TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science Institute of Theoretical Computer Science, Chair of Automata Theory

Implementation of an Engine for Answering Regular Path Queries under Approximate Semantics over $\mathcal{ELH}$ and $\mathcal{ELHI}_\bot$ knowledge bases

Tom Ziegler

Geboren am: 1. März 1997 in Plauen
Matrikelnummer: 4510666
Immatrikulationsjahr: 2015

Diplomarbeit

zur Erlangung des akademischen Grades

Diplom-Informatiker (Dipl.-Inf.)

Fachreferent
Prof. Dr.-Ing. Anni-Yasmin Turhan

Betreuender Hochschullehrer
Dr. Oliver Fernández Gil

Eingereicht am: 10. Januar 2025

Selbstständigkeitserklärung

Hiermit versichere ich, dass ich das vorliegende Dokument mit dem Titel *Implementation of an Engine for Answering Regular Path Queries under Approximate Semantics over $\mathcal{ELH}$ and $\mathcal{ELHI}_\bot$ knowledge bases* selbstständig und ohne unzulässige Hilfe Dritter verfasst habe. Es wurden keine anderen als die in diesem Dokument angegebenen Hilfsmittel und Quellen benutzt. Die wörtlichen und sinngemäß übernommenen Zitate habe ich als solche kenntlich gemacht. Es waren keine weiteren Personen an der geistigen Herstellung des vorliegenden Dokumentes beteiligt. Mir ist bekannt, dass die Nichteinhaltung dieser Erklärung zum nachträglichen Entzug des Hochschulabschlusses führen kann.

Dresden, 10. Januar 2025

Tom Ziegler

Tom Ziegler

Digital unterschrieben
von Tom Ziegler
Datum: 2025.01.10
19:53:04 +01'00'

Abstract

The $\mathcal{EL}$-family of Description Logics are a relatively well investigated field, in particular due to $\mathcal{EL}$ and some of its extensions allowing for polynomial reasoning. This makes them the logical choice for many applications where efficiency and scalability are required. In the biomedical domain, $\mathcal{EL}$ has been successfully employed for the development of several large ontologies. Answering Regular Path Queries (RPQs) has been investigated for many tractable and intractable extensions of $\mathcal{EL}$, and they have found their way into SPARQL under the name of *property paths*. On the other hand, approximate semantics for answering RPQs over Description Logic Knowledge Bases have only been proposed recently. In this work, we bridge the gap between theory and practice for answering 2RPQs under approximate semantics over both a tractable and an intractable member of the $\mathcal{EL}$-family. To this end, we have extended the notion of approximate semantics from $\mathcal{ELH}$ to $\mathcal{ELHI}_\bot$, and developed and implemented a practical procedure to answer 2RPQs under approximate semantics for both $\mathcal{ELH}$ and $\mathcal{ELHI}_\bot$ Knowledge Bases.

Contents

List of Tables

List of Algorithms

1. Introduction

In this thesis, we present a deterministic procedure to answer *Two-Way Regular Path Queries* under *approximate semantics* over $\mathcal{ELH}$ and $\mathcal{ELHI}_\perp$ *Description Logic Knowledges Bases*, and explain how the procedure has been implemented.

For many years, there has been hardly any area of application where the size and complexity of data needing to be stored and retrieved has not seen a constant increase. According to recent statistics, we are currently witnessing an exponential trend on the amount of data created, copied or consumed each year. Reasons for the increase might go well beyond just the availability of less expensive data storage devices, mobile phones and sensors. This global trend not only increases the overall amount of data needing to be handled but also, in many cases, increases the degree of connectivity between such data. Popular areas that feature a huge amount of highly connected data include social networks, biomedical applications, as well as many applications in the area commonly referred to as the *Internet of Things* (IoT).

To store and query such highly connected data, graph databases present themselves as the natural choice, and bring many advantages over relational or document-oriented databases. In the biomedical domain, many applications of graph databases can be found [TRM21].

Regular path queries (RPQs) provide a versatile way to retrieve data from graph databases. RPQs and their extensions are a part of SPARQL, an official recommendation by the W3C to query RDF data. They are used to find database individuals that are connected by paths of a certain pattern. To this end, they define a regular language, and the paths connecting individuals must be labeled by a word within the language.

The field of *Ontology-Mediated Query Answering* (OMQA) further expands the versatility of querying graph databases by extending the facts stored within a database with implicit consequences obtained from the data. The character of such consequences is highly domain-dependent and is provided by *ontologies*. The Web Ontology Language (OWL) provides a standardized way to formulate, use and exchange such ontologies in the context of the *semantic web*.

However, even in the presence of an ontology, a good awareness of the application domain is still required. In particular, a RPQ might yield no answers over a given data source, but, given expert domain knowledge, there might be several answers that are "close enough" for them to be included in the results. Even if a RPQ does yield answers, including such closely related answers might better capture a users need for information.

In [FT21], *approximate semantics* for answering 2RPQs over *Description Logic Knowledge Bases* are proposed. Intuitively, this is used to capture how closely related objects within the application domain are. Such (dis-)similarity is measured by a numeric value called *approximation cost*. A very intuitive example of such an approximation is to use the *word edit distance* to capture lexicographic similarity of terms within the query and the database.

However, approximate semantics allow more than just capturing related answers. While, in many cases, it might be desirable to preserve classical answers, i.e. exact matches of the query within the database, this is not necessarily true for all applications. One could, for example, disencourage the usage of certain information within a database that is expected to be outdated or less significant, by including an appropriate approximation cost to retrieve such results. This cost could even be changed for a different use case, all without the need to modify the underlying database content.

In this thesis, we have developed and implemented a method for answering 2RPQs under approximate semantics over Knowledge Bases formulated in the Description Logic (DL) $\mathcal{ELH}$ or $\mathcal{ELHI}_\bot$. The theoretical underpinning is obtained from [FT21], and is based on reducing the problem to finding shortest paths in a graph. This is done via the help of two relations sp and spa. We show that a similar approach to the be problem can be taken for the DL $\mathcal{ELHI}_\bot$ by providing a uniform notation for approximate semantics for both DLs. We then extend the procedure to construct sp and spa into a practical algorithm that contains several optimizations while maintaining matching upper bounds. This algorithm forms the backbone of our implementation. To the best of our knowledge, the implementation obtained during this thesis is the first one to address this problem.

1.1. Organization of the thesis

Chapter 2 acts as an introduction to the topic and is used to declare basic terms and definitions. Chapter 3 introduces *approximate semantics* and explains the foundation of the answering process used in our implementation. Chapter 4 constitutes the main result of this thesis by presenting an optimized algorithm used to calculate the relations sp and spa. Finally, Chapter 5 is used to explain the details of our implementation.

The appendix consists of three parts: Appendix A contains additional pseudocode and proofs used in chapter 4. Appendix B and C contain a condensed and slightly revised version of results obtained during preliminary work on this topic, done during my graduate courses at TU Dresden.

2. Preliminaries

In this chapter, we introduce the basic terms and notations used in the subsequent chapters. We start with a short introduction to Description Logics.

2.1. Description Logics $\mathcal{ELH}$ and $\mathcal{ELHI}_\bot$

In this section, we give a short introduction to the Description Logics used in this report. Our main focus is on $\mathcal{ELHI}_\bot$, an extension of the DL $\mathcal{EL}$. In general, Description Logics are used to provide a logical representation of a certain domain of interest. To this end, a specific *DL vocabulary* is used. By N_C, N_R and N_I, we denote three countable, disjunct sets of symbols as follows:

- N_C denotes the set of concept names,
- N_R denotes the set of role names, and
- N_I denotes the set of individual names.

To represent the knowledge about our domain, a DL vocabulary is used to build two different kinds of expressions:

- *Terminological axioms* are used to describe the general properties of *concepts* and *roles*. These constraints apply to *all* objects in the domain. *Concept Inclusions* are used to model relations between (sets of) concepts, and *role inclusion* describe relations between different roles.
- *Assertions* state facts about specific objects in the domain. They ensure that specific individuals participate in some concept, or that two individuals are connected by some role.

To explain how these *axioms* look, we first need to introduce *concept descriptions*. In $\mathcal{EL}$, *(complex) concept description* are obtained using the constructors *conjunction* ($A \sqcap B$), *existential restriction* ($\exists r.A$) and *top* ($\top$). The set of all $\mathcal{EL}$ concept descriptions is build inductively from N_C and N_R according to the following syntactic rule:

$$C ::= \top \mid A \mid C \sqcap C \mid \exists r.C$$

$\mathcal{EL}_\bot$ is the extension of $\mathcal{EL}$ where we additionally allow the *bottom* ($\bot$) constructor. We can futher extend $\mathcal{EL}_\bot$ to $\mathcal{ELI}_\bot$ by additionally allowing *inverse roles* (r^-) to be used in existential restrictions. Complex concept descriptions in $\mathcal{ELI}_\bot$ are then build according to the rule:

$$C ::= \top \mid \bot \mid A \mid C \sqcap C \mid \exists r.C \mid \exists r^-.C$$

By allowing another type of axioms, called *role inclusions*, we can further extends $\mathcal{EL}$ and $\mathcal{ELI}_\bot$to obtain $\mathcal{ELH}$ and $\mathcal{ELHI}_\bot$, respectively. *Terminological axioms* for $\mathcal{ELH}$ and $\mathcal{ELHI}_\bot$ can then be of two types:

- *General concept inclusions* (GCIs) are of the form $C \sqsubseteq D$, where C and D are (complex) concept descriptions
- *role inclusions* are of the form $r \sqsubseteq s$. Here, r and s are role names from $\mathsf{N_R}$

Assertions are of the form $A(a)$ (concept assertion) or $r(a, b)$ (role assertion). Here, A is a concept name, r is a role name, and a, b are individual names.

A finite set of terminological axioms is called a $\mathcal{T}$*-Box*, and a finite set of assertions is called an $\mathcal{A}$*-Box*. Together, they form a description logic *Knowledge Base* $\mathcal{K} = (\mathcal{T}, \mathcal{A})$. By $\mathsf{N_R^-} := \{r^- \mid r \in \mathsf{N_R}\}$, we denote the set of inverse role names, and by $\mathsf{N_R^\pm} := \mathsf{N_R} \cup \mathsf{N_R^-}$ the set of role names including their inverse. We will use $|\mathcal{T}|$ to denote the amount of axioms in $\mathcal{T}$, and $|\mathcal{A}|$ to denote the amount of axioms in $\mathcal{A}$. Consequently, we use $|\mathcal{K}|$ to denote $|\mathcal{T}| + |\mathcal{A}|$. The *Signature* of a $\mathcal{T}$-Box contains all concept names and role names occurring in $\mathcal{T}$, and is denoted as $Sig(\mathcal{T})$. The set of all individuals occuring in an $\mathcal{A}$ is denoted as $\mathsf{Ind}(\mathcal{A})$.

$\mathcal{T}$-Box axioms	$\mathcal{EL}$	$\mathcal{EL}_\bot$	$\mathcal{ELH}$	$\mathcal{ELHI}_\bot$
$A_1 \sqcap ... A_n \sqsubseteq B$	✓	✓	✓	✓
$A_1 \sqcap ... A_n \sqsubseteq \bot$		✓		✓
$r \sqsubseteq s$			✓	✓
$A \sqsubseteq \exists r.B$	$r \in N_R$	$r \in N_R$	$r \in N_R$	$r \in \mathsf{N_R^\pm}$
$\exists r.B \sqsubseteq A$	$r \in N_R$	$r \in N_R$	$r \in N_R$	$r \in \mathsf{N_R^\pm}$

Table 2.1.: Syntax for *normalized* $\mathcal{T}$-Boxes for different DLs from the $\mathcal{EL}$-Family. $A_1, ..., A_n, B$ denote atomic concepts from $\mathsf{N_C} \cup \top$, and $r, s \in \mathsf{N_R^\pm}$. Ticks (✓) indicates that axioms of this form are allowed in a normalized $\mathcal{T}$-Box. A condition $r \in \mathsf{N_R}$ denotes that axioms of this form are allowed if the condition is satisfied.

Semantics for $\mathcal{ELH}$ and $\mathcal{ELHI}_\bot$

We assume that $\mathcal{ELH}$ and $\mathcal{ELHI}_\bot$ $\mathcal{T}$-Boxes are given in a *normal form* that only allows axioms of a certain structure. Table 2.1 summarizes the syntactic rules allowed for the presented DLs using this normal form. This is w.l.o.g., as any $\mathcal{EL}$ or $\mathcal{ELHI}_\bot$ $\mathfrak{T}$-Box can be efficiently transformed into normal form by introducing fresh concept names [BBL05].

The semantics of DLs are obtained from *interpretations*. Such an interpretation $\mathcal{I}$ is a pair $(\Delta^I, \cdot^I)$, where

- $\Delta^\mathcal{I}$ is a non-empty set called the interpretation domain, and
- $\cdot^\mathcal{I}$ is the interpretation function that assigns the elements in $\Delta^\mathcal{I}$ to the concept names, role names and individual names present in a DL KB.

Such an interpretation naturally forms a graph-like structure, where the nodes are the elements from $\Delta^{\mathcal{I}}$, and the edges are obtained from the interpretation function. We use the notion of *paths* to describe how the elements in $\Delta^{\mathcal{I}}$ are connected:

Definition 1. Let $d, d' \in \Delta^{\mathcal{I}}$. A *path* π from d to d' in $\mathcal{I}$ is a sequence $d_0 u_1 d_1 u_2 d_2 ... u_m d_m$ such that $m \geq 0, d_0 = d, d_m = d'$ and for all $1 \leq j \leq m$:

- $d_j \in \Delta^{\mathcal{I}}$ and $u_j \in \mathsf{N}_\mathsf{R}^{\pm} \cup \{A? \mid A \in \mathsf{N}_\mathsf{C}\}$,
- $u_j = A?$ implies $d_{j-1} = d_j$ and $d_j \in A^{\mathcal{I}}$, and
- $u_j \in \mathsf{N}_\mathsf{R}^{\pm}$ implies $(d_{j-1}, d_j) \in u_j^{\mathcal{I}}$

We define the *label* of a path π as $l(\pi) := u_1 ... u_m$, and use $d \xrightarrow{\mathcal{I},u} d'$ to denote that there exists a path from d to d' in $\mathcal{I}$ with label u.

Given an interpretation $\mathcal{I}$, the semantics of the DL concept and role constructors, $\mathcal{T}$-Box axioms and $\mathcal{A}$-Box assertions that are used in this thesis are given in Table 2.2. We say that a $\mathcal{T}$-Box axiom or $\mathcal{A}$-Box assertion is *satisfied* by $\mathcal{I}$ if the inclusion given in tale 2.2 holds. Further, we say that $\mathcal{I}$ is a *model* of a $\mathcal{T}$-Box $\mathcal{T}$ if all axioms in $\mathfrak{T}$ are satisfied using $\mathcal{I}$, and accordingly that $\mathcal{I}$ is a *model* of an $\mathcal{A}$-Box $\mathcal{A}$ if all assertions in $\mathcal{A}$ are satisfied.

For a KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, we use $\mathcal{I} \models \mathcal{K}$ to denote that an interpretation $\mathcal{I}$ is a *model* of $\mathcal{K}$, i.e. it is a model of both $\mathcal{T}$ and $\mathcal{A}$. If there is at least one such model, we say that $\mathcal{K}$ is *satisfiable*.

