrm{r}^-]\!]^I$ and then by taking the previous result $[\![\mathrm{r}']\!]^{I^\beta} = [\![\mathrm{p}^- \cup \mathrm{r}^-]\!]^I$ into account, we get $[\![\mathrm{r}']\!]^{I^\beta} = [\![\mathrm{r}'_{\mathrm{p} \leftarrow \mathrm{p} \cup \mathrm{r}}]\!]^I$ as desired.

  Now suppose $\mathrm{q} \neq \mathrm{p}$. Then $\mathrm{q}_{\mathrm{p} \leftarrow \mathrm{p} \cup \mathrm{r}} = \mathrm{q}$ by definition of $\leftarrow$ and then from the semantics in Table 3.2 follows $[\![(\mathrm{q}^-)_{\mathrm{p} \leftarrow \mathrm{p} \cup \mathrm{r}}]\!]^I = [\![\mathrm{q}^-]\!]^I$. From this and the assumption that $\mathrm{r}' = \mathrm{q}^-$ follows $[\![(\mathrm{r}')_{\mathrm{p} \leftarrow \mathrm{p} \cup \mathrm{r}}]\!]^I = [\![\mathrm{q}^-]\!]^I$. Thus, it remains to show that $[\![\mathrm{q}^-]\!]^{I^\beta} = [\![\mathrm{q}^-]\!]^I$. By semantics of actions holds $G^\beta = S_{\mathrm{p} \oplus \mathrm{r}}(G) = G \cup \{\mathrm{p}\,(a,b) \mid (a,b) \in [\![\mathrm{r}]\!]^G\}$. This means that the action only adds binary atoms that have p as predicate. Thus $\{(a,b) \mid \mathrm{q}\,(a,b) \in G\} = \{(a,b) \mid \mathrm{q}\,(a,b) \in G^\beta\}$ and consequently by definition of $[\![\cdot]\!]^I$ and assignments holds $[\![\mathrm{q}^-]\!]^I = [\![\mathrm{q}^-]\!]^{I^\beta}$. Then, by assumption that $\mathrm{r}' = \mathrm{q}^-$ and our previous result, we have again $[\![\mathrm{r}']\!]^{I^\beta} = [\![\mathrm{r}'_{\mathrm{p} \leftarrow \mathrm{p} \cup \mathrm{r}}]\!]^I$ as desired.

- Suppose that r' is a pair of individuals. In other words, there is $c_0, d_0 \in \mathbb{I}$ such that $\mathrm{r}' = (c_0, d_0)$. As $\mathrm{p} \in \mathbb{O}$ is an object property name and (c_0, d_0) a pair of nodes, it follows that $\mathrm{r}' \neq \mathrm{p}$. Then $(c_0, d_0)_{\mathrm{p} \leftarrow \mathrm{p} \cup \mathrm{r}} = (c_0, d_0)$. Then from the semantics of $[\![\cdot]\!]^I$ in Table 3.2 follows $[\![\mathrm{r}'_{\mathrm{p} \leftarrow \mathrm{p} \cup \mathrm{r}}]\!]^I = [\![(c_0, d_0)]\!]^I = \{(c_0, d_0)\}$. Thus, it remains to show that $[\![\mathrm{r}'_{\mathrm{p} \leftarrow \mathrm{p} \cup \mathrm{r}}]\!]^I = [\![(c_0, d_0)]\!]^I = [\![(c_0, d_0)]\!]^{I^\beta}$. The evaluation of pairs of individuals in Table 3.2 is the same for any assignment. Thus, $[\![(c_0, d_0)]\!]^{I^\beta} = [\![(c_0, d_0)]\!]^I$ holds by definition.

Induction hypothesis Let r'_a and r'_b be basic object properties. We assume that $[\![\mathrm{r}'_a]\!]^{I^\beta} = [\![(\mathrm{r}'_a)_{\mathrm{p}\leftarrow\mathrm{p}\cup\mathrm{r}}]\!]^I$ and $[\![\mathrm{r}'_b]\!]^{I^\beta} = [\![(\mathrm{r}'_b)_{\mathrm{p}\leftarrow\mathrm{p}\cup\mathrm{r}}]\!]^I$.

Induction step We need to show that $[\![\mathrm{r}']\!]^{I^\beta} = [\![\mathrm{r}'_{\mathrm{p}\leftarrow\mathrm{p}\cup\mathrm{r}}]\!]^I$. By definition, basic object properties are built using the operators $\cup, \cap$ and $\setminus$. Thus, we need three cases. We, however, show only one of them, as all three cases work exactly the same.

Suppose $\mathrm{r}' = \mathrm{r}'_a \cup \mathrm{r}'_b$. Then, by assumption $[\![\mathrm{r}']\!]^{I^\beta} = [\![\mathrm{r}'_a \cup \mathrm{r}'_b]\!]^{I^\beta}$. Then from semantics of $[\![\cdot]\!]^{I^\beta}$ follows $[\![\mathrm{r}'_a \cup \mathrm{r}'_b]\!]^{I^\beta} = [\![\mathrm{r}'_a]\!]^{I^\beta} \cup [\![\mathrm{r}'_b]\!]^{I^\beta}$. Then by the induction hypotheses $[\![\mathrm{r}'_a]\!]^{I^\beta} \cup [\![\mathrm{r}'_b]\!]^{I^\beta} = [\![(\mathrm{r}'_a)_{\mathrm{p}\leftarrow\mathrm{p}\cup\mathrm{r}}]\!]^I \cup [\![(\mathrm{r}'_b)_{\mathrm{p}\leftarrow\mathrm{p}\cup\mathrm{r}}]\!]^I$. Then from semantics of $[\![\cdot]\!]^I$ follows $[\![(\mathrm{r}'_a)_{\mathrm{p}\leftarrow\mathrm{p}\cup\mathrm{r}}]\!]^I \cup [\![(\mathrm{r}'_b)_{\mathrm{p}\leftarrow\mathrm{p}\cup\mathrm{r}}]\!]^I = [\![(\mathrm{r}'_a)_{\mathrm{p}\leftarrow\mathrm{p}\cup\mathrm{r}} \cup (\mathrm{r}'_b)_{\mathrm{p}\leftarrow\mathrm{p}\cup\mathrm{r}}]\!]^I$. Then from the definition of $\leftarrow$ follows $[\![(\mathrm{r}'_a)_{\mathrm{p}\leftarrow\mathrm{p}\cup\mathrm{r}} \cup (\mathrm{r}'_b)_{\mathrm{p}\leftarrow\mathrm{p}\cup\mathrm{r}}]\!]^I = [\![(\mathrm{r}'_a \cup \mathrm{r}'_b)_{\mathrm{p}\leftarrow\mathrm{p}\cup\mathrm{r}}]\!]^I$. Then from the initial assumption $\mathrm{r}' = \mathrm{r}'_a \cup \mathrm{r}'_b$ follows that $[\![(\mathrm{r}'_a \cup \mathrm{r}'_b)_{\mathrm{p}\leftarrow\mathrm{p}\cup\mathrm{r}}]\!]^I = [\![\mathrm{r}'_{\mathrm{p}\leftarrow\mathrm{p}\cup\mathrm{r}}]\!]^I$. Thus, in summary, we get $[\![\mathrm{r}']\!]^{I^\beta} = [\![\mathrm{r}'_{\mathrm{p}\leftarrow\mathrm{p}\cup\mathrm{r}}]\!]^I$. □

The argument for Claim 1 works the same for actions of the form $\beta = \mathrm{p} \ominus \mathrm{r}$.

The following Lemma 1 is like Claim 1, but with a whole path expression instead of just a basic object property. Recall that in Section 4.2 we said that we may write E_α to denote $\mathrm{TR}_\alpha(E)$ with $\mathrm{TR}_\alpha(E)$ defined analogously to $\mathrm{TR}_\alpha(\phi)$ with ϕ a shape expression.

Lemma 1. *Let E be a path expression, $\beta = \mathrm{p} \oplus \mathrm{r}$ a basic action, and I an assignment. Then $[\![E]\!]^{I^\beta} = [\![E_\beta]\!]^I$.*

Proof. We prove by strong induction on the structural complexity of E.

Base Case In the base case, E is a basic object property. Then we can apply Claim 1 as β is of the form $\mathrm{p} \oplus \mathrm{r}$.

Induction hypothesis Suppose that $[\![E_i]\!]^{I^\beta} = [\![E_{i_\beta}]\!]^I$ with $i \in 0, 1, 2, \ldots n$ where each E_i is of structural complexity i.

Induction step We show that for E with structural complexity $n+1$ built from E_i, E_j with structural complexity at most n using the operators $*, \cdot, \cup$ holds $[\![E]\!]^{I^\beta} = [\![E_\beta]\!]^I$.

Suppose that $E = E_i \cdot E_j$. Then $[\![E]\!]^{I^\beta} = [\![E_i \cdot E_j]\!]^{I^\beta} = [\![E_i]\!]^{I^\beta} \circ [\![E_j]\!]^{I^\beta}$ by definition of $[\![\cdot]\!]^{I^\beta}$. By assumption, the complexities of $[\![E_i]\!]^{I^\beta}$ and $[\![E_j]\!]^{I^\beta}$ are each at most n and thus by the induction hypothesis $[\![E_i]\!]^{I^\beta} \circ [\![E_j]\!]^{I^\beta} = [\![E_{i_\beta}]\!]^I \circ [\![E_{j_\beta}]\!]^I$. Then follows from the definition of $[\![\cdot]\!]^{I^\beta}$ that $[\![E_{i_\beta}]\!]^I \circ [\![E_{j_\beta}]\!]^I = [\![E_{i_\beta} \cdot E_{j_\beta}]\!]^I$. By the definition of $\leftarrow$ holds

$[\![E_{i_\beta} \cdot E_{j_\beta}]\!]^I = [\![(E_i \cdot E_j)_\beta]\!]^I$. Finally, from the assumption that $E = E_i \cdot E_j$ follows $[\![(E_i \cdot E_j)_\beta]\!]^I = [\![(E)_\beta]\!]^I = [\![E]\!]^{I^\beta}$ as desired. The argument for $E = E_i \cup E_j$ is the same.

Suppose that $E = (E_i)^*$. Then follows from the definition of $[\![\cdot]\!]^{I^\beta}$ that $[\![E]\!]^{I^\beta} = [\![(E_i)^*]\!]^{I^\beta} = \{(a,a) \mid a \in V(I^\beta)\} \cup [\![E_i]\!]^{I^\beta} \cup [\![E_i \cdot E_i]\!]^{I^\beta} \cup \dots$. We look at this expression in two parts $\{(a,a) \mid a \in V(I^\beta)\}$ and $[\![E_i]\!]^{I^\beta} \cup [\![E_i \cdot E_i]\!]^{I^\beta} \cup \dots$. In the first part, by action semantics $\{(a,a) \mid a \in V(I^\beta)\} = \{(a,a) \mid a \in V(I)\}$. In the second part, from semantics of $[\![\cdot]\!]^{I^\beta}$ follows that $[\![E_i]\!]^{I^\beta} \cup [\![E_i \cdot E_i]\!]^{I^\beta} \cup \dots = [\![E_i]\!]^{I^\beta} \cup [\![E_i]\!]^{I^\beta} \circ [\![E_i]\!]^{I^\beta} \cup \dots$. By assumption the structural complexity of $[\![E_i]\!]^{I^\beta}$ is at most n and thus by the induction hypothesis $[\![E_i]\!]^{I^\beta} \cup [\![E_i]\!]^{I^\beta} \circ [\![E_i]\!]^{I^\beta} \cup \dots = [\![E_{i_\beta}]\!]^I \cup [\![E_{i_\beta}]\!]^I \circ [\![E_{i_\beta}]\!]^I \cup \dots$. From semantics of $[\![\cdot]\!]^{I^\beta}$ follows again that $[\![E_{i_\beta}]\!]^I \cup [\![E_{i_\beta}]\!]^I \circ [\![E_{i_\beta}]\!]^I \cup \dots = [\![E_{i_\beta}]\!]^I \cup [\![E_{i_\beta} \cdot E_{i_\beta}]\!]^I \cup \dots$. We put the two parts together again and by definition of $[\![\cdot]\!]^I$ get $\{(a,a) \mid a \in V(I)\} \cup [\![E_{i_\beta}]\!]^I \cup [\![E_{i_\beta} \cdot E_{i_\beta}]\!]^I \cup \dots = [\![(E_{i_\beta})^*]\!]^I$. From definition of $\leftarrow$ follows that $[\![(E_{i_\beta})^*]\!]^I = [\![((E_i)^*)_\beta]\!]^I$. Finally, by our assumption that $E = (E_i)^*$ we get that $[\![((E_i)^*)_\beta]\!]^I = [\![E_\beta]\!]^I$ and thus, in summary, that $[\![E]\!]^{I^\beta} = [\![E_\beta]\!]^I$ as required. □

The following Claim 2 essentially is our goal property that G^α validates (C, T) if and only if G validates (C^α, T) for one specific shape expression and a single basic action. In this way, it acts as one induction base case for Lemma 2. Example 21 is an illustration for it. Recall that we said we may write ϕ_α to denote $\mathrm{TR}_\alpha(\phi)$ in Section 4.2.