Finally, we introduce a notion to say that a $\mathcal{T}$-Box axiom or $\mathcal{A}$-Box assertion α is satisfied for all models $\mathcal{I}$ of $\mathcal{K}$. This is denoted as $\mathcal{K} \models \alpha$, and we say that $\mathcal{K}$ *entails* α.

Name	Syntax	Semantics
Top concept	$\top$	$\Delta^{\mathcal{I}}$
Bottom concept	$\bot$	$\varnothing$
Nominal	$\{a\}$	$\{a^{\mathcal{I}}\}$
Conjunction	$C_1 \sqcap C_2$	${C_1}^{\mathcal{I}} \cap {C_2}^{\mathcal{I}}$
Existential restriction	$\exists r.C$	$\{d_1 \mid \text{there exists } (d_1, d_2) \in r^{\mathcal{I}} \text{ with } d_2 \in C^{\mathcal{I}}\}$
Inverse role	r^-	$\{(d_2, d_1) \mid (d_1, d_2) \in r^{\mathcal{I}}\}$
Concept inclusion	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
Role inclusion	$r \sqsubseteq s$	$r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$
Concept assertion	$A(a)$	$a^{\mathcal{I}} \in A^{\mathcal{I}}$
Role assertion	$r(a, b)$	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$

Table 2.2.: Syntax and Semantics for $\mathcal{ELH}$ and $\mathcal{ELHI}_\bot$ concept and role constructors, $\mathcal{T}$-Box axioms and $\mathcal{A}$-Box assertions. A denotes a concept name, C_1, C_2 denote (possibly complex) concept descriptions, r denotes a role name, and a, b are individual names.

Complexity of Reasoning in $\mathcal{ELH}$ and $\mathcal{ELHI}_\bot$

Here, we will give a brief overview of the basic reasoning tasks over DL KBs which are used in this report. A detailed discussion of the complexity of reasoning in $\mathcal{EL}$ and its extensions can be found in [BBL05] and [BBL08].

Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be an $\mathcal{ELH}$ or $\mathcal{ELHI}_\bot$ KB. The *subsumption* problem asks, given two (complex) concept descriptions C, D, whether it holds that $\mathcal{K} \models C \sqsubseteq D$. In that case, we say that C *is subsumed by* D. A related problem is to *calculate all subsumees*, i.e. given a (complex) concept description C, to calculate the set of all concept names $A \in \mathsf{N_C}$ s.t. $\mathcal{K} \models A \sqsubseteq C$.

Analogously, the *role subsumption* problem asks, given two roles $r, s \in \mathsf{N_R}$ ($r, s \in \mathsf{N_R^\pm}$ for $\mathcal{ELHI}_\bot$), whether it holds that $\mathcal{K} \models r \sqsubseteq s$.

Finally, an *instance check* asks, for a concept $A \in \mathsf{N_C}$ and an individual $a \in \mathsf{Ind}(\mathcal{A})$, whether it holds that $\mathcal{K} \models A(a)$.

To shorten notation, we additionally allow sets of concept names to be used in the presence of $\mathcal{ELHI}_\bot$ KBs. Given two sets of concept names $M, M' \subseteq \mathsf{N_C}$, we say that $\mathcal{K} \models M(a)$ iff $\mathcal{K} \models A(a)$ for all $A \in M$. Let $M = \{A_1, ...A_m\}$ and $M' = \{B_1, ..., B_n\}$. We write $\mathcal{K} \models M \sqsubseteq M'$ iff $\mathcal{K} \models A_1 \sqcap ... \sqcap A_m \sqsubseteq B_1 \sqcap ... \sqcap B_n$.

For some extensions of $\mathcal{EL}$, such as $\mathcal{ELH}$ or $\mathcal{EL}_\bot$, the additional features do not impact the complexity of these reasoning tasks. However, allowing inverse roles induces a significant increase in complexity. Note that answering *subsumption* over a DL KB is independent of the content of $\mathcal{A}$. In $\mathcal{ELH}$, answering *subsumption* is feasible in polynomial time, but is EXP-time complete for $\mathcal{ELHI}_\bot$ [BBL08]. *Instance checking* in $\mathcal{ELH}$ is P-complete for both data and combined complexity, whereas combined complexity increases to EXP-time for $\mathcal{ELHI}_\bot$ ([BO15]).

2.2. Two-way regular path queries

In the OMQA setting, *Two-way regular path querys* (2RPQs) are used to find pairs of individuals connected by a certain chain of *roles*. They are defined using *regular languages*, given by the means of a *regular expression* or an *non-deterministic finite automaton (NFA)*. In this work, we will exclusively be using the NFA representation.

Formally, a 2RPQ q is of the form $q(x, y) = \mathfrak{R}(x, y)$, where $x, y \in \mathsf{N_I}$ and $\mathfrak{R} = (Q_\mathfrak{R}, \Sigma, \delta_\mathfrak{R}, I_\mathfrak{R}, F_\mathfrak{R})$ is an NFA representing a regular language. The alphabet Σ is a finite subset of $\mathsf{N_R^\pm} \cup \{A? \mid A \in \mathsf{N_C}\}$. We use $\mathcal{L}(\mathfrak{R})$ to denote the regular language represented by $\mathfrak{R}$.

Regular path queries (RPQs) are a simpler, more restricted form of 2RPQs that do not use inverse roles, i.e. Σ is a finite subset of $\mathsf{N_R} \cup \{A? \mid A \in \mathsf{N_C}\}$. The more complex *Conjunctive Two-way Regular Path Queries* (C2RPQs) are not considered in this report.

2.3. Certain answers and universal model

Given a satisfiable $\mathcal{ELHI}_\bot$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, we denote the *universal model* of $\mathcal{K}$ as $\mathcal{U}_\mathcal{K} = (\Delta^{\mathcal{U}_\mathcal{K}}, \cdot^{\mathcal{U}_\mathcal{K}})$. We will use the definition of the universal model for an $\mathcal{ELHI}_\bot$ KB from [BO15]:

The domain $\Delta^{\mathcal{U}_\mathcal{K}}$ consists of sequences of the form $aR_1M_1...R_nM_n (n \geq 0)$, where $a \in \mathsf{Ind}(A)$, and for every $i \geq 1$, $R_i \in \mathsf{N}_\mathsf{R}^\pm$ and M_i is a conjunction of concepts from $\mathsf{N}_\mathsf{C} \cup \{\top\}$. Formally, $\Delta^{\mathcal{U}_\mathcal{K}}$ consists of all sequences $aR_1M_1...R_nM_n$ that satisfy:

- If $n \geq 1$, then $\mathcal{T} \models M_0 \sqsubseteq \exists R_1.M_1$ where $M_0 = \{A \in \mathsf{N}_\mathsf{C} \cup \top \mid K \models A(a)\}$ and there does not exist $M_1' \supsetneq M_1$ such that $\mathcal{T} \models M_0 \sqsubseteq \exists R_1.M_1'$
- For every $1 \leq i < n, \mathcal{T} \models M_i \sqsubseteq \exists R_{i+1}.M_{i+1}$ and there is no $M_{i+1}' \supsetneq M_{i+1}$ such that $\mathcal{T} \models M_i \sqsubseteq \exists R_{i+1}.M_{i+1}'$.

It remains to fix the interpretation $\cdot^{\mathcal{U}_\mathcal{K}}$ of the individual names, concept names, and role names from K. This is done as follows:

$$
\begin{aligned}
a^{\mathcal{U}_\mathcal{K}} &= a \text{ for all } a \in \mathsf{Ind}(A),\\
A^{\mathcal{U}_\mathcal{K}} &= \{a \in \mathsf{Ind}(\mathcal{A}) \mid K \models A(a)\} \cup\\
&\quad \{e \in \Delta^{\mathcal{U}_\mathcal{K}} \setminus \mathsf{Ind}(\mathcal{A}) \mid e = e'RM \text{ and } A \in M\}, \text{ and}\\
r^{\mathcal{U}_\mathcal{K}} &= \{(a,b) \mid K \models r(a,b)\} \cup\\
&\quad \{(e_1,e_2) \mid e_2 = e_1SM \text{ and } \mathcal{T} \models S \sqsubseteq r\} \cup\\
&\quad \{(e_2,e_1) \mid e_2 = e_1SM \text{ and } \mathcal{T} \models S \sqsubseteq r^-\}.
\end{aligned}
$$

We note that $\Delta^{\mathcal{U}_\mathcal{K}}$ consists of two types of elements: The elements from $\mathcal{A}$, referred to as *individuals*, and additional elements whose existence is implied by the axioms in $\mathcal{T}$. We will refer to the latter as *anonymous elements*. Such elements are of the form $e = aR_1M_1...R_nM_n (a \in \mathsf{Ind}(\mathcal{A}))$. The existence of such an $e \in \Delta^{\mathcal{U}_\mathcal{K}}$ implies the existence of an element of the form $aR_1M_1...R_{n-1}M_{n-1} \in \Delta^{\mathcal{U}_\mathcal{K}}$, and all elements of the form $aR_1M_1...aR_mM_M$ together create a tree-like structure rooted at a. Given $e = aR_1M_1...R_nM_n \in \Delta^{\mathcal{U}_\mathcal{K}}$, we use T_e to denote the sub-tree rooted at e, and we denote the final component of such a sequence as $tail(e) = M_n$.

An important property obtained from the definition of the universal model is that the *tail* of an element uniquely identifies everything that happens "below", i.e. if we have two elements $e, e' \in \Delta^{\mathcal{U}_\mathcal{K}}$ and $tail(e) = tail(e')$, the subtrees T_e and T_e' rooted at these elements are isomorphic [BO15]. We use $\mathsf{T}(\mathcal{U}_\mathcal{K})$ to denote the set containing all tails present in $\mathcal{U}_\mathcal{K}$, i.e. $\mathsf{T}(\mathcal{U}_\mathcal{K}) = \{tail(e) \mid e \in \Delta^{\mathcal{U}_\mathcal{K}}\}$. For $\mathcal{ELHI}_\bot$, the elements in $\mathsf{T}(\mathcal{U}_\mathcal{K})$ are conjunctions of concept names, and the maximum amount of such tails is $|\mathcal{P}(\mathsf{Sig}(\mathcal{T}))| - 1$[1]. To simplify their presentation, we will slightly abuse notation and treat such conjunctions as sets of concept names[2].

For $\mathcal{ELH}$ KBs, we adopt the definition of the universal model from [BOŠ13]. Here, the domain $\Delta^{\mathcal{U}_\mathcal{K}}$ consists of sequences of the form $aR_1C_1...R_nC_n$, and we denote the final concept of such as a sequence as $tail(e) = C_n$. Both constructions are closely related, and $tail(e)$ carries over its main property, which is to characterize the anonymous subtree rooted at a given element. Hence, for an $\mathcal{ELH}$ KB, we have for two elements e, e': If $tail(e) = tail(e')$, then T_e and $T_{e'}$ are isomorphic [BOŠ13]. Accordingly, we adjust the definition of $\mathsf{T}(\mathcal{U}_\mathcal{K})$ to match the different domain: For an $\mathcal{ELH}$ KB $\mathcal{K}$, we denote as $\mathsf{T}(\mathcal{U}_\mathcal{K}) = \{tail(e) \mid e \in \Delta^{\mathcal{U}_\mathcal{K}}\}$ the set of tails present in $\mathcal{U}_\mathcal{K}$. Here, the elements in $\mathsf{T}(\mathcal{U}_\mathcal{K})$ are concept names from N_C, and the amount of entries in $\mathsf{T}(\mathcal{U}_\mathcal{K})$ is limited by the size of $\mathsf{Sig}(\mathcal{T})$.

[1] Note that the empty set is not contained in $\mathsf{T}(\mathcal{U}_\mathcal{K})$.

[2] This allows us to write $A \in M$ to say that A is one of the conjuncts in M, and $M' \subseteq M$ to say that every conjunct in M' also is a conjunct of M.

Comparing both definitions, we note that the presence of inverse roles changes the domain of the universal model: While, for an $\mathcal{ELH}$ KB, the anonymous individuals can be characterized by a sequence of role transitions and concept names, an $\mathcal{ELHI}_\bot$ KB requires a sequence of role transitions and conjunctions of concept names. While the size of $\mathsf{T}(\mathcal{U}_\mathcal{K})$ is bounded by the size of $\mathcal{T}$ for $\mathcal{ELH}$, the size of $\mathsf{T}(\mathcal{U}_\mathcal{K})$ is exponential in the size of $\mathcal{T}$ under the presence of an $\mathcal{ELHI}_\bot$ KB.

As every DL KB allows for multiple (possibly infinitely many) models, we are only interested in answers that hold in all models of a given KB. These answers are referred to as *certain answers*. The crucial property of $\mathcal{U}_\mathcal{K}$ is that is 'contained' in every model of $\mathcal{K}$, i.e. for each model $\mathcal{I}$ of $\mathcal{K}$, there exists a homomorphism from $\mathcal{U}_\mathcal{K}$ to $\mathcal{I}$.

It follows that, for a query q, answers in the universal model of $\mathcal{K}$ are answers in every model of $\mathcal{K}$. The answers to q in $\mathcal{U}_\mathcal{K}$ then coincide with the set of *certain answers* for q for both $\mathcal{ELH}$ and $\mathcal{ELHI}_\bot$ KBs [BOŠ13], [BO15].

3. Answering Approximate 2RPQs over $\mathcal{ELH}$ and $\mathcal{ELHI}_\bot$ KBs

In this chapter, we introduce *approximate semantics* for $\mathcal{ELH}$ and $\mathcal{ELHI}_\bot$. We then proceed to explain how the problem of answering 2RPQs under approximate semantics can be reduced to finding shortest paths in a graph.

3.1. Approximate semantics for 2RPQs over $\mathcal{ELH}$ and $\mathcal{ELHI}_\bot$ KBs

This section presents a notion for approximate semantics for 2RPQs[1] over $\mathcal{ELH}$ and $\mathcal{ELHI}_\bot$ KBs as presented in [FT21], and explains the relevant terms and definitions. The approximate semantics presented here can be seen as an extension of the approximate semantics for RPQs over graph databases proposed in [GT05], which uses a particular form of *weighted finite-state transducers* (WFTs) called *distortion transducers*:

Definition 2. ([GT05])
A *distortion transducer* (dT) is an NFA defined as a tuple $\mathfrak{T} = (Q, \Sigma, \delta, I, F)$, where:

- Q is a finite set of states,
- Σ denotes a finite alphabet,
- $I, F \subseteq Q$ are the sets of initial and final states, respectively, and
- The transition relation δ is a subset of $Q \times \Sigma \times \Sigma \times \mathbb{N} \times Q$.[2]

For a given dT $\mathfrak{T} = (\Sigma, Q, \delta, I, F)$, a run of $\mathfrak{T}$ on a word $u \in \Sigma^*$ is a sequence of tuples

$$\rho = (q_1, u_1, v_1, w_1, q_2), ..., (q_n, u_n, v_n, w_n, q_{n+1})$$

such that $u = u_1 \dots u_n$, $q_1 \in I$, $q_{n+1} \in F$, and each $(q_i, u_i, v_i, w_i, q_{i+1}) \in \delta$ for $i \leq n$.