Claim 2. *Let* $\mathrm{EQ}(E_m, E_n)$ *be a shape,* p *a property name,* r *a basic object property,* $\beta = \mathrm{p} \oplus \mathrm{r}$ *a basic action, and* I *an assignment. Then it holds that* $[\![\mathrm{EQ}(E_m, E_n)]\!]^{I^\beta} = [\![\mathrm{EQ}(E_m, E_n)_\beta]\!]^I$.

Proof. Let $i \in \{m, n\}$ and $P(m,n)$ be the property $[\![\mathrm{EQ}(E_m, E_n)]\!]^{I^\beta} = [\![\mathrm{EQ}(E_m, E_n)_\beta]\!]^I$ where E_m is of structural complexity m and E_n of structural complexity n. We perform a double induction on the structural complexity of E_i. This means, we use $P(0,0)$ as base case and then show that both $P(m,n) \implies P(m+1,n)$ and $P(m,n) \implies P(m,n+1)$. This will prove that $P(m,n)$ holds for all m and n. The implication $P(m,n) \implies P(m+1,n+1)$ alone does *not* prove P for all m and n, only for the cases where $m = n$ [21].

Base Case In the base case, we have structural complexity 0 for E_m and E_n. Then both E_m and E_n are basic object properties, say $E_m = \mathrm{r}_1$ and $E_n = \mathrm{r}_2$. As the structural complexity of path expressions only counts the operators that construct it and not the operators within basic object properties, it is still possible that, for example, $E_m = \mathrm{p}_1 \setminus \mathrm{p}_2$ with $\mathrm{p}_1, \mathrm{p}_2 \in \mathbb{O}$. We need to show that $[\![\mathrm{EQ}(\mathrm{r}_1, \mathrm{r}_2)]\!]^{I^\beta} = [\![\mathrm{EQ}(\mathrm{r}_1, \mathrm{r}_2)_\beta]\!]^I$. From semantics of $[\![\cdot]\!]^{I^\beta}$ follows $[\![\mathrm{EQ}(\mathrm{r}_1, \mathrm{r}_2)]\!]^{I^\beta} = \{a \mid \forall b : (a,b) \in [\![\mathrm{r}_1]\!]^{I^\beta} \text{ iff } (a,b) \in [\![\mathrm{r}_2]\!]^{I^\beta}\}$. From Lemma 1 follows that this equals $\{a \mid \forall b : (a,b) \in [\![\mathrm{r}_{1_\beta}]\!]^I \text{ iff } (a,b) \in [\![\mathrm{r}_{2_\beta}]\!]^I\}$ as basic object properties are base cases of path expressions. By semantics of $[\![\cdot]\!]^I$ this equals $[\![\mathrm{EQ}(\mathrm{r}_{1_\beta}, \mathrm{r}_{2_\beta})]\!]^I$. And by the definition of $\leftarrow$ this is the same as $[\![\mathrm{EQ}(\mathrm{r}_1, \mathrm{r}_2)_\beta]\!]^I$. This completes the base case.

Induction hypothesis Suppose that $P(m,n)$ holds. This means $\llbracket \mathrm{EQ}(E_m, E_n)\rrbracket^{I^\beta} = \llbracket \mathrm{EQ}(E_m, E_n)_\beta\rrbracket^{I}$ for any E_m with structural complexity m and any E_n with structural complexity n.

Induction step Without loss of generality, we will show only the step $P(m,n) \implies P(m+1,n)$. The argument for the implication $P(m,n) \implies P(m,n+1)$ is exactly the same. We prove for E_{m+1} of the forms $E_m \cup E'_m$, $E_m \cdot E'_m$, and $(E_m)^*$. Thus, there are three subcases.

In the first subcase, we assume that $E_{m+1} = E_m \cup E'_m$. Then $\llbracket \mathrm{EQ}(E_{m+1}, E_n)\rrbracket^{I^\beta} = \llbracket \mathrm{EQ}(E_m \cup E'_m, E_n)\rrbracket^{I^\beta}$. By semantics of $\llbracket\cdot\rrbracket^{I^\beta}$, this equals $\{a \mid \forall b : (a,b) \in \llbracket E_m \cup E'_m\rrbracket^{I^\beta}$ iff $(a,b) \in \llbracket E_n\rrbracket^{I^\beta}\}$ and then $\{a \mid \forall b : (a,b) \in \llbracket E_m\rrbracket^{I^\beta} \cup \llbracket E'_m\rrbracket^{I^\beta}$ iff $(a,b) \in \llbracket E_n\rrbracket^{I^\beta}\}$. By set properties, this equals $\{a \mid \forall b : (a,b) \in \llbracket E_m\rrbracket^{I^\beta}$ iff $(a,b) \in \llbracket E_n\rrbracket^{I^\beta}\} \cup \{a \mid \forall b : (a,b) \in \llbracket E'_m\rrbracket^{I^\beta}$ iff $(a,b) \in \llbracket E_n\rrbracket^{I^\beta}\}$. By semantics of $\llbracket\cdot\rrbracket^{I^\beta}$, this equals $\llbracket \mathrm{EQ}(E_m, E_n)\rrbracket^{I^\beta} \cup \llbracket \mathrm{EQ}(E'_m, E_n)\rrbracket^{I^\beta}$. As E_m and E'_m are path expressions of length m and E_n of length n, we can apply the induction hypothesis and get $\llbracket \mathrm{EQ}(E_m, E_n)_\beta\rrbracket^{I} \cup \llbracket \mathrm{EQ}(E'_m, E_n)_\beta\rrbracket^{I}$. Then, by the definition of $\leftarrow$, this equals $\llbracket \mathrm{EQ}(E_{m_\beta}, E_{n_\beta})\rrbracket^{I} \cup \llbracket \mathrm{EQ}(E'_{m_\beta}, E_{n_\beta})\rrbracket^{I}$. By semantics of $\llbracket\cdot\rrbracket^{I}$, we return to the set form $\{a \mid \forall b : (a,b) \in \llbracket E_{m_\beta}\rrbracket^{I}$ iff $(a,b) \in \llbracket E_{n_\beta}\rrbracket^{I}\} \cup \{a \mid \forall b : (a,b) \in \llbracket E'_{m_\beta}\rrbracket^{I}$ iff $(a,b) \in \llbracket E_{n_\beta}\rrbracket^{I}\}$. By set properties, this equals $\{a \mid \forall b : (a,b) \in \llbracket E_{m_\beta}\rrbracket^{I} \cup \llbracket E'_{m_\beta}\rrbracket^{I}$ iff $(a,b) \in \llbracket E_{n_\beta}\rrbracket^{I}\}$. By semantics of $\llbracket\cdot\rrbracket^{I}$, this equals $\{a \mid \forall b : (a,b) \in \llbracket E_{m_\beta} \cup E'_{m_\beta}\rrbracket^{I}$ iff $(a,b) \in \llbracket E_{n_\beta}\rrbracket^{I}\}$. By definition of $\leftarrow$, this equals $\{a \mid \forall b : (a,b) \in \llbracket (E_m \cup E'_m)_\beta\rrbracket^{I}$ iff $(a,b) \in \llbracket E_{n_\beta}\rrbracket^{I}\}$. By definition of $\leftarrow$, this equals $\{a \mid \forall b : (a,b) \in \llbracket (E_m \cup E'_m)_\beta\rrbracket^{I}$ iff $(a,b) \in \llbracket E_{n_\beta}\rrbracket^{I}\}$. From the initial assumption that $E_{m+1} = E_m \cup E'_m$ follows the equality with $\{a \mid \forall b : (a,b) \in \llbracket E_{{m+1}_\beta}\rrbracket^{I}$ iff $(a,b) \in \llbracket E_{n_\beta}\rrbracket^{I}\}$. By semantics of $\llbracket\cdot\rrbracket^{I}$ is equal to $\llbracket \mathrm{EQ}(E_{{m+1}_\beta}, E_{n_\beta})\rrbracket^{I}\}$. Finally, by definition of $\leftarrow$, this equals $\llbracket \mathrm{EQ}(E_{m+1}, E_n)_\beta\rrbracket^{I}\}$. Thus, we have shown that $\llbracket \mathrm{EQ}(E_{m+1}, E_n)\rrbracket^{I^\beta} = \llbracket \mathrm{EQ}(E_{m+1}, E_n)_\beta\rrbracket^{I}$ for the first subcase with $E_{m+1} = E_m \cup E'_m$.

We proceed with the second subcase and assume that $E_{m+1} = E_m \cdot E'_m$. Then, by assumption $\llbracket \mathrm{EQ}(E_{m+1}, E_n)\rrbracket^{I^\beta} = \llbracket \mathrm{EQ}(E_m \cdot E'_m, E_n)\rrbracket^{I^\beta}$. By semantics of $\llbracket\cdot\rrbracket^{I^\beta}$ holds $\llbracket \mathrm{EQ}(E_m \cdot E'_m, E_n)\rrbracket^{I^\beta} = \{a \mid \forall b : (a,b) \in \llbracket E_m \cdot E'_m\rrbracket^{I^\beta}$ iff $(a,b) \in \llbracket E_n\rrbracket^{I^\beta}\} = \{a \mid \forall b : (a,b) \in \llbracket E_m\rrbracket^{I^\beta} \circ \llbracket E'_m\rrbracket^{I^\beta}$ iff $(a,b) \in \llbracket E_n\rrbracket^{I^\beta}\}$. From Lemma 1 follows that $\{a \mid \forall b : (a,b) \in \llbracket E_m\rrbracket^{I^\beta} \circ \llbracket E'_m\rrbracket^{I^\beta}$ iff $(a,b) \in \llbracket E_n\rrbracket^{I^\beta}\} = \{a \mid \forall b : (a,b) \in \llbracket E_{m_\beta}\rrbracket^{I} \circ \llbracket E'_{m_\beta}\rrbracket^{I}$ iff $(a,b) \in \llbracket E_{n_\beta}\rrbracket^{I}\}$. By semantics of $\llbracket\cdot\rrbracket^{I}$ holds $\{a \mid \forall b : (a,b) \in \llbracket E_{m_\beta}\rrbracket^{I} \circ \llbracket E'_{m_\beta}\rrbracket^{I}$ iff $(a,b) \in \llbracket E_{n_\beta}\rrbracket^{I}\} = \{a \mid \forall b : (a,b) \in \llbracket E_{m_\beta} \cdot E'_{m_\beta}\rrbracket^{I}$ iff $(a,b) \in \llbracket E_{n_\beta}\rrbracket^{I}\}$. From the definition of $\leftarrow$ follows that $\{a \mid \forall b : (a,b) \in \llbracket E_{m_\beta} \cdot E'_{m_\beta}\rrbracket^{I}$ iff $(a,b) \in \llbracket E_{n_\beta}\rrbracket^{I}\} = \{a \mid \forall b : (a,b) \in \llbracket (E_m \cdot E'_m)_\beta\rrbracket^{I}$ iff $(a,b) \in \llbracket E_{n_\beta}\rrbracket^{I}\}$. Then from the definition of $\llbracket\cdot\rrbracket^{I}$ and the assumption $E_{m+1} = E_m \cdot E'_m$ follows that $\{a \mid \forall b : (a,b) \in \llbracket\rrbracket^{I}$ iff $(a,b) \in \llbracket E_{n_\beta}\rrbracket^{I}\} = \llbracket \mathrm{EQ}((E_m \cdot E'_m)_\beta, E_{n_\beta})\rrbracket^{I^\beta} = \llbracket \mathrm{EQ}((E_{m+1})_\beta, E_{n_\beta})\rrbracket^{I^\beta}$ and thus for this subcase holds in summary that $\llbracket \mathrm{EQ}(E'_m, E_n)\rrbracket^{I^\beta} = \llbracket \mathrm{EQ}(E'_m, E_n)_\beta\rrbracket^{I}$ as required.