[1] The definitions presented here rely on the results shown in [FT21] and Appendix C, where C2RPQs are used. As 2RPQs are a simpler form of C2RPQs, this applies w.l.o.g. to 2RPQs.

[2] To simplify notation, we assume that $\mathfrak{T}$ contains no ϵ-transitions. This assumption can be made without loss of generality, as demonstrated in [FT21].

The *weight* of the run ρ is denoted as $wt(\rho) := w_1 + \cdots + w_n$. A run ρ transforms u into $v = v_1 \dots v_n$ with cost $wt(\rho)$. Let $\mathcal{R}(\mathfrak{T}, u, v)$ be the set of all pairs $(\rho, wt(\rho))$ such that ρ is a run of $\mathfrak{T}$ transforming u into v. The cost of transforming u into v via $\mathfrak{T}$ is defined as

$$c_{\mathfrak{T}}(u, v) := \min\{wt(\rho) \mid (\rho, wt(\rho)) \in R(\mathfrak{T}, u, v)\}.$$

[FT21] extends the approximate semantics from [GT05] to the OMQA by using DL interpretations. Here, an answer to a 2RPQ over a DL KB under approximate semantics is called an *approximate answer* and is defined as follows:

Definition 3. (Definition 5, [FT21]): Let $q(x, y)$ be a 2RPQ. The set of *approximate answers* of q in an interpretation $\mathcal{I}$, through a dt $\mathfrak{T}$, is defined as

$$\tilde{\mathsf{ans}}_{\mathfrak{T}}(q, \mathcal{I}) := \Big\{(d, e, \eta_{d,e}) \mid d, e \in \Delta^{\mathcal{I}} \text{ and } \eta_{d,e} = \min\{c_{\mathfrak{T}}(u, v) \mid u \in \mathcal{L}(\mathfrak{R}), v \in \Sigma^*, d \xrightarrow{I,v} e\}\Big\}.$$

In the OMQA setting, our goal is to find answers that exist in every model of our KB. Such an answer is called a *certain approximate answer*, and the approximation cost corresponds to the most costly approximate answer for any model of $\mathcal{K}$.

Definition 4. Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ an $\mathcal{ELH}$ or $\mathcal{ELHI}_\bot$ KB and $q(x, y)$ a 2RPQ. The set of *certain approximate answers* of q w.r.t. $\mathcal{K}$, trough a dt $\mathfrak{T}$, is defined as

$$\tilde{\mathsf{cert}}_{\mathfrak{T}}(q, \mathcal{K}) := \Big\{(a, b, \eta_{a,b}) \mid a, b \in \mathsf{Ind}(\mathcal{A}) \text{ and } \eta_{a,b} = \sup_{\mathcal{I} \models \mathcal{K}} \{\eta_{d,e} \mid (d, e, \eta_{\bar{d}}) \in \tilde{\mathsf{ans}}_{\mathfrak{T}}(q, \mathcal{I}) \wedge (d, e) = (a^{\mathcal{I}}, b^{I})\}\Big\}.$$

Similar to answering 2RPQs under classical semantics, we can use the universal model to characterize the set $\tilde{\mathsf{cert}}_{\mathfrak{T}}$. As shown in [FT21] (for $\mathcal{ELH}$) and Appendix C (for $\mathcal{ELHI}_\bot$), the following holds:

Proposition 1. *Let $K = (\mathcal{T}, \mathcal{A})$ be an $\mathcal{ELH}$ or $\mathcal{ELHI}_\bot$ KB, $q(x, y)$ a 2RPQ, and $\mathfrak{T}$ a distortion transducer. Then, $(a, b, \eta_{a,b}) \in \tilde{\mathsf{cert}}_{\mathfrak{T}}(q, \mathcal{K})$ iff $(a, b, \eta_{a,b})$ is an approximate answer of q in $\mathcal{U}_{\mathcal{K}}$.*

3.2. Answering 2RPQs by finding shortest paths

The procedure to calculate the approximation cost of all elements $(a, b, \eta_{a,b} \in \tilde{\mathsf{cert}}_{\mathfrak{T}}$ presented in [FT21] uses a weighted graph $G_{\mathcal{U}_{\mathcal{K}}}$ obtained from the *Cartesian product* of the query NFA, the transducer NFA, and the universal model $\mathcal{U}_{\mathcal{K}}$. It is shown that the existence and minimal cost of paths in this graph can be used to characterize $\tilde{\mathsf{cert}}_{\mathfrak{T}}$. We start by explaining how the graph $G_{\mathcal{U}_{\mathcal{K}}}$ is constructed.

Definition 5. (Definition 12, [FT21]) Let $\mathcal{I}$ be an interpretation, $\mathfrak{R}(x, y)$ be a 2RPQ with $\mathfrak{R} = (Q_{\mathfrak{R}}, \Sigma, \delta_{\mathfrak{R}}, I_{\mathfrak{R}}, F_{\mathfrak{R}})$ and $\mathfrak{T}$ a dt with $\mathfrak{T} = (Q_{\mathfrak{T}}, \Sigma, \delta_{\mathfrak{T}}, I_{\mathfrak{T}}, F_{\mathfrak{T}})$. The weighted graph $G_{\mathfrak{R} \times \mathfrak{T} \times \mathcal{I}} = (V, E)$ is defined as:

- $V := Q_{\mathfrak{R}} \times Q_{\mathfrak{T}} \times \mathcal{I}$
- $E \subseteq V \times \mathbb{N} \times V$ is a set of weighted edges such that $((s, t, d), w, (s', t', d')) \in E$ iff its components satisfy one of the following conditions:
  - $(s, \sigma, s') \in \delta_{\mathfrak{R}}$, $(t, \sigma, \sigma', w, t') \in \delta_{\mathfrak{T}}$ and $d \xrightarrow{\mathcal{I}, \sigma'} d'$, or
  - $s = s'$, $(t, \epsilon, \sigma, w, t') \in \delta_{\mathfrak{T}}$ and $d \xrightarrow{\mathcal{I}, \sigma} d'$.

When $\mathfrak{R}$ and $\mathfrak{T}$ are clear from the context, we use $G_{\mathcal{U}_\mathcal{K}}$ to denote the graph $G_{\mathfrak{R}\times\mathfrak{T}\times\mathcal{U}_\mathcal{K}}$.

Then, given $a, b \in \mathsf{Ind}(\mathcal{A})$, calculating $\eta_{a,b}$ s.t. $(a, b, \eta_{a,b} \in \tilde{\mathsf{cert}}_\mathfrak{T}(q, \mathcal{K})$ amounts to finding the cost of a shortest path from a node (s, t, a) to (s', t', b) in $G_{\mathcal{U}_\mathcal{K}}$ s.t. $s \in I_\mathfrak{R}$, $s' \in F_\mathfrak{R}$, $t \in I_\mathfrak{T}$ and $t' \in F_\mathfrak{T}$. However, as the amount of elements in $\Delta^{\mathcal{U}_\mathcal{K}}$ can possibly be infinite, in general, it is not possible to construct $G_{\mathcal{U}_\mathcal{K}}$ directly. Instead, a finite graph $G^*_{\mathcal{U}_\mathcal{A}}$ is used to find shortest paths of equivalent cost. We will explain how this graph is constructed.

Let $G_{\mathcal{U}_\mathcal{A}}$ be the finite subgraph of $G_{\mathcal{U}_\mathcal{K}}$ restricted to nodes (s, t, a) s.t. a is an $\mathcal{A}$-Box individual. As the amount of individuals is limited by the size of $\mathcal{A}$, the amount of nodes in $G_{\mathcal{U}_\mathcal{A}}$ is $|Q_\mathfrak{R}| \cdot |Q_\mathfrak{T}| \cdot |\mathsf{Ind}(\mathcal{A})|$. Now, for all nodes (s, t, a) in $G_{\mathcal{U}_\mathcal{A}}$, assume the following: If there is a (possibly infinite) path from (s, t, a) to some (s', t', a) in $G_{\mathcal{U}_\mathcal{K}}$, and w^* is the minimal cost of such a path, add an edge $(s, t, a), w^*, (s', t', a)$ to $G_{\mathcal{U}_\mathcal{A}}$. This way, we obtain a graph $G^*_{\mathcal{U}_\mathcal{A}}$. Then, the following holds, for $\mathcal{K}$ either an $\mathcal{ELH}$ or $\mathcal{ELHI}_\bot$ KB. The corresponding proofs are found in [FT21] and AppendixC.

Proposition 2. *(Proposition 17, [FT21])*
Let $G^*_{\mathcal{U}_\mathcal{A}}$ be the extension of $G_{\mathcal{U}_\mathcal{A}}$ with all the edges $((s, t, a), c^*, (s', t', a))$ such that: c^* is the minimal cost if an a-path[3] from (s, t, a) to (s', t', a) in $G_{\mathcal{U}_\mathcal{K}}$.

Then, given two vertices (s, t, a) and (s', t', b), the minimal cost of a path from (s, t, a) to (s', t', b) is the same in $G_{\mathcal{U}_\mathcal{K}}$ and $G^*_{\mathcal{U}_\mathcal{A}}$.

For the construction of $G^*_{\mathcal{U}_\mathcal{A}}$, it is required to know which paths of minimal cost exist from (s, t, a) to (s', t', a) in $G_{\mathcal{U}_\mathcal{K}}$. This is achieved with the help of two relations sp and spa, referred to as *loop tables*. Entries in sp are of the form $sp[(s, t), (s', t'), D]$[4], where $(s, t), (s', t') \in \delta_\mathfrak{R} \times \delta_\mathfrak{T}$, and $D \in N_C$. Each such entry is associated with a value $v \in \mathbb{N} \cup \infty$ and represents the following information: Let $a \in \mathsf{Ind}(\mathcal{A})$ $G_{\mathcal{U}_\mathcal{K}}$. If $\mathcal{K} \models D(a)$, $G_{\mathcal{U}_\mathcal{K}}$ contains a path from (s, t, a) to (s', t', a) with minimal cost v (or, if $v = \infty$, there is no such path).

To construct sp, an additional relation spa is used. Entries in spa are of the form $spa[(s, t), (s', t'), C] = w$ and represent the following information: Let $e \in \Delta^{\mathcal{U}_\mathcal{K}} \setminus \mathsf{Ind}(\mathcal{A})$ be an anonymous individual in $\mathcal{U}_\mathcal{K}$. If $tail(e) = C$, there exists an e-path of minimal cost from (s, t, e) to (s', t', e) in $G_{\mathcal{U}_\mathcal{K}}$ with cost w.

The idea behind the construction of spa is that $tail(e)$ can be used to uniquely identify the subtree of $\mathcal{U}_\mathcal{K}$ rooted at e. Consequently, for two nodes (s, t, e) and (s, t, e') in $G_{\mathcal{U}_\mathcal{K}}$ with $tail(e) = tail(e')$, for each e-path of minimal cost from (s, t, e) to (s', t', e), a path of similar cost can be found from (s, t, e') to (s', t', e').

For $\mathcal{ELH}$, the tables sp and spa can be seen as an extension of the relations *Loop* and *ALoop* presented for answering 2RPQs under classical semantic in [BOŠ13]. For $\mathcal{ELHI}_\bot$, a related construction using only a single table *Loop* for answering 2RPQs under classical semantics is presented in [BO15].

A theoretical procedure to construct spa in the presence of an $\mathcal{ELH}$ KB can be found in [FT21]. A corresponding procedure exists for $\mathcal{ELHI}_\bot$ KBs, as shown in Appendix B. For $\mathcal{ELH}$, this procedure requires at most polynomial time in the size of $\mathfrak{R}, \mathfrak{T}$ and $\mathcal{K}$. For $\mathcal{ELHI}_\bot$, the

[3] We use the definition of a-path (resp. e-path) from [FT21]. Intuitively speaking, an a-path is a path that never visits any $e \in \Delta^{\mathcal{U}_\mathcal{K}}$ outside the subtree rooted at a.
[4] We use a slightly different form for $\mathcal{ELHI}_\bot$, as explained in section 4.5

complexity increases to EXP-time in the combined size of $\mathfrak{R}, \mathfrak{T}$ and $\mathcal{K}$. Although the procedure can be regarded as a deterministic algorithm, it lacks any consideration regarding optimal performance, and a straightforward implementation would be an inefficient approach to this problem.

In the following chapter, we give a detailed explanation on how the tables sp and spa can be constructed for both $\mathcal{ELH}$ and $\mathcal{ELHI}_\bot$ using an explicit, deterministic procedure containing several efficiency optimizations while maintaining matching upper bounds for combined complexity. The resulting algorithm represents the main result of this work, and acts as the core component of our implementation.

3.3. Reasoning problems under approximate semantics

To conclude this chapter, we introduce the reasoning problems that can be answered by our implementation. The answering process is described in section 5.4.

Given an $\mathcal{ELH}$ or $\mathcal{ELHI}_\bot$ KB $\mathcal{K}$, a 2RPQ $\mathfrak{R}$ and a dt $\mathfrak{T}$, we define the following reasoning problems:

- *τ-entailment*: Given $(a, b) \in \mathsf{Ind}(\mathcal{A})$ and a threshold value μ, decide whether $(a, b, \eta_{a,b}) \in \check{\mathsf{cert}}_{\mathfrak{T}}(q, \mathcal{K})$ for some $\eta_{a,b} \leq \mu$.
- *Cost computation*: Given (a, b), calculate the approximation cost $\eta_{a,b}$ s.t. $(a, b, \eta_{a,b}) \in \check{\mathsf{cert}}_{\mathfrak{T}}(q, \mathcal{K})$.
- *Calculate* $\check{\mathsf{cert}}_{\mathfrak{T}}(q, \mathcal{K})$: Calculate the set of all approximate answers.
- *Calculate* $\check{\mathsf{cert}}^{\mu}_{\mathfrak{T}}(q, \mathcal{K})$: Given a threshold value μ, calculate the set of approximate answers with an approximation cost of at most μ, i.e. the set containing all $(a, b, \eta_{a,b}) \in \check{\mathsf{cert}}_{\mathfrak{T}}(q, \mathcal{K})$ with $\eta_{a,b} \leq \mu\}$.

4. Deterministic loop table construction for $\mathcal{ELH}$ and $\mathcal{ELHI}_\bot$

Constructing the loop tables represents the main computational challenge when answering approximate 2RPQs over DL KBs. In this chapter, we present two deterministic algorithms to compute the tables spa and sp, for both $\mathcal{ELH}$ and $\mathcal{ELHI}_\bot$.

In general, the complexity results presented in [FT21] already provide valuable insights into how the different computation steps impact the overall effort to compute these relations. While it is relatively easy to construct best-case and worst-case scenarios, there is a large area in between for which it is not trivial to determine how effortful the loop table construction procedure is going to be by purely looking at the input data.

Therefore, we will use the term *moderate-case* scenario to refer to any situation that represents neither a best-case nor worst-case scenario, thus including, but not restricting what can be considered an average-case scenario, i.e. a "typical input". Note that we use the terms best-/worst-/moderate-case scenarios only to refer to the inherent properties of the input data, independent of the the input size. Moreover, we stick to a theoretical point of view where we do not assess whether our input data has any real-world meaning or is optimal amongst all equivalent representations.