We proceed with the third subcase and assume that $E_{m+1} = (E_m)^*$. We need to show that $[\![\mathrm{EQ}((E_m)^*, E_n)]\!]^{I^\beta} = [\![\mathrm{EQ}((E_m)^*, E_n)_\beta]\!]^I$. By assumption $[\![\mathrm{EQ}(E_{m+1}, E_n)]\!]^{I^\beta} = [\![\mathrm{EQ}((E_m)^*, E_n)]\!]^{I^\beta}$. From the semantics of $[\![\cdot]\!]^{I^\beta}$ follows that $[\![\mathrm{EQ}((E_m)^*, E_n)]\!]^{I^\beta} = \{a \mid \forall b : (a,b) \in [\![(E_m)^*]\!]^{I^\beta}$ iff $(a,b) \in [\![E_n]\!]^{I^\beta}\}$. From Lemma 1 follows that $\{a \mid \forall b : (a,b) \in [\![(E_m)^*]\!]^{I^\beta}$ iff $(a,b) \in [\![E_n]\!]^{I^\beta}\} = \{a \mid \forall b : (a,b) \in [\![((E_m)^*)_\beta]\!]^I$ iff $(a,b) \in [\![E_{n_\beta}]\!]^I\}$. From the semantics of $[\![\cdot]\!]^I$ follows that $\{a \mid \forall b : (a,b) \in [\![((E_m)^*)_\beta]\!]^I$ iff $(a,b) \in [\![E_{n_\beta}]\!]^I\} = [\![\mathrm{EQ}(E_{m+1_\beta}, E_{n_\beta})]\!]^I$. Thus, for this subcase holds in summary that $[\![\mathrm{EQ}((E_m)^*, E_n)]\!]^{I^\beta} = [\![\mathrm{EQ}((E_m)^*, E_n)_\beta]\!]^I$ and $[\![\mathrm{EQ}(E'_m, E_n)]\!]^{I^\beta} = [\![\mathrm{EQ}(E'_m, E_n)_\beta]\!]^I$ as required. □

The following Lemma 2 shows the final Theorem 2 restricted to arbitrary single shape expressions under a single update and with a single assignment.

Lemma 2. *Let ϕ be a shape expression, β a basic action, and I an assignment. Then* $[\![\phi]\!]^{I^\beta} = [\![\phi_\beta]\!]^I$.

Proof. Proof by induction on the structure of ϕ.

Base cases For the base cases, we consider ϕ of the form $\top$, individual t, shape name s, class name A, $\circledast(d, \psi)$, and $\mathrm{EQ}(E, E')$.

We start with three very similar base cases. Suppose that $\phi = \top$. Then $\phi_\alpha = \phi$. Thus, it remains to show that $[\![\phi]\!]^{I^\beta} = [\![\phi]\!]^I$. By semantics $[\![\phi]\!]^{I'} = V(I')$ for any assignment I'. Thus, $[\![\phi]\!]^{I^\beta} = [\![\phi]\!]^I$ as required. Suppose that $\phi = t$. By semantics $[\![\phi]\!]^{I'} = t$ for any assignment I'. Then the argument is as for $\phi = \top$. Suppose that $\phi = s$. L is the same in I and I^β. Then by semantics $[\![\phi]\!]^I = [\![\phi]\!]^{I^\beta} = [\![\phi]\!]^L$. The remaining argument is as for $\phi = \top$.

Suppose that $\phi = \mathrm{B}$ with $\mathrm{B} \in \mathbb{C}$. If β is of the form $\mathrm{p} \oplus \mathrm{r}$ or $\mathrm{p} \ominus \mathrm{r}$, then $\phi_\alpha = \phi$ directly. Furthermore, such β cannot add unary atoms $\mathrm{B}(b)$ to G for any individual b. Thus, $[\![\phi]\!]^{I^\beta} = [\![\phi_\alpha]\!]^I$ for such β. Otherwise, β is of the form $\mathrm{A} \oplus \phi_c$ or $\mathrm{A} \ominus \phi_c$. Assume without loss of generality that $\beta = \mathrm{A} \oplus \phi_c$. Then, there are two subcases:

First, suppose that $\mathrm{B} \neq \mathrm{A}$. Then $\phi_\alpha = \phi_{\mathrm{A} \leftarrow \mathrm{A} \vee \phi_c} = \phi$. Thus, it remains to show that $[\![\phi]\!]^{I^\beta} = [\![\phi]\!]^I$. β does not add the class B to any individual in G because $G^{(\mathrm{A} \oplus \phi_c)} = G \cup \left\{\mathrm{A}(v) \mid v \in [\![\phi_c]\!]^I\right\}$. Thus, $[\![\phi]\!]^{I^\beta} = [\![\phi]\!]^I$ as required for the first subcase.

Second, suppose that $\mathrm{B} = \mathrm{A}$. Then, by assumption $[\![\phi]\!]^{I^\beta} = [\![\mathrm{A}]\!]^{I^\beta}$. From the definition of $[\![\cdot]\!]^{I^\beta}$ follows that $[\![\mathrm{A}]\!]^{I^\beta} = \{a \mid \mathrm{A}(a) \in I^\beta\}$. By definition of I^β and $[\![\cdot]\!]^{I^\beta}$ holds that $\{a \mid \mathrm{A}(a) \in I^\beta\} = \{a \mid \mathrm{A}(a) \in G^\beta\}$. As by assumption that $\beta = \mathrm{A} \oplus \phi_c$ and semantics of actions holds that $G^\beta = G \cup \{\mathrm{A}(b) \mid b \in [\![\phi_c]\!]^G\}$ and thus, we can split the set $\{a \mid \mathrm{A}(a) \in G^\beta\}$ up into $\{a \mid \mathrm{A}(a) \in G\} \cup [\![\phi_c]\!]^G$. From the definition of I and as concepts like ϕ_c do not have shape names by definition, it follows that $\{a \mid \mathrm{A}(a) \in G\} \cup [\![\phi_c]\!]^G = \{a \mid \mathrm{A}(a) \in I\} \cup [\![\phi_c]\!]^I$. From the definition of $[\![\cdot]\!]^I$ follows that $\{a \mid \mathrm{A}(a) \in I\} \cup [\![\phi_c]\!]^I = [\![\mathrm{A}]\!]^I \cup [\![\phi_c]\!]^I = [\![\mathrm{A} \vee \phi_c]\!]^I$. From the definition of $\leftarrow$ follows that $[\![\mathrm{A} \vee \phi_c]\!]^I = [\![\mathrm{A}_{\mathrm{A} \leftarrow \mathrm{A} \vee \phi_c}]\!]^I$ and by assumption $[\![\mathrm{A}_{\mathrm{A} \leftarrow \mathrm{A} \vee \phi_c}]\!]^I = [\![\phi_{\mathrm{A} \leftarrow \mathrm{A} \vee \phi_c}]\!]^I$ and thus, in summary, $[\![\phi]\!]^{I^\beta} = [\![\phi_\beta]\!]^I$ as required for the second subcase. This concludes the base case $\phi = \mathrm{B}$.

Suppose that $\phi = \circledast(d, \psi)$ with $\psi \in \mathbb{D} \cup \mathbb{L}$. β can be of four different forms $\mathrm{A} \oplus \phi_c$, $\mathrm{A} \ominus \phi_c$, $\mathrm{p} \oplus \mathrm{r}$, and $\mathrm{p} \ominus \mathrm{r}$. By definition $\mathrm{d} \in \mathbb{D}$, $\mathrm{p} \in \mathbb{O}$, $\mathrm{A} \in \mathbb{C}$ and $\mathbb{D}$, $\mathbb{O}$ and $\mathbb{C}$ are disjoint. Then $\phi_\alpha = \phi$ for all four cases because there is neither a class name A nor an object property name p in ϕ. Thus, it remains to show that $[\![\phi]\!]^{I^\beta} = [\![\phi]\!]^I$. ϕ is evaluated using binary atoms $\mathrm{d}(a, b)$ with $\mathrm{d} \in \mathbb{D}$ both if $\psi \in \mathbb{L}$ and if $\psi \in \mathbb{D}$. In none of its four possible forms does β add such atoms. From the disjointedness of $\mathbb{D}$, $\mathbb{C}$, and $\mathbb{O}$ follows $[\![\phi]\!]^{I^\beta} = [\![\phi]\!]^I$.

Suppose that $\phi = \mathrm{EQ}(E, E')$. If β is of the form $\mathrm{A} \oplus \phi_c$ or $\mathrm{A} \ominus \phi_c$ then the argument is like for $\phi = \circledast(d, \psi)$. If β is of the form $\mathrm{p} \oplus \mathrm{r}$ then we can directly use Claim 2. Finally, if β is of the form $\mathrm{p} \ominus \mathrm{r}$ then we can still use Claim 2 as its proofs works analogously for such β.

Induction hypotheses $[\![\phi_1]\!]^{I^\beta} = [\![{\phi_1}_\beta]\!]^I$ and $[\![\phi_2]\!]^{I^\beta} = [\![{\phi_2}_\beta]\!]^I$.

Induction step We consider ϕ of the forms $\phi = \phi_1 \wedge \phi_2$, $\phi = \neg\phi_1$, and $\phi = \geq_n E.\phi_1$.

Suppose that $\phi = \phi_1 \wedge \phi_2$. Then, by assumption $[\![\phi]\!]^{I^\beta} = [\![\phi_1 \wedge \phi_2]\!]^{I^\beta}$. From the definition of $[\![\cdot]\!]^{I^\beta}$ follows that $[\![\phi_1 \wedge \phi_2]\!]^{I^\beta} = [\![\phi_1]\!]^{I^\beta} \cap [\![\phi_2]\!]^{I^\beta}$. By induction hypotheses $[\![\phi_1]\!]^{I^\beta} \cap [\![\phi_2]\!]^{I^\beta} = [\![{\phi_1}_\beta]\!]^I \cap [\![{\phi_2}_\beta]\!]^I$. From the definition of $[\![\cdot]\!]^I$ follows that $[\![{\phi_1}_\beta]\!]^I \cap [\![{\phi_2}_\beta]\!]^I = [\![{\phi_1}_\beta \wedge {\phi_2}_\beta]\!]^I$. From the definition of $\leftarrow$ follows that $[\![{\phi_1}_\beta \wedge {\phi_2}_\beta]\!]^I = [\![(\phi_1 \wedge \phi_2)_\beta]\!]^I$. From the initial assumption $\phi = \phi_1 \wedge \phi_2$ follows that $[\![(\phi_1 \wedge \phi_2)_\beta]\!]^I = [\![\phi_\beta]\!]^I$ and thus, in summary, $[\![\phi]\!]^{I^\beta} = [\![\phi_\beta]\!]^I$ as required.