The development of the algorithms presented in this chapter has been conducted with focus on the actual performance of our implementation given a limited amount of test input data. To this end, an internal profiler, tracking computation time and memory consumption on a per-code-line base, was used. This allowed us to evaluate the impact each part of the computation step has on the overall performance and to identify parts that act as bottlenecks for the entire computation.

Based on these insight, we have developed a set of optimized algorithms that aim to provide good performance over reasonably large inputs for moderate-case scenarios. Note that in this context, we use the term optimizations only to refer to computational steps included in the formal algorithm, as opposed to implementation-specific optimizations, which are discussed in chapter 5.
In the following, we will present our optimized algorithms for constructing spa and sp under the presence of either an $\mathcal{ELH}$ or $\mathcal{ELHI}_\bot$ KB.

4.1. General approach

In general, optimizations included in these algorithms fall into one of two categories: Filtering and Caching. However, there is no free lunch: Both types of optimizations introduce some form of overhead not accounted for in the procedure presented in [FT21]. The idea behind these constructions is that they can be used to skip other, computationally more expensive operations - most notably DL reasoning tasks such as answering subsumption for complex concept descriptions.

- **Filtering:** By filtering, we refer to constructions that use the deterministic behaviour of the algorithm to eliminate parts of the input data that are guaranteed to have no impact on the results. These filtering procedures are lightweight computations, usually only introduce a minimal overhead in terms of computation time and the amount of memory used. In this sense, these optimizations have very little impact for worst-case scenarios, but might provide significant improvements for other scenarios.

- **Caching:** Caching optimizations are such that store intermediate data during computation that otherwise could be omitted. These constructions usually introduce a significant, but predictable amount of overhead memory consumption alongside some computation effort required to store and retrieve these intermediate results. Such constructions significantly reduce performance for worst-case scenarios, but provide improved performance for all other scenarios.

The following sections give a detailed explanation of the different parts of the algorithms for both $\mathcal{ELH}$ and $\mathcal{ELHI}_\bot$. For both scenarios, we each have to compute 3 different types of construction rules, referred to as rules S1, S2 and S3.

A correct procedure to construct spa over $\mathcal{ELH}$ KBs is obtained from [FT21] and acts as a baseline for our algorithm. The same procedure can be used for $\mathcal{ELHI}_\bot$ KBs when accounting for the difference in $\mathsf{T}(\mathcal{U}_\mathcal{K})$ and a redefinition of the rules S1, S2 and S3. A detailed explanation can be obtained from preliminary work on this topic, which is included in this thesis as Appendix B.

This allows us to state a single procedure which applies to both $\mathcal{ELH}$ and $\mathcal{ELHI}_\bot$ KBs. The definition of rules S1, S2 and S3 obtained from [FT21] and Appendix B are provided in sections 4.2 and 4.3, respectively. The procedure uses $\mathsf{T}(\mathcal{U}_\mathcal{K}) \subseteq \mathsf{Sig}(\mathcal{T})$ for $\mathcal{ELH}$ KBs, and $\mathsf{T}(\mathcal{U}_\mathcal{K}) \subseteq \mathcal{P}(\mathsf{Sig}(\mathcal{T})) \setminus \{\}$ for $\mathcal{ELHI}_\bot$ KBs.

Procedure SPA

1: Initialize $spa[\mathfrak{p}, \mathfrak{q}, C] = \infty$ (or 0 if $\mathfrak{p} = \mathfrak{q}$)
2: Apply rule S2 to all $(\mathfrak{p}, \mathfrak{q}, C) \in (Q_\mathfrak{R} \times Q_\mathfrak{T})^2 \times \mathsf{T}(\mathcal{U}_\mathcal{K})$
3: Apply rule S3 until spa does not change;
4: **repeat**
5: $\quad spa := f(spa)$
6: **until** spa does not change

function f
$\quad spa^* := spa$
$\quad$ Update spa by applying rule S1 to all $(\mathfrak{p}, \mathfrak{q}, C) \in (Q_\mathfrak{R} \times Q_\mathfrak{T})^2 \times \mathsf{T}(\mathcal{U}_\mathcal{K})$ using spa^*
$\quad$ Apply S3 until spa does not change;
end function

In general, *Procedure SPA* represents an iterative process that extends the information currently stored in the table spa with each iteration of f, until the content of spa can no longer change. This is the case as soon as an iteration of f produced no updates. As shown in [FT21], this is guaranteed to be the case after a maximum of $(|Q_\mathfrak{R}| \cdot |Q_\mathfrak{T}|)^2 \cdot |\mathsf{T}(\mathcal{U}_\mathcal{K})|$ iterations. It is notable that the values in spa can only decrease during the computation, and thus entries of the form $spa[\mathfrak{p}, \mathfrak{p}, C]$ have a fixed value 0.

4.2. Rule Calculation over $\mathcal{ELH}$ KBs

To show how spa can be constructed deterministically under the presence of an $\mathcal{ELH}$ KB, this section introduces procedures to calculate applications of rules S1,S2 and S3. The procedures presented here are optimized for moderate-case scenarios, but match the upper bounds obtained from [FT21] for combined complexity.

Recall that, for an $\mathcal{ELH}$ KB, the elements in $\mathsf{T}(\mathcal{U}_\mathcal{K})$ are basic concepts from $\mathsf{N_C}$, and the size of $\mathsf{T}(\mathcal{U}_\mathcal{K})$ is polynomial in the number of concept names in $\mathcal{K}$.

A short remark on notation used in this section is required: All procedures presented in this section assume the presence of an $\mathcal{ELH}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, a 2RPQ $\mathfrak{R}$ and a distortion Transducer $\mathfrak{T}$. To shorten notation, we will not explicitly state these as input.

We start by giving the definition of rules S1, S2 and S3 in the presence of an $\mathcal{ELH}$ KB as obtained from [FT21], with $R, R', R'' \in \mathsf{N_R}$ and $A, C, D \in \mathsf{N_C}$:

S1. $spa[(s,t),(s',t'),C] \ll w_1 + spa[(s_1,t_1),(s_2,t_2),A] + w_2$, if $C1^*$ holds.

S2. $spa[(s,t),(s',t'),C] \ll w$, if $\mathcal{T} \models C \sqsubseteq A$, $(s,u,s') \in \delta_\mathfrak{R}$, and $(t,u,A?,w,t') \in \delta_\mathfrak{T}$,

S3. $spa[(s,t),(s',t'),C] \ll spa[(s,t),(s'',t''),C] + spa[(s'',t''),(s',t'),C]$.

and $C1^*$:

$C1^*$. $\mathcal{T} \models D \sqsubseteq \exists R.A$, $\mathcal{T} \models R \sqsubseteq R'$, $\mathcal{T} \models R \sqsubseteq R''$, $(s,u,s_1) \in \delta_\mathfrak{R}$, $(t,u,R',w_1,t_1) \in \delta_\mathfrak{T}$, $(s_2,u',s') \in \delta_\mathfrak{R}$ and $(t_2,u',R''^-,w_2,t') \in \delta_\mathfrak{T}$.

4.2.1. Calculating S1 over $\mathcal{ELH}$ KBs

Here, we present a procedure to compute the result of applying rule S1 to all $(\mathfrak{p},\mathfrak{q},C) \in (Q_\mathfrak{R} \times Q_\mathfrak{T})^2 \times \mathsf{T}(\mathcal{U}_\mathcal{K})$, as required for one iteration of f. We note that the order in which these rule applications are computed is arbitrary.

The deterministic procedure consists of 3 major stages: 2 filtering stages followed by one computational stage. Subsequent stages rely on the results of the previous one, but are otherwise independent of each other. However, only the final stage depends on the actual values stored in the table spa. Thus, one could use additional memory to store the results of the second stage to avoid repeating the calculation. However, we expect to obtain better results by storing the role hierarchy instead, as discussed in section *INSERT REFERENCE*. The third stage, however, needs to be computed for each iteration of f. We discuss each stage individually before presenting the entire algorithm.

4.2.1.1. Stage 1: Extracting transducer edges

The first stage represents a **filtering optimization** on the input query and transducer automaton. The input to this stage is a tuple $((s,t),(s_1,t_1),(s_2,t_2),(s',t')) \in (Q_\mathfrak{R} \times Q_\mathfrak{T})^4$. Its goal is to extract such edges $(t,u,R',w_1,t_1) \in \delta_\mathfrak{T}$ and $(t_2,u',R''^-,w_2,t') \in \delta_\mathfrak{T}$ for which corresponding edges $(s,u,s_1) \in \delta_\mathfrak{R}$ and $(s_2,u,s') \in \delta_\mathfrak{R}$, respectively, exist. To ensure that condition $C1^*$ can be satisfied, we further require $R',R'' \in \mathsf{N_R}$. Finally, these edges are sorted by their cost w in ascending order. It is notable that this stage only uses information from the query and transducer automatons.

Procedure 1 *filterEdges*

Input: $(s,t),(s_1,t_1),(s_2,t_2),(s',t') \in (Q_\mathfrak{R} \times Q_\mathfrak{T})$
Output: Two sorted sets $down \subseteq \delta_\mathfrak{T}$ and $up \subseteq \delta_\mathfrak{T}$

1: set $down := \{(t,u,R',w_1,t_1) \mid (s,u,s_1) \in \delta_\mathfrak{R}, (t,u,R',w_1,t_1) \in \delta_\mathfrak{T}, R' \in \mathsf{N_R}\}$
2: set $up := \quad \{(t_2,u',R''^-,w_2,t') \mid (s_2,u',s') \in \delta_\mathfrak{R}, (t_2,u,R''^-,w_2,t') \in \delta_\mathfrak{T}, R'' \in \mathsf{N_R}\}$
3: sort $down$ and up by cost in ascending order

Lemma 1. *Let $\mathfrak{R}$ a 2RPQ, $\mathfrak{T}$ a distortion transducer, and $(s,t),(s_1,t_1),(s_2,t_2),(s',t') \in Q_\mathfrak{R} \times Q_\mathfrak{T}{}^4$. After* Procedure filterEdges *finishes, it holds that $(t,u,R',w_1,t_1) \in down$ iff there is $(s,u,s_1) \in \delta_\mathfrak{R}$; and $(t_2,u',R''^-,w_2,t') \in up$ iff there is $(s_2,u',s') \in \delta_\mathfrak{R}$.*

Proof. This is a direct consequence from lines 2 and 3. □

4.2.1.2. Stage 2: Extracting feasible roles

The second stage uses the results obtained from the first stage to apply a filtering optimization on the role names present in the ontology. As opposed to the first stage, this stage requires DL Reasoning to check for *role subsumption*.
The aim of this stage is to extract only those roles $r \in \mathsf{N_R^\pm}$ for which a corresponding pair of transducer edges exists in $down \times up$ obtained from Stage 1. Each such r is associated with the minimum cost $w_1 + w_2$ of such an edge.
The procedure to compute these edges is presented as *Procedure filterRoles*. The following lemma formalizes the results computed by this Stage.

Lemma 2. *Let $\mathfrak{R}$ be a 2RPQ, $\mathfrak{T}$ a distortion transducer, $\mathcal{K} = (\mathcal{T},\mathcal{A})$ an $\mathcal{ELH}$ KB, and $(s,t),(s',t'),(s_1,t_1),(s_2,t_2) \in (Q_\mathfrak{R} \times Q_\mathfrak{T})$. Further, let $r \in N_R \cap Sig(\mathcal{T})$. After* Procedure filterRoles *finishes, it holds that $(r,w) \in candidateRoles$ iff $w = w_1 + w_2$ and w_1,w_2 are minimal amongst all $(t,u,R',w_1,t_1),(t_2,u',R''^-,w_2,t') \in \delta_\mathfrak{T}$ s.t. $\mathcal{T} \models r \sqsubseteq R'$, $\mathcal{T} \models r \sqsubseteq R''$, $(s,u,s_1),(s_2,u',s') \in \delta_\mathfrak{R}$.*

Proof. $\Leftarrow$: Let $e_1 = (t,u,R',w_1,t_1) \in edgesDown$ and $e_2 = (t_2,u',R''^-,w_2,t') \in edgesUp$ s.t. $\mathcal{T} \models r \sqsubseteq R'$, $\mathcal{T} \models r \sqsubseteq R''$. From Lemma 1, we obtain that $(s,u,s_1) \in \delta_\mathfrak{R}$ and $(s_2,u',s') \in \delta_\mathfrak{R}$. Assume there is no $e_1' = (t,u_1',R_1',w_1',t_1) \in edgesDown$ s.t. $\mathcal{T} \models r \sqsubseteq R_1'$ and $w_1' < w_1$ (otherwise, e_1' could have been chosen). Analogously, assume e' was chosen s.t. w_2 is minimal. Because the edges in $down$ and up are sorted by their weight, we obtain that e(e' respectively) is the first edge processed within the for loop at line 5(14) that satisfies the condition in line 6(15). Thus, after *Procedure filterRoles* finishes line 22, we have $downCost = w_1$ and $upCost = w_2$, and thus $(r, w_1 + w_2) \in candidateRoles$.

$\Rightarrow$: After *Procedure filterRoles* finishes, let $(r, w_1 + w_2) \in candidateRoles$. Then, there is $e_1 = (t, u, R', w_1, t_1) \in edgesDown$ s.t. $\mathcal{T} \models r \sqsubseteq R'$, and there is no $e_1' = (t, u_1', R_1', w_1', t_1) \in edgesDown$ s.t. $\mathcal{T} \models r \sqsubseteq R_1'$ and $w_1' < w_1$ (otherwise, e_1' would have been processed before e_1 by the for loop in line 5). Analogously, we obtain $e_2 = (t_2, u', R''^-, w_2, t') \in edgesUp$. Because $e_1 \in edgesDown$ and $e_2 \in edgesUp$, we have $(s, u, s_1), (s_2, u', s') \in \delta_{\mathfrak{R}}$. □

Procedure 2 *filterRoles*

Input: sorted sets $down, up \subseteq \delta_{\mathfrak{T}}$ obtained from Stage 1
Output: A set $candidateRoles \subseteq (\mathsf{N_R} \times \mathbb{N})$

```
1: initialise candidateRoles = {}
2: for all r ∈ N_R ∩ Sig(T) do
3:     initialise c_down := ∞, c_up := ∞
4:     for all (t, u, R', w_1, t_1) ∈ down do
5:         if T ⊨ r ⊑ R' then
6:             set c_down := w_1
7:             exit and continue at line 10
8:         end if
9:     end for
10:    if c_down = ∞ then
11:        discard and continue with next r
12:    end if
13:    for all (t_2, u, R''^-, w_2, t') ∈ up do
14:        if T ⊨ r ⊑ R'' then
15:            set c_up := w_2
16:            exit and continue at line 19
17:        end if
18:    end for
19:    if c_up = ∞ then
20:        discard and continue with next r
21:    end if
22:    add (r, c_down + c_up) to candidateRoles
23: end for
24: return candidateRoles
```

4.2.1.3. Stage 3: Checking entailment and updating spa

The third stage computes the updated values for the loop table spa. While directly using the current state of spa for the computation would yield correct results, we notice that not all entries currently present in our loop table have to be considered. This is based on the following observation:

- The results obtained from Stage 2 are independent of spa and thus immutable during the entire construction.
- The results calculated during one run of Stage 3 only depend on the entries in spa and the results obtained from Stage 2

This means that a subsequent iteration of Stage 3 can only produce different results if spa has been updated since the previous iteration. A corresponding observation based on *Procedure SPA* is formalized in the following lemma:

Lemma 3. *Consider a run of* Procedure SPA*. For two subsequent iterations i, i^+ of f, let spa^i be the state of spa at the beginning of i, and spa^{i^+} the state of spa at the beginning of i^+. Then, if the value $spa[(s,t),(s',t'),C]$ was updated during i^+ using an application of rule S1 of the form $w_1{}^{i^+}+spa^{i^+}[(s_1,t_1),(s_2,t_2),A]+w_2{}^{i^+}$, we have $spa^{i^+}[(s_1,t_1),(s_2,t_2),A] < spa^i[(s_1,t_1),(s_2,t_2),A]$*

Proof. Assume the contrary: If $spa^{i^+}[(s_1,t_1),(s_2,t_2),A] = spa^i[(s_1,t_1),(s_2,t_2),A]$ and $spa[(s,t),(s',t'),C]$ was updated with value v during iteration i^+ using an S1-application of the form $w_1+spa^{i^+}[(s_1,t_1),(s_2,t_2),A]+w_2$, it follows that there are $e_1, e_2 \in \delta_\mathfrak{T}$ with $c(e_1) = w_1$ and $c(e_2) = w_2$ s.t. condition $C1^*$ is satisfiable. Then, the same e_1, e_2 could have been chosen to update $spa[(s,t),(s',t'),C]$ using $w_1 + spa^i[(s_1,t_1),(s_2,t_2),A] + w_2$ during the previous iteration i. Consequently, we would have $spa^{i^+}[(s,t),(s',t'),C] = v$, which is a contradiction to the fact that $spa[(s,t),(s',t'),C]$ was updated during i^+.

□

Our algorithm exploits this by keeping track of the changes applied to spa during each iteration. To this end, Stage 3 receives, in addition to the actual loop table spa, a fragment of spa^* containing only entries that were updated since the previous iteration. Accordingly, the output is another fragment containing only the entries of spa that were updated during this run.

To calculate the updated values, DL Reasoning is employed to retrieve all concepts C that satisfy $C \sqsubseteq \exists r.A$, for each combination of (r,w) obtained from Stage 2 and (A,c) for which an entry $spa^*[(s_1,t_1),(s_2,t_2),A] = c$ with $c \in \mathbb{N}$ exists. For each such C, it is ensured that condition $C1^*$ is satisfiable using A and r. From Lemma 2, we obtain that w is minimal for the chosen r. Accordingly $spa[(s,t),(s',t'),C]$ can be updated if the calculated value is lower than the current value.

Finally, we can show that *Procedure S1 - $\mathcal{ELH}$* correctly computes all S1-Applications as required in *Procedure SPA - $\mathcal{ELH}$* for one iteration of f. The following lemma states the equivalence between these operations:

Lemma 4. *Let $\mathfrak{R}$ a 2RPQ, $\mathfrak{T}$ a distortion transducer, $\mathcal{K} = (\mathcal{T},\mathcal{A})$ an $\mathcal{ELH}$ KB, spa a loop table, and spa^* a fragment of spa containing all entries updated since the previous iteration. After a run of Procedure S1 - $\mathcal{ELH}$ with result spa^{**}, the following holds for all $(C) \in \mathsf{T}(\mathcal{U}_\mathcal{K})$:*
$spa^{**}[(s,t),(s',t'),C] = v$ iff v is the minimal[1] updated value after applying rule S1 to $((s,t),(s',t'),C)$.

Proof. ($\Leftarrow$) Let $C \in \mathsf{T}(\mathcal{U}_\mathcal{K})$. We consider two cases: First, if $spa[(s,t),(s',t'),C]$ was not updated during an application of S1, there is no $(t,u,R',w_1,t_1),(t_2,u',R''^-,w_2,t') \in \delta_\mathfrak{T}$ and $A \in \mathsf{T}(\mathcal{U}_\mathcal{K})$ s.t. condition $C1^*$ is satisfied and $w_1+spa[(s_1,t_1),(s_2,t_2),A]+w_2 < spa[(s,t),(s',t'),C]$. By Lemma 2, it holds that $candidateRoles = \varnothing$, and thus spa^{**} contains no entries.

For the second case, assume $spa[(s,t),(s',t'),C]$ was updated with value v by an S1-application. Then, there is $(t,u,R,w_1,t_1),(t_2,u',R''^-,w_2,t') \in \delta_\mathfrak{T}, (s,u,s_1),(s_2,u',s') \in \delta_\mathfrak{R}$, $r \in \mathsf{N}_\mathsf{R}^\pm$ and $A \in \mathsf{N}_\mathsf{C}$ s.t. condition $C1^*$ is satisfied, i.e. it holds that:

[1] We can assume that v is the minimum value amongst all S1-Applications to $((s,t),(s',t'),C)$

(i) $v = w_1 + spa[(s_1, t_1), (s_2, t_2), A] + w_2$,

(ii) $\mathcal{T} \models r \sqsubseteq R'$, $\mathcal{T} \models r \sqsubseteq R''$, and

(iii) $\mathcal{T} \models C \sqsubseteq \exists r.A$.

Let $c = spa[(s_1, t_1), (s_2, t_2), A]$ and $w = w_1 + w_2$. By Lemma 2, we have $(r, w) \in candidateRoles$. Additionally, we obtain $spa^*[(s_1, t_1), (s_2, t_2), A] = c$ from Lemma 3, and therefore $(A, c) \in fillers$. From (iii), it follows that $C \in subsumees$. As v can be used to update $spa[(s, t), (s', t'), C]$, it holds that $spa[(s, t), (s', t'), C] > v$, and therefore $spa[(s, t), (s', t'), C] > w + c$. As a consequence, we obtain $spa^{**}[(s, t), (s', t'), C] \leq v$ after processing line 14.

Assume $spa^{**}[(s, t), (s', t'), C] < v$. Then, there must exists some $(s, w') \in candidateRoles$ and $(B, c') \in fillers$ s.t. $\mathcal{T} \models C \sqsubseteq \exists s.B$ and $w' + c' < v$. By Lemma 2, this implies there are $e_1 = (t, u, S', w'_1, t_1)$ and $e_2 = (t_2, u', S''^-, w'_2, t') \in \delta_{\mathfrak{T}}$ with $w' = w'_1 + w'_2$ s.t. $\mathcal{T} \models s \sqsubseteq S'$, $\mathcal{T} \models s \sqsubseteq S''$; and $c' = spa^*[(s, t), (s', t'), B]$ s.t. $\mathcal{T} \models C \sqsubseteq \exists s.B$. Then, condition $C1^*$ is satisfied using e_1, e_2 and B. Consequently $spa[(s, t), (s', t'), C]$ could have been updated with $w'_1 + c' + w'_2 < v$, which is a contradiction to v being the minimal updated value. It follows that $spa^{**}[(s, t), (s', t'), C] = v$. □

Regarding complexity, we obtain that procedures *filterEdges* and *filterRoles* are computable in at most polynomial time in the combined size of $\mathfrak{R}$, $\mathfrak{T}$ and $\mathcal{K}$. The amount of elements in $candidateRoles$ is bounded by the size of Sig($\mathcal{T}$), and the amount of entries in spa^* is limited by the size of spa, hence polynomial in the combined size of $\mathfrak{R}$, $\mathfrak{T}$ and $\mathcal{K}$. For each combination of $(r, w) \in candidateRoles$ and $(M, c) \in fillers$, calculating all subsumees is required. For $\mathcal{K}$ an $\mathcal{ELH}$ KB, this is feasible in at most polynomial time [BOŠ13]. In summary, we obtain that a run of *Procedure S1 - $\mathcal{ELH}$* requires at most polynomial time in combined complexity. Thus, our upper bound matches the one presented for computing all S1-Applications given in [FT21].

4.2.2. Calculating S2 over $\mathcal{ELH}$ KBs

Here, we present a procedure to calculate all S2-applications as required in line 2 of *Procedure SPA*. Note that this procedure is only applied once during the entire computation of spa, whereas S1 and S3 Applications are repeated for each iteration of the function f.

Calculating S2-Applications can be considered a simpler and slightly adapted variant of calculating S1-Applications, for which the content of spa is irrelevant. For this reason, the procedure presented here shows a lot of similarities to the one presented in section 4.2.1 above.

Procedure 3 *S1 - $\mathcal{ELH}$*

Input: Loop table spa, partial loop table spa^*
Output: A relation spa^{**} containing updates to spa

initialize empty table spa^{**}
for all $\mathfrak{p}, \mathfrak{q}, \mathfrak{p}_1, \mathfrak{p}_2 \in (\delta_{\mathfrak{R}} \times \delta_{\mathfrak{T}})^4$ do
    if $s = s'$ and $t = t'$: then
        Skip and continue with next $\mathfrak{p}, \mathfrak{q}, \mathfrak{p}_1, \mathfrak{p}_2$
    end if

    set $(down, up) := filterEdges(\mathfrak{p}, \mathfrak{q}, \mathfrak{p}_1, \mathfrak{p}_2)$
    set $candidateRoles := filterRoles(down, up)$

    set $fillers := \{(A, c) \mid spa^*[\mathfrak{p}_1, \mathfrak{p}_2, A] = c, A \in \mathsf{T}(\mathcal{U}_\mathcal{K}), c \in \mathbb{N}\}$
    for all $(r, w) \in candidateRoles$ do
        for all $(A, c) \in fillers$ do
            Calculate $subsumees := \{C \mid C \in \mathsf{Sig}(\mathcal{T}) \cap \mathsf{N_C}, C \sqsubseteq \exists r.A\}$
            for all $C \in subsumees$ do
                if $spa[\mathfrak{p}, \mathfrak{q}, C] > w + c$ then
                    set $spa^{**}[\mathfrak{p}, \mathfrak{q}, C] = min(spa^{**}[\mathfrak{p}, \mathfrak{q}, C], w + c)$
                end if
            end for
        end for
    end for
end for

return spa^{**}

Procedure 4 *S2 - $\mathcal{ELH}$*

Output: relation spa^{**} containing all S2-Applications

initialize empty table spa^{**}
for all $(s, t), (s', t') \in (\delta_{\mathfrak{R}} \times \delta_{\mathfrak{T}})^2$ do
    if $s = s'$ and $t = t'$: then
        Skip and continue with next $(s, t), (s', t')$
    end if
    set $edges := \{(t, u, A?, w, t') \mid (s, u, s') \in \delta_{\mathfrak{R}}, (t, u, A?, w, t') \in \delta_{\mathfrak{T}}, A \in \mathsf{N_C}\}$

    for all $(t, u, A?, w, t') \in edges$ do
        Calculate $subsumees := \{C \mid C \in \mathsf{T}(\mathcal{U}_\mathcal{K}), \mathcal{T} \models C \sqsubseteq A\}$
        for all $C \in subsumees$ do
            set $spa[(s, t), (s', t'), C] := min(spa[(s, t), (s', t'), C],\ w)$
        end for
    end for
end for

return spa^{**}

The procedure consists of only 2 Stages, roughly corresponding to Stages 1 and 3 from *Procedure S1*. Line 6 extracts edges $(t, u, A?, w, t') \in \delta_{\mathfrak{T}}$ s.t. a corresponding $(s, u, s') \in \delta_{\mathfrak{R}}$ exists. This requires at most linear time in the size of $\delta_{\mathfrak{R}}$ and $\delta_{\mathfrak{T}}$, and the amount of such edges is restricted by $|\delta_{\mathfrak{T}}|$. For each such edge, we need to compute all $A \in \mathsf{N_C} \cap \mathsf{Sig}(\mathcal{T})$ s.t. $\mathcal{T} \models C \sqsubseteq A$ holds, which is feasible in polynomial time in the size of $\mathcal{K}$ [BO15]. Hence, we obtain a P-Time upper bound in the combined size of $\mathfrak{R}, \mathfrak{T}$ and $\mathcal{K}$, matching the upper bound obtained from [FT21].

The following lemma shows that a run of *Procedure S2 -* $\mathcal{ELH}$ correctly computes all S2-Applications.

Lemma 5. *Let $\mathfrak{R}$ a 2RPQ, $\mathfrak{T}$ a distortion transducer, $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ an $\mathcal{ELH}$ KB. Let spa' be the state of spa after Procedure SPA finishes line 2.For all $C \in \mathsf{T}(\mathcal{U}_\mathcal{K})$, the following holds:*
After a run of Procedure S2 - $\mathcal{ELH}$ that returns spa^{**}, we have $spa^{**}[(s,t),(s',t'),C] = w$ iff $spa'[(s,t),(s',t'),C]$ was updated with w after applying S2 to $((s,t),(s',t'),C)$.

Proof. If $spa'[(s,t),(s',t'),C]$ was not updated by an S2-Application, there is no $(s,u,s') \in \delta_{\mathfrak{R}}$ and $(t,u,A?,v,t') \in \delta_{\mathfrak{T}}$ s.t. $\mathcal{T} \models C \sqsubseteq A$. Then, for all $(t,u,A?,w,t') \in edges$, the set $subsumees$ calculated in line 8 is empty. It follows that $spa^{**}[(s,t),(s',t'),C]$ is never updated.

Assume $spa'[(s,t),(s',t'),C]$ was updated with value w. Then, there is $(s,u,s') \in \delta_{\mathfrak{T}}$ and $(t,u,A?,w,t') \in \delta_{\mathfrak{T}}$ s.t. $\mathcal{T} \models C \sqsubseteq A$. Hence, we have $(t,u,A?,w,t') \in edges$, and $C \in subsumees$, and consequently $spa^{**}[(s,t),(s',t'),C] \leq w$. Assume that $spa^{**}[(s,t),(s',t'),C] < w$. Then, there is some $(t,u',B?,v,t') \in edges$, and consequently $(s,u',s') \in \delta_{\mathfrak{R}}$, s.t. $\mathcal{T} \models C \sqsubseteq B$ and $v < w$. Then, the same (s,u',s') and $(t,u',B?,v,t')$ could have been used to update $spa'[(s,t),(s',t'),C]$, which is a contradiction to the fact that this entry was updated with w.
It follows that $spa^{**}[(s,t),(s',t'),C] = spa'[(s,t),(s',t'),C]$. □

4.2.3. Calculating S3 over $\mathcal{ELH}$ KBs

While the procedures to calculate S1- and S2-Applications show some similarities, S3-Applications require an entirely different approach. The deterministic procedure presented here is based on the idea mentioned in [FT21] to calculate S3-Applications using Floyd-Warshall algorithm. To explain the correspondence between S3-Applications and finding shortest paths in a graph, let us recall what the information stored in spa represents:

Given an $\mathcal{ELH}$ or $\mathcal{ELHI}_\bot$ KB $\mathcal{K}$, the table spa is used to identify loops through the anonymous part of $\mathcal{U}_\mathcal{K}$, starting and ending at the same anonymous individual e. As explained in section 2.3, the subtree rooted at such an individual (denoted as T_e) can be uniquely identified its tail. After $\mathfrak{d}$ iterations of f, an entry $spa[(s,t),(s',t'),C] = w$ represents the fact: If $tail(e) = C$, $G_{\mathcal{U}_\mathcal{K}}$ contains a path from (s,t,e) to (s',t',e) with cost w and depth at most $\mathfrak{d}$.