Suppose that $\phi = \neg\phi_1$. Then, by assumption $[\![\phi]\!]^{I^\beta} = [\![\neg\phi_1]\!]^{I^\beta}$. From the definition of $[\![\cdot]\!]^{I^\beta}$ follows that $[\![\neg\phi_1]\!]^{I^\beta} = V(I^\beta) \setminus [\![\phi_1]\!]^{I^\beta}$. From the definition of $[\![\cdot]\!]^{I^\beta}$ follows that $V(I^\beta) = V(I)$ and thus $V(I^\beta) \setminus [\![\phi_1]\!]^{I^\beta} = V(I) \setminus [\![\phi_1]\!]^{I^\beta}$. By induction hypothesis $V(I) \setminus [\![\phi_1]\!]^{I^\beta} = V(I) \setminus [\![{\phi_1}_\beta]\!]^I$. From the definition of $[\![\cdot]\!]^I$ follows that $V(I) \setminus [\![{\phi_1}_\beta]\!]^I = [\![\neg({\phi_1}_\beta)]\!]^I$. From the definition of $\leftarrow$ follows that $[\![\neg({\phi_1}_\beta)]\!]^I = [\![(\neg\phi_1)_\beta]\!]^I$. From the initial assumption $[\![\phi]\!]^{I^\beta} = [\![\neg\phi_1]\!]^{I^\beta}$ follows that $[\![(\neg\phi_1)_\beta]\!]^I = [\![\phi_\beta]\!]^I$ and thus, in summary, $[\![\phi]\!]^{I^\beta} = [\![\phi_\beta]\!]^I$ as required.

Suppose that $\phi = \geq_n E.\phi_1$. Then by semantics $[\![\phi]\!]^{I^\beta} = \{a \mid |\{(a, b) \in [\![E]\!]^{I^\beta}$ and $b \in [\![\phi_1]\!]^{I^\beta}\}| \geq n\}$. We apply the induction hypothesis to get $[\![\phi]\!]^{I^\beta} = \{a \mid |\{(a, b) \in [\![E]\!]^{I^\beta}$ and $b \in [\![{\phi_1}_\beta]\!]^I\}| \geq n\}$. We need to look at two cases.

In the first case, β is of the form $\mathrm{A} \oplus \phi_c$ or $\mathrm{A} \ominus \phi_c$. Then it does not add any binary atoms $\mathrm{p}(a, b)$ to G by definition. However, E is evaluated using such atoms. Therefore, in this case $[\![E]\!]^{I^\beta} = [\![E]\!]^I$. Furthermore, in this case $E = E_\beta$ because A cannot be in E by definition. Therefore, $[\![E]\!]^{I^\beta} = [\![E_\beta]\!]^{I^\beta}$. In the second case, β is of the form $\mathrm{p} \oplus \mathrm{r}$ or $\mathrm{p} \ominus \mathrm{r}$. Then we can without loss of generality apply Lemma 1 to get $[\![E]\!]^{I^\beta} = [\![E_\beta]\!]^{I^\beta}$.

We got the same intermediate result for both cases, and from them follows that $[\![\phi]\!]^{I^\beta} = \{a \mid |\{(a,b) \in [\![E_\beta]\!]^I \text{ and } b \in [\![\phi_{1_\beta}]\!]^I\}| \geq n\}$. Then by definition of $\leftarrow$ holds for both of them $[\![\phi]\!]^{I^\beta} = (\{a \mid |\{(a,b) \in [\![E]\!]^I \text{ and } b \in [\![\phi_1]\!]^I\}| \geq n\})_\beta$. Then by semantics $[\![\phi]\!]^{I^\beta} = (\geq_n E.\phi_1)_\beta$ and then we get the final result $[\![\phi]\!]^{I^\beta} = [\![\phi_\beta]\!]^I$. This completes the induction step. □

In Theorem 1, we extend Lemma 2 and take the step from simple to complex actions, and from shape expressions to constraints.

Theorem 1 ([2])**.** *Let α be a ground complex action, I an assignment, $s \leftrightarrow \phi$ a constraint. Then, $I^\alpha \models s \leftrightarrow \phi$ if and only if $I \models \mathrm{TR}^{\leftrightarrow}_{\alpha}(s \leftrightarrow \phi)$.*

Proof. Proof by induction on $\ell(\alpha)$ defined as $\ell(\epsilon) = 0$ and $\ell(\beta \cdot \alpha) = 1 + \ell(\alpha)$.

Base Case The base case has $\alpha = \epsilon$ and $\ell(\alpha) = 0$. Then $S_\alpha(G) = G$ and $\mathrm{TR}^{\leftrightarrow}_{\alpha}(s \leftrightarrow \phi) = s \leftrightarrow \phi$ by definition, and thus, the claim holds.

Induction step First, we show that $I^\beta \models s' \leftrightarrow \phi'$ iff $I \models s' \leftrightarrow \phi'_\beta$ for any constraint $s' \leftrightarrow \phi'$ and basic action β. To see this, suppose that $I^\beta \models s' \leftrightarrow \phi'$. Then $[\![s']\!]^L = [\![\phi']\!]^{I^\beta}$ by definition of models. But then also $[\![s']\!]^L = [\![\phi'_\beta]\!]^I$ by Lemma 2. Thus, $I \models s' \leftrightarrow \phi'_\beta$ by definition of models. The converse direction works equally and the equivalence holds.

We will show just for $\beta = (\mathrm{A} \oplus \phi_c) \cdot \alpha'$, as for the cases $\alpha = (\mathrm{A} \ominus \phi_c) \cdot \alpha'$, $\alpha = (\mathrm{p} \oplus \mathrm{r}^+) \cdot \alpha'$ and $\alpha = (\mathrm{p} \ominus \mathrm{r}^+) \cdot \alpha'$ the argument is analogous. Let $I_1^\alpha = S_{\mathrm{A} \oplus \phi_c}(G) \cup L$ be the result of applying just $\mathrm{A} \oplus \phi_c$ to I.

By the preliminary result of this induction step holds in particular $I_1^\alpha \models \mathrm{TR}^{\leftrightarrow}_{\alpha'}(s \leftrightarrow \phi)$ iff $I \models (\mathrm{TR}^{\leftrightarrow}_{\alpha'}(s \leftrightarrow \phi))_{\mathrm{A} \leftarrow \mathrm{A} \vee \phi_c}$. Since $(\mathrm{TR}^{\leftrightarrow}_{\alpha'}(s \leftrightarrow \phi))_{\mathrm{A} \leftarrow \mathrm{A} \vee \phi_c} = \mathrm{TR}^{\leftrightarrow}_{\alpha}(s \leftrightarrow \phi)$ we get $I_1^\alpha \models \mathrm{TR}^{\leftrightarrow}_{\alpha'}(s \leftrightarrow \phi)$ iff $I \models \mathrm{TR}^{\leftrightarrow}_{\alpha}(s \leftrightarrow \phi)$. By the induction hypothesis holds $I_1^\alpha \models \mathrm{TR}^{\leftrightarrow}_{\alpha'}(s \leftrightarrow \phi)$ iff $S_{\alpha'}(I_1^\alpha) \cup L \models s \leftrightarrow \phi$. Thus, $I \models \mathrm{TR}^{\leftrightarrow}_{\alpha}(s \leftrightarrow \phi)$ iff $S_{\alpha'}(I_1^\alpha) \cup L \models s \leftrightarrow \phi$. By definition $S_{\alpha'}(I_1^\alpha) = S_{\alpha'}(S_{(\mathrm{A} \oplus \phi_c)}(I)) = S_\alpha(I)$. Thus, as desired $I \models \mathrm{TR}^{\leftrightarrow}_{\alpha}(s \leftrightarrow \phi)$ iff $I^\alpha \models s \leftrightarrow \phi$. □

Finally, Theorem 2 is our main result. It extends 1 by taking sets of constraints and targets into account.

Theorem 2. *Let α be a ground complex action, G a data graph and (C, T) a shape document. Then G^α validates (C, T) if and only if G validates (C^α, T).*

Proof. $\Rightarrow$ Suppose that G^α validates (C, T). Then by definition of assignments there is a set of shape atoms L such that $G^\alpha \cup L \models C$ and $T \subseteq L$. Then $\forall (s \leftrightarrow \phi) \in C : G^\alpha \cup L \models s \leftrightarrow \phi$ by definition. Then $\forall (s \leftrightarrow \phi) \in C : G \cup L \models \mathrm{TR}^{\leftrightarrow}_{\alpha}(s \leftrightarrow \phi)$ follows from Theorem 1 because $G^\alpha = S_\alpha(G)$. By definition $\{\mathrm{TR}^{\leftrightarrow}_{\alpha}(s \leftrightarrow \phi) \mid s \leftrightarrow \phi \in C\} = \mathrm{TR}^{\leftrightarrow}_{\alpha}(C) = C^\alpha$.

Thus $G \cup L \models C^\alpha$. It is crucial to observe that T and L remain unchanged. Because of this, the second condition $T \subseteq L$ still holds and G validates (C^α, T) as required.

$\Leftarrow$ Suppose that G validates (C^α, T). Then by definition of assignments there is a set of shape atoms L such that $G \cup L \models C^\alpha$ and $T \subseteq L$. Then $\forall (s' \leftrightarrow \phi') \in C^\alpha : G \cup L \models s' \leftrightarrow \phi'$ by definition. In addition, $C^\alpha = \mathrm{TR}_\alpha^\leftrightarrow(C) = \{\mathrm{TR}_\alpha^\leftrightarrow(s \leftrightarrow \phi) \mid s \leftrightarrow \phi \in C\}$ by definition. Consequently, $\forall (s' \leftrightarrow \phi') \in \{\mathrm{TR}_\alpha^\leftrightarrow(s \leftrightarrow \phi) \mid s \leftrightarrow \phi \in C\} : G \cup L \models s' \leftrightarrow \phi'$. Then $\forall (s \leftrightarrow \phi) \in C : G^\alpha \cup L \models s \leftrightarrow \phi$ follows from Theorem 1 because $G^\alpha = S_\alpha(G)$. As T and L remain unchanged, the second condition $T \subseteq L$ still holds and G^α validates C, T as required. $\square$

This completes the proof of correctness of the new static validation technique.

CHAPTER 6

Implementation and Experiments

In this chapter, we initially outline some details about the prototype implementation. We then proceed with an explanation of the experiments and their results. Our implementation of the transformation and action algorithm is available at

`https://github.com/dominicjaeger/validate-transforming-rdf`.

6.1 Implementation considerations

In this section, we first provide some details about the scope of our prototype. Then we compare various validators to our chosen validator. Lastly, we explain the update language implementation and how the transformation algorithm works on the shapes graph.

6.1.1 Prototype

We explain the possibilities and limitations of our prototype implementation. As starting point for our prototype, we chose the validator of the Apache Jena Project [6]. We use it to load graphs from files and remote locations, and for some operations on graphs.

The prototype consists of two main parts. The first part is the update language implementation. The implementation is able to parse a sequence of updates or rather actions that is written in a text file. Then inserting and removing nodes from the data graph according to the action semantics is done with the help of the Apache Jena methods.

The second part is the transformation implementation. In practice, the shape documents from the formalization correspond to *shape graphs* that are written down in text files. Therefore, our implementation adds nodes to and removes nodes from those shapes graphs, and it does so using Apache Jena again.

At this point, it is important to see the difference between two different types of graph updates. First, we have updates on data graphs, which we call actions. Second, we have updates on the shapes graph, which correspond to the transformation of constraints.