The idea behind rule applications of the form $spa[(s,t),(s',t'),C] \ll spa[(s,t),(s'',t''),C] + spa[(s'',t''),(s',t'),C]$ is to combine the information currently stored in spa to find paths p_1, p_2 in $G_{\mathcal{U}_\mathcal{K}}$ that go from (s,t,e) to (s'',t'',e) and respectively from (s'',t'',e) to (s',t',e). By combining such paths, we obtain a new path p_3 from (s,t,e) to (s',t',e) with cost $c(p_3) = c(p_1) + c(p_2)$. Therefore, we can update $spa[(s,t),(s',t'),C]$ with $c(p_3)$. As the updated values can only decrease, there is a finite amount of S3-Applications after which the value no longer changes.

Consider a run of *Procedure SPA* and let $e \in \Delta^{\mathcal{U}_\mathcal{K}} \setminus \mathsf{Ind}(\mathcal{A})$ with $tail(e) = C$. During the $\mathfrak{d} - th$ iteration of f, after exhaustively applying rule S3 to all $spa[(s,t),(s',t'),C]$, we obtain that $spa[(s,t),(s',t'),C] = c(p')$ exactly if p' is a minimum-cost path from (s,t,e) to (s',t',e) in $G_{\mathcal{U}_\mathcal{K}}$ with maximum depth $\mathfrak{d}$ [FT21][2]. From here, it is easy to see how Floyd-Warhshall could be used on $G_{\mathcal{U}_\mathcal{K}}$ to find the updated value for this entry. However, as constructing $G_{\mathcal{U}_\mathcal{K}}$ is not an option, we use the information stored in spa to construct a set of graphs that allow us to find similar paths.

Note that S3-Applications only consider entries with the same $C \in \mathsf{T}(\mathcal{U}_\mathcal{K})$, meaning for different tails $C, D \in \mathsf{T}(\mathcal{U}_\mathcal{K})$, S3-Applications can be calculated independently of each other. To this end, for each $C \in \mathsf{T}(\mathcal{U}_\mathcal{K})$, we define as $G_C = (V_C, E_C)$ a weighted graph with

- $V_C := \{(s,t) \mid s \in \delta_\mathfrak{R}, t \in \delta_\mathfrak{T}\}$
- $E_C := \{((s,t), w, (s',t')) \mid spa[(s,t),(s',t'),C] = w\}$

Such a graph has exactly $|Q_\mathfrak{R} \cdot Q_\mathfrak{T}|$ nodes and $|Q_\mathfrak{R} \cdot Q_\mathfrak{T}|^2$ edges. The following lemma shows that the cost of the shortest path from (s,t) to (s',t') in G_C corresponds the minimum value of $spa[(s,t),(s',t'),C]$ after exhaustively applying S3.

Lemma 6. *Let $\mathfrak{R}$ a 2RPQ, $\mathfrak{T}$ a distortion transducer, $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ an $\mathcal{ELH}$ or $\mathcal{ELHI}_\bot$ KB, spa a loop table. Further, let spa' be the state of spa after an exhaustive application of rule S3. For all $((s,t),(s',t'),C) \in (\delta_\mathfrak{R} \times \delta_\mathfrak{T})^2 \times \mathsf{T}(\mathcal{U}_\mathcal{K})$, the following holds:*
Let p be a path of minimal cost from (s,t) to (s',t') in G_C. Then, we have $c(p) = spa'[(s,t),(s',t'),C]$.

Proof. $spa'[(s,t),(s',t'),C] \leq c(p)$: Assume $c(p) \neq \infty$. Then, p is a path of length $n+1$ in G_C of the form

$$(s,t)w_1(s_1,t_1)...(s_n,t_n)w_{n+1}(s',t') \text{ with } c(p) = w_1 + ... + w_{n+1} \ (n \geq 0)$$

If $n = 0$, we have $p = (s,t)w_1(s',t')$. Consequently, $(s,t), w_1(s',t')$ is an edge in G_C, and thus $spa[(s,t)(s',t'),C] = w_1$. It follows that $c(p) \leq spa'[(s,t),(s',t'),C]$.

For $n = 1$, we have a rather trivial case where p is of the form $(s,t)w_1(s_1,t_1)w_2(s',t')$. Then, we have $spa^*[(s,t),(s_1,t_1),C] = w_1$ and $spa^*[(s_1,t_1),(s',t'),C] = w_2$. Consequently, we can use a single application of S3 to update $spa[(s,t),(s',t'),C]$ using $spa[(s,t),(s_1,t_1),C] + spa[(s_1,t_1),(s',t'),C]$.

We use induction on n to show that, for all $n > 1$, there is a corresponding sequence of S3-Applications s.t. $spa[(s,t),(s',t'),C]$ can be updated with v.

For $n = 2$, p is of the form $(s,t)w_1(s_1,t_1)w_2(s_2,t_2)w_3(s',t')$. Then, there is $spa^*[(s,t),(s_1,t_1),C] = w_1$, $spa^*[(s_1,t_1),(s_2,t_2),C] = w_2$ and and $spa^*[(s_2,t_2),(s',t'),C] = w_3$. Consequently, the following sequence of S3 applications can be applied: First, update $spa[(s,t),(s_2,t_2),C]$ with value $w_1 + w_2$, followed by an update to $spa[(s,t),(s',t'),C]$ using $spa[(s,t),(s_2,t_2),C] + spa[(s_2,t_2),(s',t'),C]$. We obtain $spa[(s,t),(s',t'),C] = w_1 + w_2 + w_3$.

[2] A corresponding proof for $\mathcal{ELHI}_\bot$ can be found in the appendix 16.

For $i = n + 1$, we have p is of the form $(s,t), w_1(s_1,t_1)...(s_n,t_n)w_{n+1}(s_{n+1},t_{n+1})w_{n+2}(s',t')$. By induction, we have $spa[(s,t),(s_{n+1},t_{n+1}),C] = w_1 + ... + w_{n+1}$, and S3 can be applied to update $spa[(s,t),(s',t'),C]$ using $spa[(s,t),(s_n,t_n),C] + spa[(s_n,t_n),(s',t'),C]$ with $w_1 + ... + w_{n+2} = v$.

This satisfies our claim, and we obtain that $spa'[(s,t),(s',t'),C] \leq c(p)$.

$c(p) \leq spa'[(s,t),(s',t'),C]$: Let $spa'[(s,t),(s',t'),C] = v$. If v was not updated by an S3-Application, we have $v = spa[(s,t),(s',t'),C]$. Consequently, there is an edge $((s,t)v(s',t')$ in G_C. It follows that $c(p) < v$. Otherwise, let $\mu_1, ..., \mu_k$ be the sequence of S3-Applications that were used to obtain spa'. Assume this sequence is ordered, i.e. μ_1 was the first update and μ_k the last one. In addition, we denote as spa^i the state of spa after update μ_i. Now, let μ_j be the final update to $spa'[(s,t),(s',t'),C]$, i.e. $spa'[(s,t),(s',t'),C] = spa^j[(s,t),(s',t'),C]$.

We show by induction on the length j that, for each such sequence, there is a corresponding path p in G_C with $c(p) = spa'[(s,t),(s',t'),C]$:

For the case where $j = 1$, only a single application was performed, i.e. there is (s'',t'') s.t. $spa[(s,t),(s'',t''),C] = w_1$, $spa[(s'',t''),(s',t'),C] = w_2$ and $v = w_1 + w_2$. Then, there are corresponding edges $((s,t),w_1,(s'',t''))$ and $((s'',t''),w_2,(s',t'))$ in G_C. It follows that there is a path $p = (s,t)w_1(s'',t'')w_2(s',t')$ in G_C with $c(p) = v$.

For $j > 2$: μ_j corresponds to an update $spa'[(s,t),(s',t'),C] = spa^{j-1}[(s,t),(s'',t''),C] + spa^{j-1}[(s'',t''),(s',t'),C]$ for some (s'',t''). By applying induction, we obtain that there are paths $p_1 = (s,t)...(s'',t'')$ and $p_2 = (s'',t'')...(s',t')$ in $G_{\mathcal{U}_\mathcal{K}}$ s.t. $c(p_1) = spa^{j-1}[(s,t),(s'',t''),C]$ and $c(p_2) = spa^{j-1}[(s,t),(s'',t''),C]$. By combining these paths, we obtain p' with $c(p') = spa'[(s,t),(s',t'),C]$, thus satisfying our claim.

As p is a path of minimal cost from (s,t) to (s',t'), it follows that $c(p) \leq c(p')$, and consequently $c(p) \leq spa'[(s,t),(s',t'),C]$.

As we have shown both direction, it holds that $c(p) = spa'[(s,t),(s',t'),C]$. □

We present *Procedure S3*, which is used to calculate the result exhaustively applying rule S3 as required in line 3 and one iteration of f in *Procedure SPA*. For each $C \in \mathsf{T}(\mathcal{U}_\mathcal{K})$, a graph G_C is constructed using the content of spa, and Floyd-Warhshall is used to find the cost of shortest paths between all nodes. By Lemma 6, the minimal cost of a path from (s,t) to (s',t') can be used to update $spa[(s,t),(s',t'),C]$. Similar to *Procedure S1*, a relation spa^{**} containing only the updated entries is returned.

For $\mathcal{ELH}$, the size of $\mathsf{T}(\mathcal{U}_\mathcal{K})$ is polynomial in |Sig($\mathcal{T}$)|. Building G_C is feasible in linear time in the combined size of $Q_\mathfrak{R}$ and $Q_\mathfrak{T}$. The time complexity of calculating the minimal cost of all paths using Floyd-Warshall is $|V_C|^3$ ([Flo62],[War62]), and the amount of nodes in G_C is exactly $|Q_\mathfrak{R} \times Q_\mathfrak{T}|$. Thus, a run of *Procedure S3* is feasible in at most polynomial time in the combined size of $\mathfrak{R}, \mathfrak{T}$ and $\mathcal{K}$.

Using Lemma 6, it is easy to see that *Procedure S3* calculates the correct results.

Lemma 7. *Let $\mathfrak{R}$ a 2RPQ, $\mathfrak{T}$ a distortion transducer, $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ an $\mathcal{ELH}$ or $\mathcal{ELHI}_\bot$ KB. After a run of Procedure S3, for all $((s,t),(s',t'),C) \in (\delta_\mathfrak{R} \times \delta_\mathfrak{T})^2 \times \mathsf{T}(\mathcal{U}_\mathcal{K})$, we have $spa^{**}[(s,t),(s',t'),C] = v$ iff v is the updated value of $spa[(s,t),(s',t'),C]$ after exhaustively applying rule S3.*

Proof. Let spa^* be the state of spa used as input to *Procedure S3* (before any applications of rule S3) and spa' be the state of spa after exhaustively applying rule S3.

By Lemma 6, for each $C \in \mathsf{T}(\mathcal{U}_\mathcal{K})$, it holds that $spa'[(s,t),(s',t'),C] = w$ iff w is the minimal cost of a path from (s,t) to (s',t') in G_C. For an arbitrary $C \in \mathsf{T}(\mathcal{U}_\mathcal{K})$ chosen in line 2, after the procedure reaches line 11, $distance((s,t),(s',t')$ contains the minimal cost of path from (s,t) to (s',t') in G_C for all $(s,t),(s',t') \in (\delta_\mathfrak{R} \times \delta_\mathfrak{T})$. Thus, we have $spa'[(s,t),(s',t'),C] = distance((s,t),(s',t'))$. If v was not updated by an application of rule S3, we have $spa^*[(s,t),(s',t'),C] = spa'[(s,t),(s',t'),C]$, and consequently $spa^{**}[(s,t),(s',t'),C]$ is not set. Otherwise, we have $spa'[(s,t),(s',t'),C] < spa^*[(s,t),(s',t'),C]$, and thus $spa^{**}[(s,t),(s',t'),C]$ receives value v in line 13. □

Procedure 5 *S3*

Input: An $\mathcal{ELH}$ or $\mathcal{ELHI}_\bot$ KB $\mathcal{K} = (\mathcal{T},\mathcal{A})$, a 2RPQ $\mathfrak{R}$, a dt $\mathfrak{T}$, loop table spa
Output: relation spa^{**} containing all updated entries

```
1:  initialize empty table spa**
2:  for all C ∈ T(U_K) do
3:      set V := {(s,t) | (s,t) ∈ (δ_R × δ_T)}
4:      set E := {((s,t),0,(s,t)) | (s,t) ∈ (δ_R × δ_T)}
5:      for all (s,t),(s',t') ∈ (δ_R × δ_T)^2 do
6:          if spa[(s,t),(s',t'),C] < ∞ then
7:              add ((s,t), spa[(s,t),(s',t'),C], (s',t')) to E
8:          end if
9:      end for

10:     calculate distance for (V,E) using Floyd-Warshall
11:     for all (s,t),(s',t') ∈ V do
12:         if distance((s,t),(s',t')) < spa[((s,t),(s',t'),C)] then
13:             set spa** = distance((s,t),(s',t'))
14:         end if
15:     end for
16: end for
17:
18: return spa**
```

4.3. Rule calculation over $\mathcal{ELHI}_\bot$ KBs

In this section, we present deterministic, moderate-case optimized procedures to compute S1-, S2- and S3-Applications under the presence of an $\mathcal{ELHI}_\bot$ KBs. We start by defining rules S1, S2 and S3.

S1. $spa[(s,t),(s',t'),M] \ll w_1 + spa[(s_1,t_1),(s_2,t_2),M_1] + w_2$, if C1* holds.

S2. $spa[(s,t),(s',t'),M] \ll w$, if $\mathcal{T} \models M \sqsubseteq A$, $(s,u,s') \in \delta_\mathfrak{R}$, and $(t,u,A?,w,t') \in \delta_\mathfrak{T}$,

S3. $spa[(s,t),(s',t'),M] \ll spa[(s,t),(s'',t''),M] + spa[(s'',t''),(s',t'),M]$.

$C2^*$. $M_1 \subseteq \mathsf{N}_C, \mathcal{T} \models M \sqsubseteq \exists R.M_1, \mathcal{T} \models R \sqsubseteq R', \mathcal{T} \models R^- \sqsubseteq R'', (s, u, s_1) \in \delta_{\mathfrak{R}}, (t, u, R', w_1, t_1) \in \delta_{\mathfrak{T}}, (s_2, u', s') \in \delta_{\mathfrak{R}_j}$ and $(t_2, u', R'', w_2, t') \in \delta_{\mathfrak{T}}$.

with $R, R', R'' \in \mathsf{N}_\mathsf{R}^{\pm}$ and $M, M_1 \subseteq \mathsf{N}_\mathsf{C}$.

In the presence of an $\mathcal{ELHI}_\bot$ KB, calculating applications of rules S1,S2 and S3 is lot harder than it is for $\mathcal{ELH}$. Recall that, for an $\mathcal{ELH}$ KB, the set $\mathsf{T}(\mathcal{U}_\mathcal{K})$ consists of single concept names $C \in \mathsf{Sig}(\mathcal{T}) \cap \mathsf{N}_\mathsf{C}$, whereas for an $\mathcal{ELHI}_\bot$ KB, the set $\mathsf{T}(\mathcal{U}_\mathcal{K})$ contains all conjunctions of these concept names. As introduced in chapter 2, such conjunctions will be treated as sets of concept names.