We can see in the SHACL part of the documentation of the Jena project that it implements SHACL Core and SHACL SPARQL constraints [6]. The components from SHACL Core include SHACL property paths for sequence, alternative, inverse, zero-or-more, one-or-more, and zero-or-one, but nothing like the difference or intersection that we introduced in Chapter 3 . Furthermore, it does not yet implement node expressions, expression constraints, and SHACL rules [26]. We have already shown some details about node expressions in Section 2.2.2. We can additionally verify these statements and track the development in the public GitHub repository of the Apache Jena project [5].

Our proof-of-concept implementation does not extend the Apache Jena validator itself. Therefore, validation is restricted to the fragment of our SHACL definition that is supported by it, which means SHACL Core and SHACL SPARQL constraints [6]. In particular, literals and predicates are part of SHACL Core in the form of value range constraint components [26].

There is one limitation in the transformation part of our prototype implementation. As SHACL core does not contain negation in property paths, it is not possible to create a proper shapes graph for the substitution $\mathrm{p} \leftarrow \mathrm{p} \setminus \mathrm{r}$ that we use for actions $\mathrm{p} \ominus \mathrm{r}$. Unfortunately, we are not aware of any validator that supports this. Presently, our prototype contains a workaround using the SHACL Core property sh:closed. Using this property forbids properties that are not explicitly enumerated via property shapes or belong to the property sh:ignoredProperties [26]. With this workaround, we can capture a few cases for the action $\mathrm{p} \ominus \mathrm{r}$.

A second limitation is about the action part of our prototype implementation. In our understanding, the W3C SHACL recommendation [26] and the Apache Jena validator [6] do not allow interpreting a shape or a basic object property as we need it for the actions. In other words, getting the sets of individuals $[\![\phi_c]\!]^I$ and $[\![\mathrm{r}]\!]^I$ is not possible directly. We can work around this for the actions $\mathrm{A} \oplus \phi_c$ and $\mathrm{A} \ominus \phi_c$ with repeated validation and ephemeral target node declarations. However, doing the same for the actions $\mathrm{p} \oplus \mathrm{r}$ and $\mathrm{p} \ominus \mathrm{r}$ is not possible as in our understanding, properties cannot be targets. Even if they could be, the restrictions for SHACL property paths would still apply, and direct translations for basic object properties like $\mathrm{r} \setminus \mathrm{r}'$ would not be possible. As a result, our update language implementation supports only object property names $\mathrm{p} \in \mathbb{O}$. Table 6.1 contains a summary of the supported actions for our prototype. The first line *Transformation* considers the transformation part of our implementation, the second line *Action Application* considers the part of our implementation that parses actions and updates the data graph.

We continue with a limitation about the shape $\mathrm{EQ}(E, E')$. The W3C SHACL recommendation defines that the „values of sh:equals in a shape are IRIs“ [26]. By definition, the only SHACL property paths that are IRIs are predicate paths. The other SHACL

Table 6.1: Supported actions in the implementation

Support \ Action	A $\oplus$ ϕ_c	A $\ominus$ ϕ_c	p $\oplus$ r	p $\ominus$ r
Transformation	Full	Full	Full	Partial
Action Application	Full	Full	Partial	Partial

property paths are blank nodes [26]. Thus, the other property paths cannot be values of sh:equals and therefore, expressing EQ(E, E') is not possible directly. This issue is the topic of discussion of the editors of the recommendation [26] on the W3C GitHub Forum [29].

6.1.2 Alternative approaches

We now compare the capabilities and limitations of alternative SHACL validators and other possible approaches for the problem.

One alternative SHACL engine is the Python implementation [34] of *Trav-SHACL* [20]. This engine does not support sh:or, which we need for the substitution A $\leftarrow$ A $\vee$ ϕ_s of the action A $\oplus$ ϕ_c.

The *rdf-validate-shacl* project implements the SHACL W3C recommendation using pure JavaScript [33]. The implementation of the paths corresponds precisely to the recommendation, and thus, this program brings us no further than the Apache Jena validator.

As SPARQL property paths have negation, the most promising alternative originates from a paper that uses SPARQL to validate SHACL constraints [16]. This project is called *SHACL2SPARQL*. However, it supports only a limited subset of the SHACL W3C recommendation, too. We can see in the code that among the unsupported features are classes which we need for the actions A $\oplus$ ϕ_c and A $\ominus$ ϕ_c. Furthermore, the grammar for shapes in the paper [16] specifically talks about SHACL paths and not SPARQL paths and in addition, the parser of the implementation contains cases only for predicate, inverse, zeroOrMore, sequence and alternative paths, but no negation [13].

While the SHACL Advanced Features document contains sh: filter and sh: intersection , it does not extend the syntax of SHACL paths [24]. In other words, even if a validator implements the advanced features, it is not possible to perform the transformation directly.

Something similar holds for the SPARQL-based constraints that are part of the SHACL recommendation [26]. SPARQL 1.1 supports negation in property paths and also MINUS and FILTER NOT EXISTS in queries [22]. However, it is not possible to directly use these queries for our transformation. Two aspects are crucial for this: First, the syntax for SHACL property paths essentially allows only IRIs and blank nodes. Second, the values of the components sh: select and sh:ask are defined to be literals of datatype xsd: string

[26]. We can see an example for this in Listing 6.11. Figure 6.1 illustrates the shapes graph corresponding to Listing 6.11. Therefore, it is not possible to edit the shapes graph in the same way we do it for the other actions.

6.1.3 Update language and transformation implementation

We will now explain how actions can be written and used in the implementation. Furthermore, we explain what transformations happen according to specific actions and demonstrate how we implement the transformation with the shapes graphs in the Apache Jena framework.

Listing 6.1 shows how the action $A \oplus \phi_c$ can be written for our implementation. Presently, our parser does not support prefixes, and the IRIs need to be fully written down.

Listing 6.1: An example for an action

```
http://example.com/ns#A + http://example.com/ns#ConceptShape
```

In Listing 6.1, the URI http://example.com/ns#A is for the class A, and for the concept ϕ_c we have the URI http://example.com/ns#ConceptShape. It is required that the shape which is denoted by the latter URI is given in the shapes graph. In contrast to Turtle syntax [8], IRIs are not enclosed in $<$ and $>$ here.

Alternatively, and with the same meaning, it is also possible to write the + or - in the front. An example for this is displayed in Listing 6.2.

Listing 6.2: Action with minus in the beginning

```
-  http://example.com/ns#A  http://example.com/ns#ConceptShape
```

We will demonstrate some possible actions and transformations. Suppose we want to transform the shapes graph from Listing 6.3 instead of applying the action of Listing 6.1. To do so, we need to use the sh:or constraint and essentially get as our C^{α} the shapes graph from Listing 6.4.

The graph denoted in Listing 6.4 corresponds to the one in Figure 6.2. The formalization from Chapter 3 does not have blank nodes. In the implementation, however, it is crucial to take the blank nodes into account for the construction of the required SHACL list of the transformed shape. In the figures, blank nodes are denoted by circles without a label within them.

Listing 6.5 and Figure 6.4 show the result of the transformation

$$(\geq_1 (\mathrm{ex:loves}).(\mathrm{ex:Person}))_{\mathrm{ex:loves} \leftarrow \mathrm{ex:loves} \cup \mathrm{ex:likes}}$$

using sh:alternativePath. To perform this transformation, we replace `sh:path ex:loves` in the original graph by `sh:path [ sh:alternativePath ( ex:loves ex:likes ) ]`.

Listing 6.6 demonstrates a negation constraint component in Turtle syntax. Figure 6.3 shows it as a graph. We use negation in the transformation $A \leftarrow A \wedge \neg\phi_c$ for actions

A $\ominus$ ϕ_c. We use the fact that some SHACL constraint components declare only a single parameter. In such a situation, the parameters may be used multiple times within the same shape and the interpretation of them is in conjunction. We are not required to use `sh:and` here.

6.2 Experiments

We experimented with two data sets, the Medical Subject Headings (MeSH) and Yago dataset. The actions and shapes for the experiments are available in our code repository.

6.2.1 MeSH Experiment

The MeSH data set is a terminology about biomedical information published by the U.S. National Library of Medicine [18]. One advantage of it is that all data is contained within one single text file that can be downloaded from `https://www.nlm.nih.gov/databases/download/mesh.html`. This way, it allows rapid evaluation of the implementation. Alternatively, it can also be queried using SPARQL. Furthermore, this data file is 1.8 GB large and contains approximately 15 million lines, permitting insights about performance as well.

The main part of the MeSH experiment is to add two additional property names and replace one class name with a shorter one. To do so, we use the actions from Listing 6.7. In the following, we show the abstract syntax for the actions from Listing 6.7. Similar to prefixes, we abbreviate *http://example.com/ns#* with *ex:* and *http://id.nlm.nih.gov/mesh/vocab#* with *nlm:*.

$$\begin{aligned}
\text{ex:id} &\oplus \text{nlm:identifier}\\
\text{ex:see} &\oplus \text{nlm:seeAlso}\\
\text{ex:GeoDescriptor} &\oplus \text{nlm:GeographicalDescriptor}\\
\text{nlm:GeographicalDescriptor} &\ominus \text{ex:GeoDescriptor}
\end{aligned}$$

Validation in the MeSH experiment is done with a set of shapes that we defined ourselves and that are displayed in Listing 6.8. In the following, we give the formal representations of those shapes. In addition, they are available in the code repository. The shape *hasIdC000724057* additionally checks the data type for the value node, which is not directly expressible in the formalization, but which we have indicated in Section 3.3.2. We use the same abbreviations as before.

$$\begin{aligned}
\mathit{hasDateCreated} &\leftrightarrow \exists \text{nlm:dateCreated}.\top\\
\mathit{hasSomeNewId} &\leftrightarrow \exists \text{ex:id}.\top\\
\mathit{hasIdC000724057} &\leftrightarrow \exists \text{nlm:identifier}.\top
\end{aligned}$$

Updating the 15 163 366 original statements of the MeSH data led to a total of 16 823 671 statements and took 7 minutes and 59 seconds. The number of statements increased by 11%. The time to transform the shapes graph is 0.019 seconds, and the time to validate the original data with the updated shapes graph is 0.024 seconds.

6.2.2 Yago Experiment

The Yago project [32] can be accessed at `https://yago-knowledge.org`. It contains entities of the real world, such as movies, cities and people, and it also contains relations between them. It covers beyond 50 million entities and 2 billion facts.

SHACL constraints have been used to generate the latest version of the knowledge base, called YAGO 4 [32]. We use these constraints for our experiments. This means that, unlike in the MeSH experiment, we have not created the shapes and constraints ourselves. The SHACL constraints of YAGO 4 define the disjointness of specific classes, and in addition, they define domains, ranges, and cardinalities of relations [32]. The shapes file that we downloaded contains 1322 statements in triple form and is available in our code repository.

Due to the size of the data set, we use SPARQL queries to access parts of it. Specifically, we query data about http://schema.org/Place. We perform two experiments for this data set, with a small change in the SPARQL queries: One with LIMIT 10 000 leading to 10 000 statements and one with LIMIT 100 000 leading to 99 995 statements. Both queries can be seen in Listing 6.9. We update the Yago data using the action from Listing 6.10, that corresponds to the formal action

$$\text{http://example.com/ns\#EntityWithPhysicalExtension} \oplus \text{http://schema.org/Place}.$$

The shape http://schema.org/Place is defined in the Yago shapes file in triple form.

Updating the 10 000 statements for the first Yago experiment led to a total of 13 971 statements and took 53 seconds. This means that the number of statements increased by 40%. Similar to the MeSH example, the time to transform the shapes graph is 0.01 seconds and the time to validate the original data with the updated shapes graph is 0.02 seconds.

In the second Yago experiment, updating the 99 995 original statements led to a total of 154 549 statements and took 1 hour and 37 minutes. Here, the number of statements increased by 55%. Like in the MeSH and first Yago experiment, the time to transform the shapes graph is roughly 0.01 seconds and the time to validate the original data with the updated shapes graph is 0.01 seconds.

6.2.3 Experiment Summary

Table 6.2 contains a summary of all experimental results. The columns denote the three experiments, where Yago 1 and 2 stand for the two different limits in the SPARQL query. The *Size of the data graph* is the number of triples in the data graph. *Time to update*

Table 6.2: Summary of experimental results

	MeSH	Yago 1	Yago 2
Size of the data graph	15 163 366	10 000	99 995
Time to update the data graph	7' 59s	53s	1h 37'
Time to transform shapes	0.019s	0.01s	0.01s
Time to validate	0.024s	0.02s	0.01s

the data graph is the time that is required to evaluate ϕ_c for actions of the form A $\oplus$ ϕ_c and A $\ominus$ ϕ_c respectively r for actions of the form p $\oplus$ r and p $\ominus$ r and to insert & remove the corresponding nodes from the graph. It does not consider loading and parsing the data graph or the actions. *Time to transform shapes* is the time that is required to insert and remove triples in the shapes graph in the way that corresponds to the formal transformation definition. Finally, *Time to validate* is the time for the Apache Jena validator to validate the updated shapes graph with the original data graph.

6.3 Discussion

We continue with the interpretation of these experimental results. The action is the same for both Yago experiments. By comparing their time to update the data graph, we see that the number of nodes in the graph plays an important role for the updates.

Independently of the different times to update the data graph between the experiments, the combined time to transform & validate the shapes graph is significantly smaller than updating the data graph within each experiment. This makes sense, as the size of the shapes graph is considerably smaller than the size of the data graphs in our experiments.

It follows that updating just the shapes graph or rather transforming the shapes and then performing the validation can massively reduce the total required time. Our goal was to avoid updating and re-validating the data graph. This shows that our approach is useful in those situations.

Listing 6.3: Two shapes with one class each

```
ex:TestShape
    a sh:NodeShape ;
    sh:class ex:A .

ex:ConceptShape
    a sh:NodeShape ;
    sh:class ex:B .
```

Listing 6.4: Transformed version of Listing 6.2

```
ex:TestShape
    a sh:NodeShape ;
    sh:or
        (
            [
                sh:class ex:A ;
            ]
            [
                sh:class ex:B ;
            ]
        ) .

ex:ConceptShape
    a sh:NodeShape ;
    sh:class ex:B .
```

Listing 6.5: Shapes graph for $\geq_1 E.\phi_c$

```
ex:S
    a sh:NodeShape ;
    sh:property
        [
            sh:path [ sh:alternativePath ( ex:loves ex:likes ) ] ;
            sh:qualifiedValueShape [ sh:class ex:Person ] ;
            sh:qualifiedMinCount 1 ;
        ] .
```

Listing 6.6: Negation as constraint component

```
ex:S
    a sh:NodeShape ;
    sh:class ex:A ;
    sh:not
        [
            sh:class ex:B ;
        ] .
```

Listing 6.7: Action for MeSH data

```
+ http://example.com/ns#id http://example.com/ns#IdentifierShape
+ http://example.com/ns#see http://example.com/ns#SeeAlsoShape
+ http://example.com/ns#GeoDescriptor http://example.com/ns#GeographicalDescriptorShape
- http://id.nlm.nih.gov/mesh/vocab#GeographicalDescriptor http://example.com/ns#GeoDescriptorShape
```

Listing 6.8: Shapes for MeSH data

```
ex:hasSomeNewId sh:targetNode <http://id.nlm.nih.gov/mesh/2022/C000724148> .
ex:hasSomeNewId sh:targetNode <http://id.nlm.nih.gov/mesh/2022/C000724057> .
ex:hasIdC000724057 sh:targetNode <http://id.nlm.nih.gov/mesh/2022/C000724057> .


ex:hasDateCreated
    a sh:NodeShape ;
    sh:property
        [
            sh:path <http://id.nlm.nih.gov/mesh/vocab#dateCreated> ;
            sh:qualifiedValueShape [ a sh:NodeShape ] ;
            sh:qualifiedMinCount 1 ;
        ] ;
.

ex:hasSomeNewId
    a sh:NodeShape ;
    sh:property
        [
            sh:path ex:id ;
            sh:qualifiedValueShape [ a sh:NodeShape ] ;
            sh:qualifiedMinCount 1 ;
        ] ;
.

ex:hasIdC000724057
    a sh:NodeShape ;
    sh:property
        [
            sh:path <http://id.nlm.nih.gov/mesh/vocab#identifier> ;
            sh:qualifiedValueShape
                [
                    a sh:NodeShape ;
                    sh:datatype xsd:string ;
                ] ;
            sh:qualifiedMinCount 1 ;
        ] ;
.




ex:IdentifierShape
    a sh:PropertyShape ;
    sh:path <http://id.nlm.nih.gov/mesh/vocab#identifier> ;
.

ex:SeeAlsoShape
    a sh:PropertyShape ;
    sh:path <http://id.nlm.nih.gov/mesh/vocab#seeAlso> ;
.

ex:GeographicalDescriptorShape
    a sh:NodeShape ;
    sh:class <http://id.nlm.nih.gov/mesh/vocab#GeographicalDescriptor> ;
.

ex:GeoDescriptorShape
    a sh:NodeShape ;
    sh:class <http://example.com/ns#GeoDescriptor> ;
.
```

Listing 6.9: SPARQL queries for Yago data

```
DESCRIBE ?s WHERE {
    ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://schema.org/Place>
} LIMIT 10000

DESCRIBE ?s WHERE {
    ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://schema.org/Place>
} LIMIT 100000
```

Listing 6.10: Action for Yago data

```
http://example.com/ns#EntityWithPhysicalExtension + http://schema.org/Place
```

Listing 6.11: A SPARQL-based constraint component

```
ex:LanguageExampleShape
    a sh:NodeShape ;
    sh:targetClass ex:Country ;
    sh:sparql
        [
            a sh:SPARQLConstraint ;   # This triple is optional
            sh:message "Values are literals with German language tag." ;
            sh:select """
                        SELECT $this (<http://example.com/ns#germanLabel> AS ?path) ?value
                        WHERE {
                                $this <http://example.com/ns#germanLabel> ?value .
                                FILTER (!isLiteral(?value) || !langMatches(lang(?value), "de"))
                        }
                        """ ;
        ]
.
```