From Appendix B, we obtain that a single application of rule S1 requires exponential time in the size of $\mathcal{T}$, mainly for two reasons:

- Checking Entailment is EXP-Complete for $\mathcal{ELHI}_\bot$
- The amount of elements in $\mathsf{T}(\mathcal{U}_\mathcal{K})$ is exponential in the size of $\mathcal{T}$

Analogously to the previous section, all procedures presented in this section assume the presence of an $\mathcal{ELHI}_\bot$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, a 2RPQ R and a distortion Transducer $\mathfrak{T}$.

4.3.1. Calculating S1 over $\mathcal{ELHI}_\bot$ KBs

The basic structure of the algorithm is very similar to the $\mathcal{ELH}$ case while accounting for the different nature of $\mathsf{T}(\mathcal{U}_\mathcal{K})$. However, a number of additional optimizations are employed with the aim to avoid calculations with predictable outcome.

Again, the algorithm consists of 3 Stages. Stages 1 and 2 are almost identical to the ones presented in section 4.2.1 with minor changes to account for the presence of inverse roles in the ontology. To avoid repetition, the adjusted subprocedures *filterEdges* and *filterRoles* are given in the appendix. The third stage features two main differences to the procedure presented for $\mathcal{ELH}$, which are explained below.

In Lines 11-14, a caching optimization is employed to keep track which $fillers$ have already been processed in the current iteration, and potentially skip to process others if the results cannot improve. Additionally, a more sophisticated procedure *calculateSubsumees* featuring a mixture of filtering and caching optimizations is used. We present the entire procedure first, and continue to discuss how the updates are calculated.

Procedure 6 *S1 - $\mathcal{ELHI}_\bot$*

Input: Loop table spa, partial loop table spa^*
Output: A relation spa^{**} containing updates to spa

1: initialize empty table spa^{**}
2: **for all** $(\mathfrak{p}, \mathfrak{q}, \mathfrak{p}_1, \mathfrak{p}_2 \in (\delta_{\mathfrak{R}} \times \delta_{\mathfrak{T}})^4$ **do**
3: **if** $s = s'$ and $t = t'$: **then**
4: Skip and continue with next $\mathfrak{p}, \mathfrak{q}, \mathfrak{p}_1, \mathfrak{p}_2$
5: **end if**

6: calculate $(down, up) := filterEdges(\mathfrak{p}, \mathfrak{q}, \mathfrak{p}_1, \mathfrak{p}_2)$
7: calculate $candidateRoles := filterRoles(down, up)$

8: initialise empty relation $processed \subseteq \mathsf{T}(\mathcal{U}_\mathcal{K}) \times \mathbb{N}$
9: set $fillers := \{(M_1, c) \mid spa^*[\mathfrak{p}_1, \mathfrak{p}_2, M_1] = c, M_1 \in \mathsf{T}(\mathcal{U}_\mathcal{K}), c \in \mathbb{N}\}$
10: **for all** $(M_1, c) \in fillers$[3] **do**
11: **if** $M' \subseteq M_1$ **and** $c = v$ for some $(M', v) \in processed$ **then**
12: Skip and continue with next M_1
13: **end if**
14: add (M_1, c) to $processed$
15: **for all** $(r, w) \in candidateRoles$ **do**
16: calculate $subsumees := calculateSubsumees(r, M_1, w + c)$
17: **for all** $M \in subsumees$ **do**
18: set $spa^{**}[\mathfrak{p}, \mathfrak{q}, M] = min(spa^{**}[\mathfrak{p}, \mathfrak{q}, M], w + c)$
19: **end for**
20: **end for**
21: **end for**
22: **end for**
23:
24: **return** spa^{**}

[2] elements are required to be processed in a specific order, see Remark 2

Stages 1 and 2 (lines 6 and 7) are used to extract roles $r \in \mathsf{N}_\mathsf{R}^\pm$ and an associated minimal cost $w \in \mathbb{N}$ s.t. condition $C2^*$ is partially satisfiable using r. From spa^*, we obtain sets $M_1 \in \mathsf{T}(\mathcal{U}_\mathcal{K})$ for which the corresponding entry $spa[(s_1, t_1), (s_2, t_2), M_1]$ has been updated since the previous iteration. Each combination of such r and M_1 corresponds to an expression $w_1 + spa[(s_1, t_1), (s_2, t_2), M_1] + w_2$ for which there is $(t, u, R', w_1, t_1), (t_2, u', R'', w_2, t') \in \delta_{\mathfrak{R}}$ s.t. the following is guaranteed:

(1) there is $(s, u, s_1), (s_2, u', s') \in \delta_{\mathfrak{R}}$,

(2) $\mathcal{T} \models r \sqsubseteq R'$ and $\mathcal{T} \models r^- \sqsubseteq R''$,

(3) $w = w_1 + w_2$ is minimal, i.e. there is no $(t, \rho, S', w_1', t_1), (t_2, \rho', S'', w_2', t') \in \delta_{\mathfrak{R}}$ with $w_1' < w_1$ or $w_2' < w_2$ satisfying conditions (1) and (2) for some $\rho, \rho' \in \mathsf{N}_{\mathsf{R},\mathcal{T}}^\pm$, $S', S'' \in \mathsf{N}_\mathsf{R}^\pm$

In order to find valid rule applications of the form

$$spa[(s,t),(s',t'),M] \ll w_1 + spa[(s_1,t_1),(s_2,t_2),M_1] + w_2,$$

it remains to find such M that satisfy $\mathcal{T} \models M \sqsubseteq \exists r.M_1$. The following section explains how procedure *calculateSubsumees* is used to find such M.

4.3.1.1. Calculating subsumees

Given r and M_1, a straight-forward way to find such sets M that satisfy $\mathcal{T} \models M \sqsubseteq \exists r.M_1$ is to check entailment for each $M \in \mathsf{T}(\mathcal{U}_\mathcal{K})$. However, such entailment checks are computationally expensive. The idea behind *calculateSubsumees* is to introduce a set of additional, but computationally less expensive checks that can be used to answer $\mathcal{T} \models M \sqsubseteq \exists r.M_1$ without using the DL Reasoner.

These optimizations use subset relations on sets $M, M' \in \mathsf{T}(\mathcal{U}_\mathcal{K})$. We start by making some observations that will be used to show the correctness of the construction.

The first observation is a direct consequence of applying semantics to such sets $M, M' \in \mathsf{T}(\mathcal{U}_\mathcal{K})$. Note that $\mathsf{T}(\mathcal{U}_\mathcal{K})$ does not contain the empty set.

Proposition 3. *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be an $\mathcal{ELHI}_\bot$ KB, $M \in \mathsf{T}(\mathcal{U}_\mathcal{K})$ and $M' \subseteq M$. Then, it holds that $\mathcal{T} \models M \sqsubseteq M'$.*

Proof. If $M = M'$, this is trivially satisfied. Otherwise, let $M' = \{C_1, ..., C_m\}$ and $M = M' \cup \{C_{m+1}, ..., C_n\}$ $(1 < m < n)$ with $C_i \in \mathsf{N_C} \cap \mathsf{Sig}(\mathcal{T})$ for all $i \leq m$. Given an interpretation I, we have $M'^I = {C_1}^I \cap ... \cap {C_m}^I$ and $M^I = M'^I \cap {C_{m+1}}^I \cap ... \cap {C_n}^I$. It follows that $M^I \subseteq M'^I$, and by applying semantics we obtain $\mathcal{T} \models M \sqsubseteq M'$. □

In the case where M' is a strict subset of M, we say that M is *more specific* than M'.

We can use subset relations between such sets to infer certain facts about our KB without the need to calculate (computationally expensive) reasoning tasks. Specifically, our algorithm makes use of the following observations:

Corollary 1. *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ an $\mathcal{ELHI}_\bot$ KB, $r \in N_R^\pm$ and $M, M_1 \in \mathsf{T}(\mathcal{U}_\mathcal{K})$. If $\mathcal{T} \not\models M \sqsubseteq \exists r.M_1$, it holds that $\mathcal{T} \not\models M' \sqsubseteq \exists r.M_1$ for all $M' \subseteq M$.*

Proof. Let I an interpretation, $M' \subseteq M$ and assume $\mathcal{T} \not\models M \sqsubseteq \exists r.M_1$. By applying semantics, we obtain that there is some $e \in M^I$ s.t. $e \notin \{d_1 \mid (d_1, d_2) \in r^I, d_2 \in M_1\}$. By proposition 3, it holds that $\mathcal{T} \models M \sqsubseteq M'$, and thus $M^I \subseteq M'^I$. Hence, the same e can be found in M'^I. It follows that $\mathcal{T} \not\models M' \sqsubseteq \exists r.M_1$. □

A similar observation can be made for the "other" direction:

Corollary 2. *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ an $\mathcal{ELHI}_\bot$ KB, $r \in N_R^\pm$ and $M, M_1 \in \mathsf{T}(\mathcal{U}_\mathcal{K})$. If $\mathcal{T} \models M \sqsubseteq \exists r.M_1$, it holds that $\mathcal{T} \models M' \sqsubseteq \exists r.M_1$ for all $M' \supseteq M$.*

Proof. Let I an interpretation, $M' \supseteq M$ and assume $\mathcal{T} \models M \sqsubseteq \exists r.M_1$. By proposition 3, it holds that $\mathcal{T} \models M' \sqsubseteq M$. By applying semantics, we obtain that for each $e \in M^I$, it holds that $e \in \{d_1 \mid (d_1, d_2) \in r^I, d_2 \in M_1\}$. Because $M'^I \subseteq M^I$, the same applies to all elements $e' \in M'^I$. It follows that $\mathcal{T} \models M' \sqsubseteq \exists r.M_1$. □

As a preliminary step, DL Reasoning is employed to retrieve the set $subsumees$ containing all atomic concepts $C \in \mathsf{N_C} \cap \mathsf{Sig}(\mathcal{T})$ s.t. $\mathcal{T} \models C \sqsubseteq \exists r.M_1$.

Subprocedure 7 *calculateSubsumees (S1 - $\mathcal{ELHI}_\bot$)*

Input: role $r \in \mathsf{N_R^{\pm}}$, set $M_1 \in \mathsf{T}(\mathcal{U}_\mathcal{K})$, update value $v \in \mathbb{N}$
Output: A set $\{M \mid \mathcal{T} \models M \sqsubseteq \exists r.M_1\} \subseteq \mathsf{T}(\mathcal{U}_\mathcal{K})$

1: initialise empty set $subsumees$
2: initialise empty set $eliminated$
3: Calculate $b := \{C \mid \mathcal{T} \models C \sqsubseteq \exists r.M_1, C \in \mathsf{N_C}\}$
4: **if** $b = \varnothing$ **then**
5: let $M^* = \{A | A \in Sig(\mathcal{T}) \cap \mathsf{N_C}\}$
6: **if** $\mathcal{T} \not\models M^* \sqsubseteq \exists r.M_1$ **then**
7: return $subsumees = \{\}$
8: **end if**
9: **end if**
10: **for all** $M \in \mathsf{T}(\mathcal{U}_\mathcal{K})$[4] **do**
11: **if** $spa[(s,t),(s',t'),M] <= v$ **then**
12: discard and continue with next M
13: **end if**
14: **if** $M \cap b \neq \varnothing$ **then**
15: add M to $subsumees$
16: **else**
17: **if** $M \subseteq M'$ for some $M' \in eliminated$ **then**
18: Discard and continue with next M
19: **end if**
20: **if** $T \models M \sqsubseteq \exists r.M_1$ **then**
21: add M to $subsumees$
22: **else**
23: add M to $eliminated$
24: **end if**
25: **end if**
26: **end for**
27:
28: **return** $subsumees$

[3] elements are required to be processed in a specific order, see Remark 1

Checking if $C2^*$ is satisfiable (Line 4-9)

The first check represents a filtering optimization to capture the case where there is no suitable $M \in \mathsf{T}(\mathcal{U}_\mathcal{K})$. The construction is based on the following observation:

Let $M^* = \{A | A \in Sig(\mathcal{T}) \cap \mathsf{N_C}\}$, i.e. the set containing all concept names from $\mathsf{Sig}(\mathcal{T})$. For all $M \in \mathsf{T}(\mathcal{U}_\mathcal{K})$, it holds that $M \subseteq M^*$. Thus, by Corollary 1, it holds that if $\mathcal{T} \not\models M^* \sqsubseteq \exists r.M_1$,

we have $\mathcal{T} \not\models M \sqsubseteq \exists r.M_1$ for all $M \in \mathsf{T}(\mathcal{U}_\mathcal{K})$. In that case, condition $C2^*$ is unsatisfiable, and the procedure returns an empty set.

By Corollary 2, $\mathcal{T} \models M^* \sqsubseteq \exists r.M_1$ is trivially satisfied if there is at least one $C \in \mathsf{N_C} \cap \mathsf{Sig}(\mathcal{T})$ s.t. $\mathcal{T} \models C \sqsubseteq \exists r.M_1$, i.e. the set b is not empty. In that case, testing $M^* \sqsubseteq \exists r.M_1$ can be skipped.

We proceed by iterating over all sets $M \in \mathsf{T}(\mathcal{U}_\mathcal{K})$. Recall that our subprocedure uses a fixed $w_1, spa[(s_1,t_1),(s_2,t_2),M_1]$ to calculate updates of the form $spa[(s,t),(s',t'),M] << w_1 + spa[(s_1,t_1),(s_2,t_2),M_1] + w_2$. To this end, it receives an input v representing the updated value. Lines 11-13 represent a simple condition to test whether the value currently stored in $spa[(s,t),(s',t'),M]$ can actually be improved using v.

Checking atomic subsumees (Lines 14-16)

Lines 14-16 represent another filtering optimization using the set $subsumees$ consisting of all atomic concepts $C \in \mathsf{N_C} \cap \mathsf{Sig}(\mathcal{T})$ that satisfy $\mathcal{T} \models C \sqsubseteq \exists r.M_1$. The optimization is based on the following correspondence:

If $M \cap b \neq \varnothing$, the following holds: Let $A \in M \cap b$ and $M_A = \{A\}$. Then, $\mathcal{T} \models M_A \sqsubseteq \exists r.M_1$ and $M_A \subseteq M$. By Corollary 2, it holds that $\mathcal{T} \models M \sqsubseteq \exists r.M_1$. It follows that v can be used to update $spa[(s,t),(s',t'),M]$.

Checking eliminated sets (Lines 17-19 + 23)

A caching optimization is employed to allow reusing intermediate results obtained while iterating over all $M \in \mathsf{T}(\mathcal{U}_\mathcal{K})$. To this end, a set $eliminated$ is used to collect such $M \in \mathsf{T}(\mathcal{U}_\mathcal{K})$ that satisfy $\mathcal{T} \not\models M \sqsubseteq \exists r.M_1$. The optimization is based on the following correspondence between such sets $M' \in eliminated$ and a set M:

For all $M' \in eliminated$, it holds that $\mathcal{T} \not\models M' \sqsubseteq \exists r.M_1$. Let $M \in \mathsf{T}(\mathcal{U}_\mathcal{K})$ s.t. there is some $M' \in eliminated$ with $M \subseteq M'$. By Corollary 1, it holds that $\mathcal{T} \not\models M \sqsubseteq \exists r.M_1$.
Lines 17-19 are used to test this condition and potentially discard the current M.