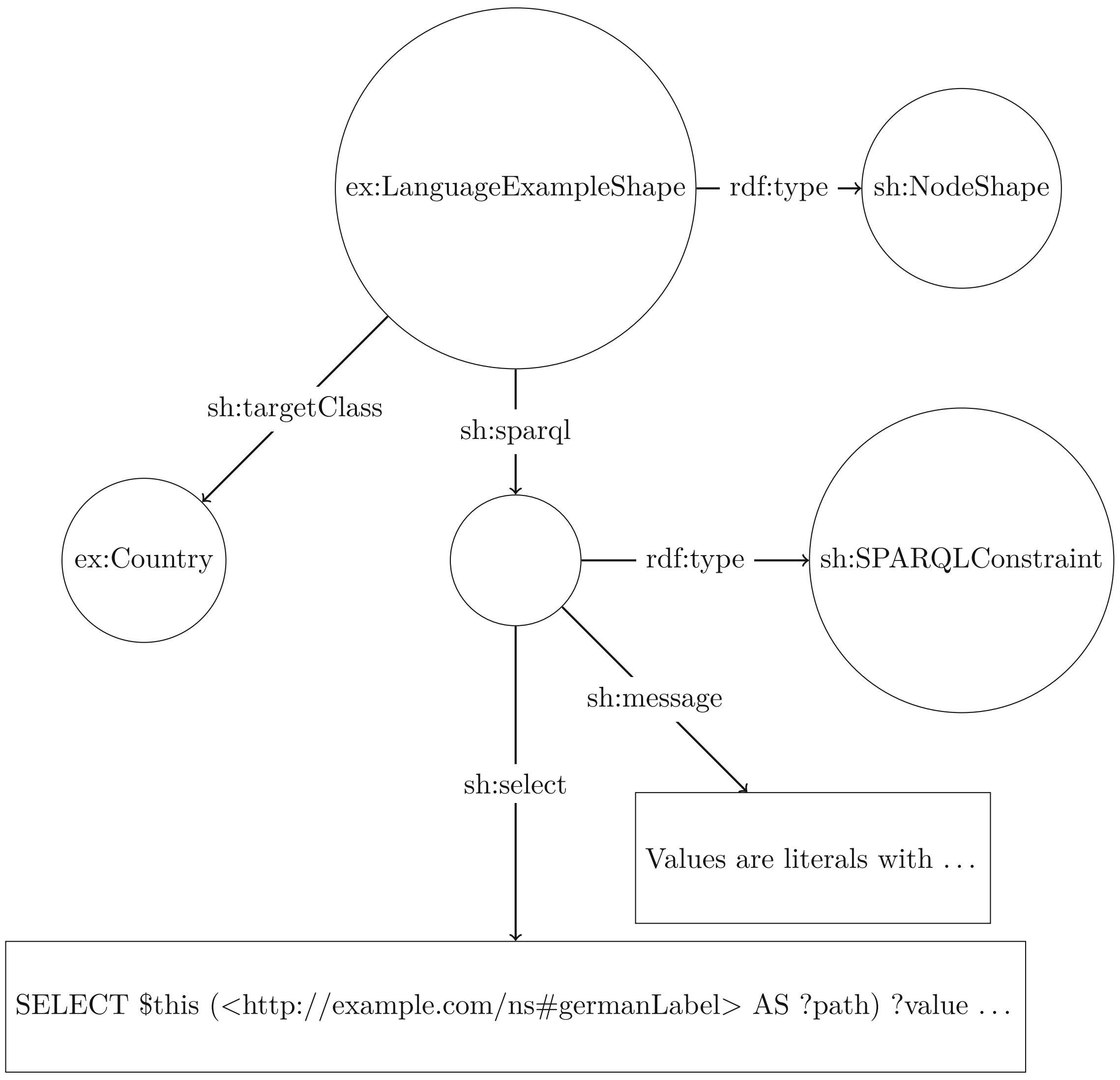


Figure 6.1: Shapes graph for a SPARQL-based constraint component

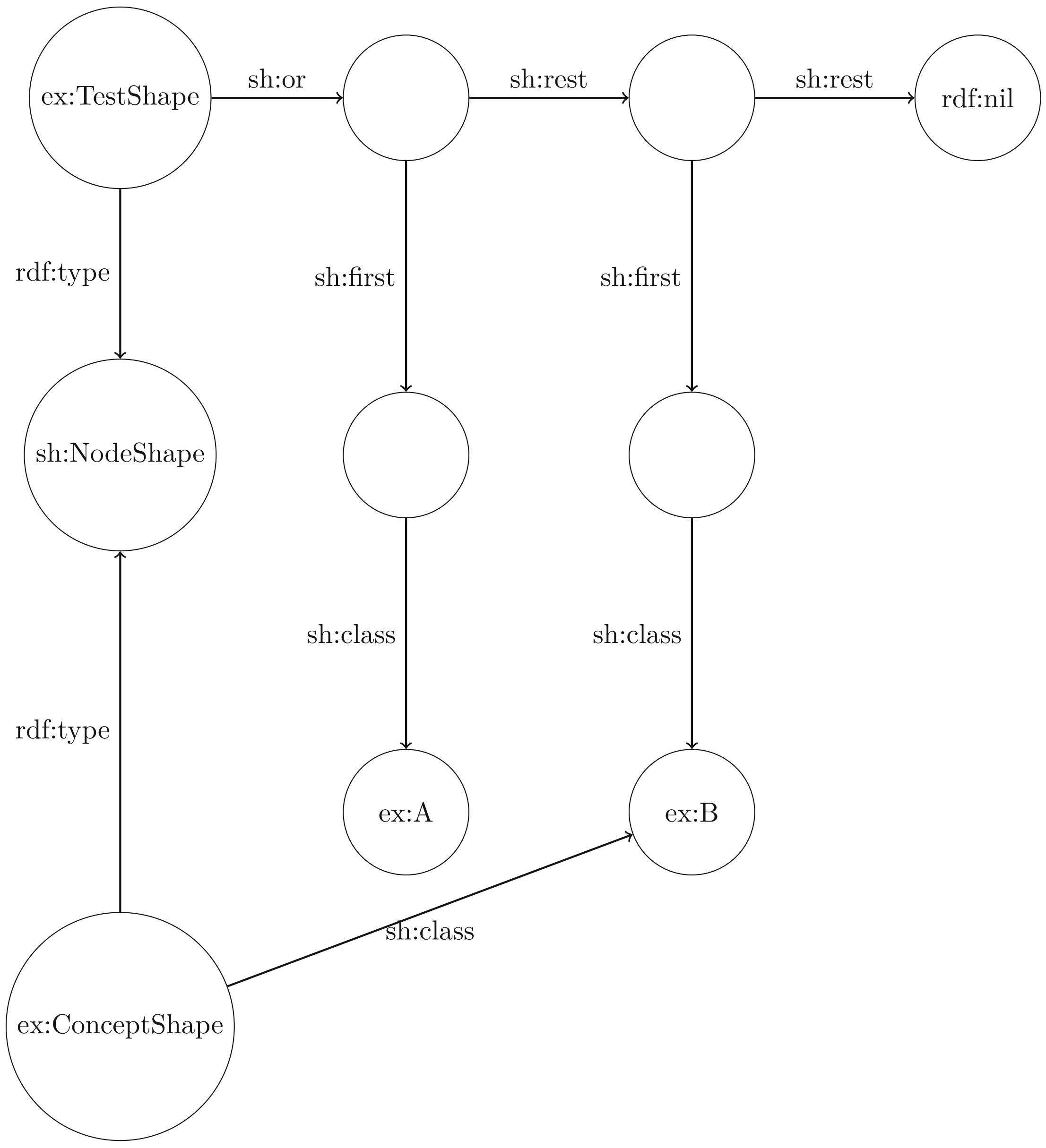


Figure 6.2: Shapes graph for sh:or

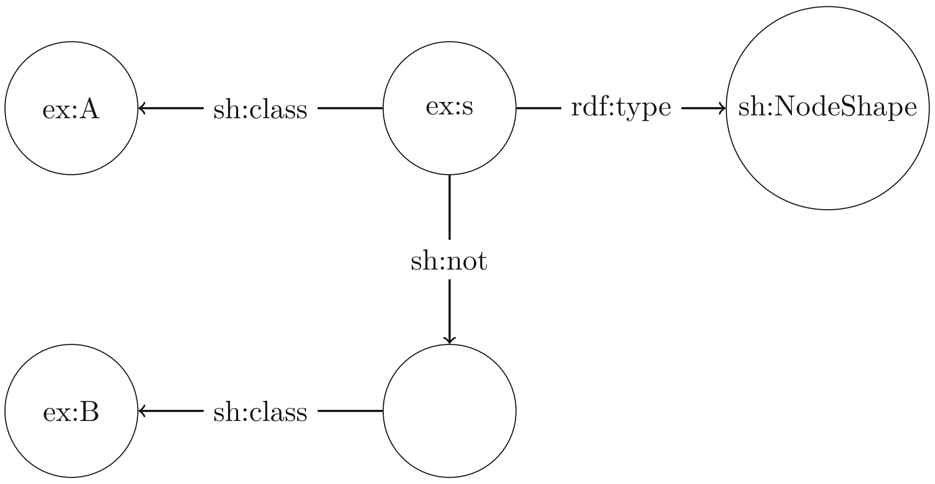


Figure 6.3: Shapes graph for sh:not

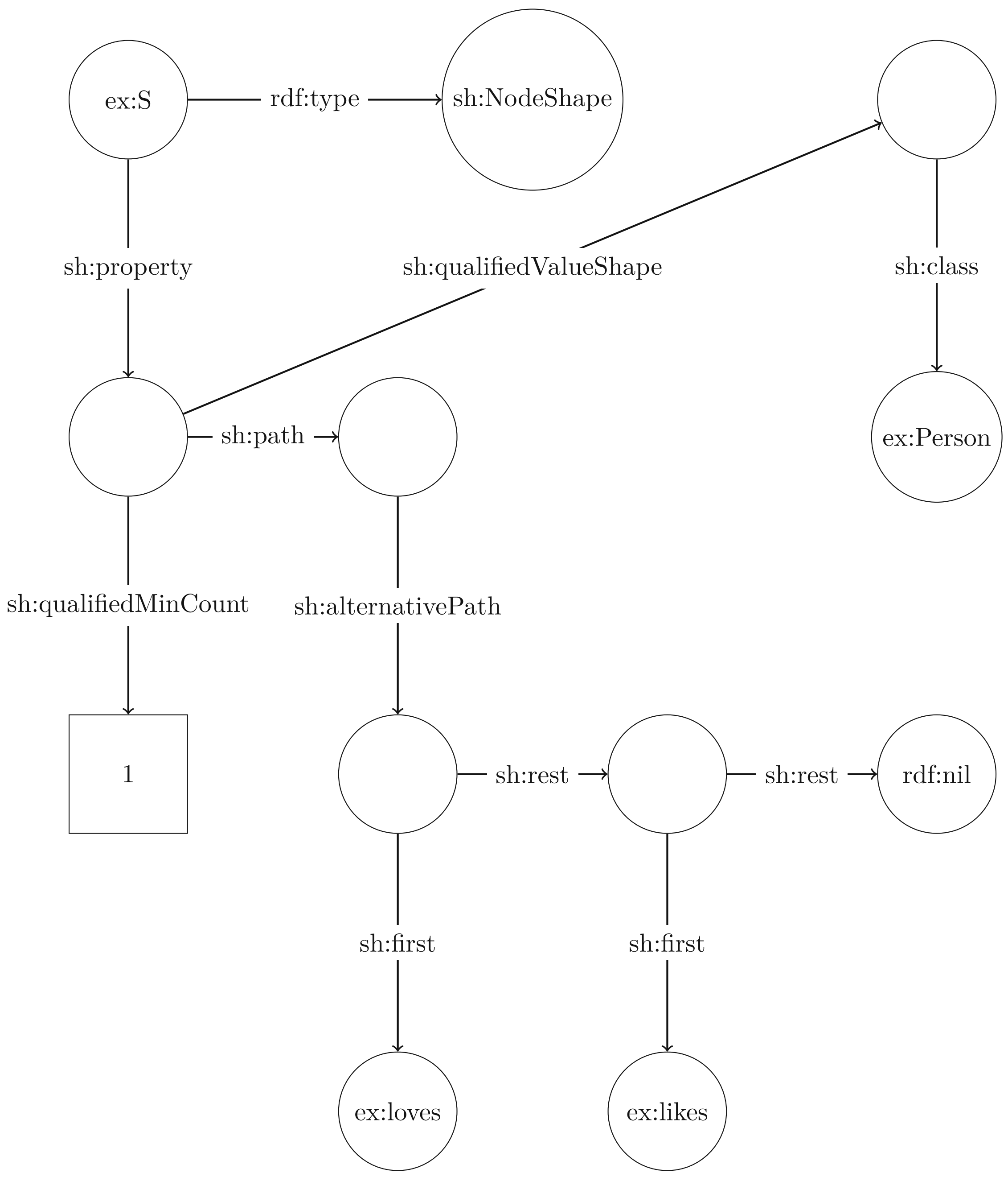


Figure 6.4: Shapes graph for sh:alternativePath and sh:qualifiedValueShape

CHAPTER 7

Summary

In this chapter, we summarize the results of the thesis and outline some possible avenues for future work.

Results

As SHACL is a relatively young language, many challenges and use-cases have not yet been addressed. In this thesis, we considered the effect of updates for SHACL validation of RDF graphs. In particular, we considered the situation that we have a validated graph and want to know if it remains valid when performing updates on the graph data. Having had performance and other considerations in mind, we wanted to find a way to verify that a set of updates yields a valid graph that avoids actually performing the updates on the graph. We called this *static validation under updates*.

The main goal of this thesis was to study the validation of SHACL constraints over RDF graphs that are subjects of updates.

In summary, the answers to the research questions are:

RQ1: How can we suitably describe updates on RDF graphs? Can we build a suitably expressive update language, building on proposals from the literature?

Inspired by an update language for description logics, we have defined syntax and semantics for an update language that can express modifications on RDF data graphs. This update language supports addition and removal of classes, as well as properties. In other words, it is able to add unary and binary atoms to the data graph. It is possible to concatenate multiple actions and to use variables.

RQ2: Is it possible to reduce validation under updates to validation over a static graph without performing the updates? Moreover, can such a validation be realized using existing validators?

The answer to this question depends on the allowed actions. Core SHACL as shown in the W3C recommendation and related papers is sufficient to capture most, but not all, actions from the update language. Therefore, it is **not** possible to reduce validation for all actions that we have defined to standard SHACL. This means in addition that existing validators that support SHACL Core can **not** cover all required cases for the static validation technique.

However, we have provided syntax and semantics for an extended SHACL formalization. In particular, this extension makes path expressions more expressible. Influenced by a transformation technique for description logics, we have shown a reduction that takes a set of constraints formulated in the extended SHACL formalization and transforms them into a different set of constraints. Using the extension, we can capture all defined actions. We have proved the correctness of the reduction in detail, which means that we showed that the original graph validates the transformed set of constraints if and only if the updated data graph validates the original constraints. In addition, we have demonstrated the transformation in multiple examples.

RQ3: Which features of SHACL can be supported in this setting that are not covered by existing formalizations, but that can increase the applicability of the approach?

We have extended shape expressions with predicates. As a useful feature for predicates, we have introduced literals which denote things like numbers, dates, or strings. We have provided multiple examples that demonstrate that this definition allows expressing interesting ideas.

RQ4: Can we gain experimental data that indicates the effectiveness of the static validation technique? Which additional artifacts do we need to get this data?

We have provided a proof-of-concept implementation based on the Apache Jena project. First, we implemented the transformation itself. Second, we also have an implementation that parses and applies the update language for data graphs. We can compare the transformation against this update implementation. In total, we have performed three experiments. Within each of these experiments, transforming the shapes and validating them on the original graph took much less time than updating the data graph. This shows, that the static validation technique can be very effective.

Future Work

The thesis shows up various questions that could be the topic of further research. In the following, we describe those that we suppose to be the most interesting.

The proposed extension of shape expressions with predicates is able to express various useful ideas. More research could be done into further refining this, especially considering the vacuous truth in Example 6.

We more thoroughly sketch an idea about using forms and one about conditional actions. Presently, it is possible to use SHACL and especially its *non-validating properties* to display forms [26]. We could also use forms to enable updates using SHACL. For example, we could use the form

Name of person:	$\boxed{\text{John}}$
Current supervisor:	William
New supervisor:	$\boxed{\text{Olivia}}$

for the action $\alpha = \text{supervises} \oplus (Olivia, John) \cdot \text{supervises} \ominus (William, John)$.

A different possible research direction is about more complex actions. It is desirable to extend the actions from Definition 1 with conditional actions. In this case, we would define complex actions as follows:

$$\alpha ::= \epsilon \mid (\beta \cdot \alpha) \mid (x \wedge \phi_c \,?\, \alpha[\![\alpha]\!]) \cdot \alpha$$

where β is a basic action, and ϵ denotes the *empty* action and the action precondition is a conjunction of a variable x and ϕ_c a non-ground concept.

This allows to perform updates such as the following two:

$$\begin{aligned} x \wedge \text{Human} \wedge (\text{age} > \mathit{18}) \,?\, \text{Adult} \oplus x[\![\text{Minor} \oplus x]\!] \\ x \wedge \text{Human} \wedge \exists \text{enrolledIn}.\, (y \wedge \text{University}) \,?\, \text{Student} \oplus x[\![\text{Student} \ominus x]\!] \end{aligned}$$

Defining the semantics for this, however, is a non-trivial task. In comparison to updates in a description logic context [2], we need to take locality into account because of the definition of validation with targets and shapes. We expect to need ideas from hybrid logic for this. One potential approach is as follows:

$$TR_{(a \wedge \phi_c \,?\, \alpha_1[\![\alpha_2]\!]) \cdot \alpha}(\phi) = (@_a \neg \phi_c \vee TR_{\alpha_1 \cdot \alpha}(\phi)) \wedge (@_a \phi_c \vee TR_{\alpha_2 \cdot \alpha}(\phi))$$

An extension of this idea is to allow whole constraints, including targets, as preconditions of actions.

Bibliography

[1] *About W3C*. W3C. 2021. URL: `https://www.w3.org/Consortium/` (visited on 08/04/2022).

[2] Shqiponja Ahmetaj et al. „Managing Change in Graph-Structured Data Using Description Logics“. In: *ACM Transactions on Computational Logic* 18.4 (Oct. 31, 2017), pp. 1–35. ISSN: 1529-3785, 1557-945X. DOI: `10.1145/3143803`. URL: `https://dl.acm.org/doi/10.1145/3143803` (visited on 06/28/2022).

[3] Shqiponja Ahmetaj et al. „Reasoning about Explanations for Non-validation in SHACL“. In: *Proceedings of the Eighteenth International Conference on Principles of Knowledge Representation and Reasoning*. 18th International Conference on Principles of Knowledge Representation and Reasoning {KR-2021}. Hanoii, Vietnam: International Joint Conferences on Artificial Intelligence Organization, Sept. 2021, pp. 12–21. ISBN: 978-1-956792-99-7. DOI: `10.24963/kr.2021/2`. URL: `https://proceedings.kr.org/2021/2` (visited on 06/28/2022).

[4] Medina Andresel et al. „Stable Model Semantics for Recursive SHACL“. In: *Proceedings of The Web Conference 2020*. WWW '20: The Web Conference 2020. Taipei Taiwan: ACM, Apr. 20, 2020, pp. 1570–1580. ISBN: 978-1-4503-7023-3. DOI: `10.1145/3366423.3380229`. URL: `https://dl.acm.org/doi/10.1145/3366423.3380229` (visited on 06/28/2022).

[5] *Apache Jena Repository*. GitHub. Apr. 17, 2023. URL: `https://github.com/apache/jena` (visited on 04/18/2023).

[6] *Apache Jena SHACL*. Apache Jena. Aug. 25, 2022. URL: `https://jena.apache.org/documentation/shacl/` (visited on 03/26/2023).

[7] Marcelo Arenas, Claudio Gutierrez, and Jorge Pérez. „Foundations of RDF Databases“. In: *Reasoning Web. Semantic Technologies for Information Systems: 5th International Summer School 2009, Brixen-Bressanone, Italy, August 30 - September 4, 2009, Tutorial Lectures*. Ed. by Sergio Tessaris et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2009, pp. 158–204. ISBN: 978-3-642-03754-2. DOI: `10.1007/978-3-642-03754-2_4`. URL: `https://doi.org/10.1007/978-3-642-03754-2_4` (visited on 04/21/2023).

[8] David Beckett et al. *RDF 1.1 Turtle.* W3C Recommendation. W3C, Feb. 25, 2014. URL: https://www.w3.org/TR/2014/REC-turtle-20140225/ (visited on 08/04/2022).

[9] Tim Berners-Lee, James Hendler, and Ora Lassila. „The Semantic Web". In: *Scientific American* 284.5 (May 2001), pp. 34–43. URL: http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21.

[10] Dan Brickley and R.V. Guha. *RDF Schema 1.1.* W3C Recommendation. W3C, Feb. 25, 2014. URL: https://www.w3.org/TR/2014/REC-rdf-schema-20140225/ (visited on 08/09/2022).

[11] Diego Calvanese, Magadalena Ortiz, and Mantas Simkus. „Evolving Graph Databases under Description Logic Constraints". In: vol. 1014. CEUR Workshop Proceedings. Aachen: RWTH, 2013, pp. 120–131.

[12] Diego Calvanese, Magdalena Ortiz, and Mantas Šimkus. „Verification of Evolving Graph-Structured Data under Expressive Path Constraints". In: *19th International Conference on Database Theory (ICDT 2016).* Ed. by Wim Martens and Thomas Zeume. Vol. 48. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016, 15:1–15:19. ISBN: 978-3-95977-002-6. DOI: 10.4230/LIPIcs.ICDT.2016.15. URL: http://drops.dagstuhl.de/opus/volltexte/2016/5784.

[13] Julien Corman. *SHACL2SPARQL Code Repository.* June 21, 2021. URL: https://github.com/rdfshapes/shacl-sparql (visited on 04/15/2023).

[14] Julien Corman, Juan L. Reutter, and Ognjen Savkovic. *Semantics and Validation of Recursive SHACL (Extended Version).* Bolzano: KRDB Research Centre, 2018.

[15] Julien Corman, Juan L. Reutter, and Ognjen Savković. „Semantics and Validation of Recursive SHACL". In: *The Semantic Web – ISWC 2018.* Ed. by Denny Vrandečić et al. Vol. 11136. Cham: Springer International Publishing, 2018, pp. 318–336. DOI: 10.1007/978-3-030-00671-6_19. URL: http://link.springer.com/10.1007/978-3-030-00671-6_19 (visited on 06/28/2022).

[16] Julien Corman et al. „Validating Shacl Constraints over a Sparql Endpoint". In: *The Semantic Web – ISWC 2019.* Ed. by Chiara Ghidini et al. Vol. 11778. Cham: Springer International Publishing, 2019, pp. 145–163. DOI: 10.1007/978-3-030-30793-6_9. URL: https://jreutter.sitios.ing.uc.cl/SHACL_19.pdf (visited on 07/07/2022).

[17] Richard Cyganiak, David Wood, and Markus Lanthaler. *RDF 1.1 Concepts and Abstract Syntax.* W3C Recommendation. W3C, Feb. 25, 2014. URL: https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/ (visited on 07/01/2022).

[18] *Download Medical Subject Headings Data.* National Library of Medicine. Feb. 17, 2023. URL: https://www.nlm.nih.gov/databases/download/mesh.html (visited on 03/31/2023).

[19] Lee Feigenbaum et al. *SPARQL 1.1 Protocol*. W3C. Mar. 21, 2013. URL: https://www.w3.org/TR/2013/REC-sparql11-protocol-20130321/ (visited on 04/16/2023).

[20] Mónica Figuera, Philipp D. Rohde, and Maria-Esther Vidal. „Trav-SHACL: Efficiently Validating Networks of SHACL Constraints". In: *Proceedings of the Web Conference 2021*. WWW '21: The Web Conference 2021. Ljubljana Slovenia: ACM, Apr. 19, 2021, pp. 3337–3348. ISBN: 978-1-4503-8312-7. DOI: 10.1145/3442381.3449877. URL: https://dl.acm.org/doi/10.1145/3442381.3449877 (visited on 04/16/2023).

[21] David S. Gunderson. „Variants of Finite Mathematical Induction". In: *Handbook of Mathematical Induction: Theory and Applications*. CRC Press, 2010. ISBN: 978-1-4200-9365-0. URL: https://www.routledgehandbooks.com/doi/10.1201/9781420093650-5.

[22] Steve Harris and Andy Seaborne. *SPARQL 1.1 Query Language*. W3C Recommendation. W3C, Mar. 21, 2013. URL: https://www.w3.org/TR/2013/REC-sparql11-query-20130321/ (visited on 07/31/2022).

[23] Patrick J. Hayes and Peter F. Patel-Schneider. *RDF 1.1 Semantics*. W3C Recommendation. W3C, Feb. 25, 2014. URL: http://www.w3.org/TR/2014/REC-rdf11-mt-20140225/ (visited on 07/01/2022).

[24] Holger Knublauch, Dean Allemang, and Simon Steyskal. *SHACL Advanced Features*. W3C Working Group. W3C, June 8, 2017. URL: https://www.w3.org/TR/2017/NOTE-shacl-af-20170608/ (visited on 07/01/2022).

[25] Holger Knublauch, Dean Allemang, and Simon Steyskal. *SHACL Advanced Features 1.1*. W3C Community Group Draft Report. SHACL Community Group, Feb. 12, 2021. URL: https://w3c.github.io/shacl/shacl-af/ (visited on 07/12/2022).

[26] Holger Knublauch and Dimitris Kontokostas. *Shapes Constraint Language (SHACL)*. W3C Recommendation. W3C, July 20, 2017. URL: https://www.w3.org/TR/2017/REC-shacl-20170720/ (visited on 07/01/2022).

[27] Jose Emilio Labra Gayo et al. *Validating RDF Data*. Vol. 7. Synthesis Lectures on the Semantic Web: Theory and Technology 1. Morgan & Claypool Publishers LLC, Sept. 2017. 328 pp. ISBN: 978-1-68173-164-3. DOI: 10.2200/s00786ed1v01y201707wbe016. URL: https://book.validatingrdf.com/ (visited on 07/01/2022).

[28] Martin Leinberger et al. „Deciding SHACL Shape Containment Through Description Logics Reasoning". In: *The Semantic Web – ISWC 2020*. Ed. by Jeff Z. Pan et al. Vol. 12506. Cham: Springer International Publishing, 2020, pp. 366–383. DOI: 10.1007/978-3-030-62419-4_21. URL: https://link.springer.com/10.1007/978-3-030-62419-4_21 (visited on 04/12/2023).

[29] Mathieu Lirzin and Holger Knublauch. *Expressing Equality between Multiple Paths #119*. GitHub. Oct. 7, 2019. URL: https://github.com/w3c/data-shapes/issues/119 (visited on 03/29/2023).

[30] Paolo Pareti, George Konstantinidis, and Fabio Mogavero. „Satisfiability and Containment of Recursive SHACL“. In: *Journal of Web Semantics* 74 (Oct. 2022), p. 100721. ISSN: 15708268. DOI: 10.1016/j.websem.2022.100721. URL: https://linkinghub.elsevier.com/retrieve/pii/S1570826822000130 (visited on 04/15/2023).

[31] Paolo Pareti et al. „SHACL Satisfiability and Containment“. In: *The Semantic Web – ISWC 2020*. Ed. by Jeff Z. Pan et al. Vol. 12506. Cham: Springer International Publishing, 2020, pp. 474–493. DOI: 10.1007/978-3-030-62419-4_27. URL: https://link.springer.com/10.1007/978-3-030-62419-4_27 (visited on 04/03/2023).

[32] Thomas Pellissier Tanon, Gerhard Weikum, and Fabian Suchanek. „YAGO 4: A Reason-able Knowledge Base“. In: *The Semantic Web*. Ed. by Andreas Harth et al. Vol. 12123. Cham: Springer International Publishing, 2020, pp. 583–596. DOI: 10.1007/978-3-030-49461-2_34. URL: http://link.springer.com/10.1007/978-3-030-49461-2_34 (visited on 03/31/2023).

[33] *Rdf-Validate-Shacl Code Repository*. GitHub. Apr. 4, 2023. URL: https://github.com/zazuko/rdf-validate-shacl (visited on 04/15/2023).

[34] Philipp D. Rohde and Figuera Mónica. *Trav-SHACL*. Scientific Data Management Group, Feb. 17, 2023. URL: https://github.com/SDM-TIB/Trav-SHACL (visited on 04/16/2023).

[35] Simon Steyskal and Karen Coyle. *SHACL Use Cases and Requirements*. W3C Working Group Note. W3C, July 20, 2017. URL: https://www.w3.org/TR/2017/NOTE-shacl-ucr-20170720/ (visited on 07/12/2022).

[36] *W3C Mission*. W3C. 2021. URL: https://www.w3.org/Consortium/mission#principles (visited on 08/04/2022).