Finally, if the procedure reaches line 20, an entailment check to test whether $\mathcal{T} \models M \sqsubseteq \exists r.M_1$ holds, is performed. In case of a positive result, M is added to $subsumees$. Otherwise, we obtain $\mathcal{T} \not\models M \sqsubseteq \exists r.M_1$, and M is added to $eliminated$.
Note that this means the elements in $eliminated$ are collected in the order in which the sets $M \in \mathsf{T}(\mathcal{U}_\mathcal{K})$ are processed. The actual benefit obtained from this optimization for moderate-case scenarios depends on the order in which the sets are collected:

Remark 1. The order in which the sets $M \in \mathsf{T}(\mathcal{U}_\mathcal{K})$ in line 10 are processed has an impact on moderate-case performance. The best results are obtained if the elements are processed in order of descending size, i.e. M' is processed after M'' iff $M' \subset M''$.

The following lemma shows that *calculateSubsumees* calculates the correct results. As usual, we assume the presence of a fixed $(s,t),(s',t') \in (\delta_\mathfrak{R} \times \delta_\mathfrak{T})^2$ and loop table spa without explicitly stating these as inputs.

Lemma 8. *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ an $\mathcal{ELHI}_\bot$ KB, $r \in N_R^\pm$, $M_1 \in \mathsf{T}(\mathcal{U}_\mathcal{K})$ and $v \in \mathbb{N}$. A run of* calculateSubsumees *with input r, M_1, v calculates the the set*
$\{M \mid \mathcal{T} \models M \sqsubseteq \exists r.M_1, spa[(s,t),(s',t'),M] > v\}$.

Proof. To shorten notation, let $subsumees$ be the set returned by *calculateSubsumees* and $T = \{M \mid \mathcal{T} \models M \sqsubseteq \exists r.M_1, spa[(s,t),(s',t'),M] > v\}$.

$\Rightarrow subsumees \supseteq T$: Let $M \in T$. By Corollary 2, it holds that $\mathcal{T} \models M^* \sqsubseteq \exists r.M_1$, and consequently the algorithm proceeds to line 10. The condition $spa[(s,t),(s',t'),M] > v$ holds as a consequence of $M \in T$.
At line 14, we consider two possibilities: If $M \cap b \neq \varnothing$, we have $M \in subsumees$. Otherwise, the algorithm proceeds to line 17.

Assume there is $M' \in eliminated$ s.t. $M \subseteq M'$, i.e. $\mathcal{T} \not\models M' \sqsubseteq \exists r.M_1$. Then, by Corollary 1, we have $\mathcal{T} \not\models M \sqsubseteq \exists r.M_1$, which is a contradiction to $M \in T$. Therefore, there is no such M', and the algorithm proceeds to line 20. Finally, we have $\mathcal{T} \models M \sqsubseteq \exists r.M_1$ as a direct consequence of $M \in T$, and thus $M \in subsumees$. As M was chosen arbitrarily, we obtain that $M \in subsumees$ holds for all $M \in T$, and thus $subsumees \supseteq T$.

$\Leftarrow subsumees \subseteq T$: Let $M \in subsumees$. In order for M to be added to $subsumees$, we have (1) $spa[(s,t),(s',t'),M] > v$ (Line 11), and either:

(2a) $M \cap b \neq \varnothing$, or

(2b) There is no $M' \in eliminated$ s.t. $M \subseteq M'$ and $\mathcal{T} \models M \sqsubseteq \exists r.M_1$

If (2a) holds, there is some $A \in M$ s.t. $\mathcal{T} \models A \sqsubseteq \exists r.M_1$. Let $M_A = \{A\}$. By Corollary 2, it holds that $\mathcal{T} \models M \sqsubseteq \exists r.M_1$. If (2b) holds, we have $\mathcal{T} \models M \sqsubseteq \exists r.M_1$ as a direct consequence. For both cases, we obtain in combination with (1) that $\mathcal{T} \models M \sqsubseteq \exists r.M_1$ and $spa[(s,t),(s',t'),M] > v$ holds. Thus, we have $M \in T$. As M was chosen arbitrarily, it holds that $M \in T$ for all $M \in subsumees$, and therefore $subsumees \subseteq T$. □

Regarding combined complexity, we have that lines 3 and 6 require at most EXP-time. For each $M \in \mathsf{T}(\mathcal{U}_\mathcal{K})$, the complexity of executing lines 11-25 is dominated by the subsumption check required in line 20, which as well requires exponential time. We obtain that a run of *calculateSubsumees* requires at most exponential time for combined complexity.

4.3.1.2. Tracking processed *fillers*

Recall that S1-Applications are of the form

$$spa[(s,t),(s',t'),M] \ll w_1 + spa[(s_1,t_1),(s_2,t_2),M_1] + w_2$$

We refer to such entries $spa[(s_1,t_1),(s_2,t_2),M_1]$ as *fillers*. The amount of entries that need to be considered is already restricted by exploiting Lemma 3. However, the set of potential fillers can still be rather large. As the processing of each such filler requires at at least one run of *calculateSubsumees* (and thus EXP-Time), our procedure can greatly benefit from further restricting the set of potential fillers.

In lines 11-14 of procedure *S1 - $\mathcal{ELHI}_\bot$*, an additional caching optimization is employed. The idea is that, by keeping track of which sets have already processed, we can decide whether a pair $(M_1, c) \in fillers$ can actually be used to find better updates than the ones already found.

To this end, the procedure uses a relation $processed \subseteq \mathsf{T}(\mathcal{U}_\mathcal{K}) \times \mathbb{N}$ to collect all fillers that have been processed for the current combination of $(s,t), (s',t'), (s_1,t_1)$ and (s_2,t_2). We will explain how the condition in line 11 can be used to decide whether a certain filler can be skipped.

Consider a run of *Procedure S1 - $\mathcal{ELHI}_\bot$* until $processed$ contains at least one element, and let $(M_1, c) \in fillers$ be the filler that is currently considered when line 11 is reached. Assume there is $(M', v) \in processed$ s.t. $M' \subseteq M_1$ and $c = v$. By Lemma 2, it holds that $\mathcal{T} \models M_1 \sqsubseteq M'$, and consequently $\mathcal{T} \models \exists r.M_1 \sqsubseteq \exists r.M'$ for all $r \in \mathsf{N}_\mathsf{R}^\pm$.

Let $M \in \mathsf{T}(\mathcal{U}_\mathcal{K})$ and $(r, w) \in candidateRoles$. We consider two cases: First, assume $\mathcal{T} \not\models M \sqsubseteq \exists r.M_1$. Then, M_1 and r cannot be used to update $spa[(s,t),(s',t'),M]$.

Otherwise, assume $spa[(s,t),(s',t'),M]$ can be updated with value with $z_1 = c + w$ using M_1, i.e. it holds that $\mathcal{T} \models M \sqsubseteq \exists r.M_1$. Consequently, we have that $\mathcal{T} \models M \sqsubseteq \exists r.M'$ holds. It follows that $spa[(s,t),(s',t'),M]$ can be updated with value $z_2 = v + w$ using M'. We have $z_1 < z_2$ only if $c < v$. As the condition in line 11 requires $c = v$, there are no updates to $spa[(s,t),(s',t'),M]$ using M_1 and r that are better than the once already found using M' and r.

As M and (r, w) were chosen arbitrarily, this holds for all combinations of M and r. Therefore, we can see that lines 11-14 do not change the results computed by *Procedure S1 - $\mathcal{ELHI}_\bot$*.

While the correctness of this construction is independent of the order in which the sets in $fillers$ are processed, it is obvious that actual value of the optimization depends on how many sets can be skipped, and thus depends on smaller sets being processed first.

Remark 2. The order in which *Procedure S1- $\mathcal{ELHI}_\bot$* processes the sets $(M, c) \in fillers$ has an impact on moderate-case performance. The best results are obtained if the elements are processed in order of ascending size, i.e. (M, c) is processed before (M', c') iff $M \subset M'$.

The maximum amount of elements (M', c) in $processed$ is bounded by the size of $\mathsf{T}(\mathcal{U}_\mathcal{K})$. For each element, the condition in line 11 requires checking subset containment, which is possible in linear time using an appropriate encoding, as explained in chapter 5. Thus, an execution of line 11 requires at most EXP-Time in the size of $\mathfrak{T}$.
Regarding space complexity, we need to store a set $M \subset |\mathsf{Sig}(\mathcal{T}) \cap \mathsf{N_C}|$ and a constant-sized value for each element. Therefore, storing $processed$ requires at most exponential space in the size of $\mathcal{K}$.

While these upper bounds do not look promising, we expect that the actual amount of elements in $processed$ remains much lower for moderate-case scenarios. Note that, to circumvent the EXP-space requirement, an equivalent construction could be obtained by restricting the set $fillers$ after line 9 to only contain such (M, c) where M is the smallest set among all (M^*, c), i.e. using:

$$fillers' := \{(M_1, c) \mid (M_1, c) \in fillers \text{ and there is no } (M^*, c) \in fillers \text{ with } M_1 \subset M^*\}$$

Calculating this restriction is possible in exponential time without requiring additional space. We do not give details here, as we expect both variants to have comparable performance for moderate-case scenarios.

4.3.1.3. Correctness and runtime of *Procedure S1 -* $\mathcal{ELHI}_\bot$

Finally, we are ready to show that *Procedure S1 -* $\mathcal{ELHI}_\bot$ correctly computes all S1-applications as required in *Procedure SPA* for one iteration of f. Due to the correspondence between procedure *calculateSubsumees* and Line 11 of *Procedure S1 -* $\mathcal{ELH}$, the proof is almost identical to Lemma 4 and can be found in the appendix.

Lemma 9. *Let $\mathfrak{R}$ a 2RPQ, $\mathfrak{T}$ a distortion transducer, $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ an $\mathcal{ELHI}_\bot$ KB, spa a loop table and spa^* a fragment of spa containing all updated entries since the previous iteration of f. For all $M \in \mathsf{T}(\mathcal{U}_\mathcal{K})$, the following holds:*

After a run of Procedure S1 - $\mathcal{ELHI}_\bot$*, we have $spa^{**}[(s,t),(s',t'),M] = v$ iff v is the minimal[5] updated value after applying rule S1 to $((s,t),(s',t'),M)$. to be continued...*

For $\mathcal{K}$ an $\mathcal{ELHI}_\bot$ KB, procedures *filterEdges* and *filterRoles* are computable in at most exponential time in the combined size of $\mathfrak{R}, \mathfrak{T}$ and $\mathcal{K}$. The amount of entries in spa^* is at most exponential in the combined size of $Q_\mathfrak{R}, Q_\mathfrak{T}$ and $\mathcal{K}$. Each such entry requires one execution of lines 11-14 and one run of *calculateSubsumees*. Both require at most exponential time in the size of $\mathfrak{R}, \mathfrak{T}$ and $\mathcal{K}$. We obtain that a run of *Procedure S1 -* $\mathcal{ELHI}_\bot$ is feasible in at most exponential time for combined complexity.

4.3.2. Calculating S2 over $\mathcal{ELHI}_\bot$ KBs

Here, we present a procedure to calculate all S2-Applications under the presence of an $\mathcal{ELHI}_\bot$ KB. In general, this procedure is identical to *Procedure S2 -* $\mathcal{ELH}$ presented in section 4.2.2 when accounting for the difference in $\mathsf{T}(\mathcal{U}_\mathcal{K})$. However, due to the increased complexity for answering subsumption over $\mathcal{ELHI}_\bot$ KBs, we obtain an EXP-Time upper bound in combined complexity for a single application of S2.
Nevertheless, this upper bound leads us to expect that there is something to be gained for moderate-case scenarios. In similar spirit to the optimizations explained for calculating S1-Applications over $\mathcal{ELHI}_\bot$, we expect to increase moderate-case performance by applying similar optimizations. To this end, we present a dedicated subprocedure *calculateSubsumees* to calculate the set $\{M \mid M \in \mathsf{T}(\mathcal{U}_\mathcal{K}), \mathcal{T} \models M \sqsubseteq A\}$ as required in line 8 of *Procedure S2 -* $\mathcal{ELH}$. The adjusted procedure *S2 -* $\mathcal{ELHI}_\bot$ can be found in the appendix.

[5] We can assume that v is the minimum value amongst all S1-Applications to $((s,t),(s',t'),M)$

Subprocedure 8 *calculateSubsumees (S2 - $\mathcal{ELHI}_\bot$)*

Input: $A \in \mathsf{N_C} \cap \mathsf{Sig}(\mathcal{T})$
Output: A set $\{M \mid M \in \mathsf{T}(\mathcal{U}_\mathcal{K}), \mathcal{T} \models M \sqsubseteq A\} \subseteq \mathsf{T}(\mathcal{U}_\mathcal{K})$

1: initialise $subsumees := \{\}$, $eliminated := \{\}$
2: Calculate $b := \{C \mid \mathcal{T} \models C \sqsubseteq A, C \in \mathsf{N_C}\} \cup A$
3: for all $M \in \mathsf{T}(\mathcal{U}_\mathcal{K})$[6] do
4: if $M \cap b \neq \varnothing$ then
5: add M to $subsumees$
6: else
7: if $M \subseteq M'$ for some $M' \in eliminated$ then
8: Discard and continue with next M
9: end if
10: if $T \models M \sqsubseteq A$ then
11: add M to $subsumees$
12: else
13: add M to $eliminated$
14: end if
15: end if
16: end for
17:
18: return $subsumees$

[4] elements are required to be processed in a specific order, following the same idea as given in Remark 1

The procedure contains two optimizations: In line 2, all basic concepts C that satisfy $\mathcal{T} \models C \sqsubseteq A$ are retrieved, which requires at most EXP-Time in the size of $\mathcal{K}$. These are used to handle cases where C is one of the conjuncts in M. The size of $eliminated$ is bounded by $|\mathsf{T}(\mathcal{U}_\mathcal{K})|$, and hence the number of checks for $M \subseteq M'$ required in line 7 is at most exponential in the size of $\mathcal{K}$. Testing $M \subseteq M'$ is possible in linear time using an appropriate encoding, as explained in chapter 5. Checking Entailment as required in line 10 is feasible in EXP-Time for $\mathcal{ELHI}_\bot$ KBs [BBL05]. We obtain that a run of *Subprocedure calculateSubsumees* requires at most EXP-Time in the size of $\mathcal{K}$.

From the complexity results in section 4.2.2, we obtain that *Procedure S2 - $\mathcal{ELHI}_\bot$* can be considered a linear-time procedure that makes at most polynomially many calls to an EXP-Time subprocedure for combined complexity. This matches the upper bound for S2-Applications presented in B. The following lemma shows that *calculateSubsumees* computes the set $\{M \mid M \in \mathsf{T}(\mathcal{U}_\mathcal{K}), \mathcal{T} \models M \sqsubseteq A\} \subseteq \mathsf{T}(\mathcal{U}_\mathcal{K})$. As usual, we assume the presence of a fixed $(s,t), (s',t') \in (\delta_\mathfrak{R} \times \delta_\mathfrak{T})^2$.

Lemma 10. *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ an $\mathcal{ELHI}_\bot$ KB and $A \in \mathsf{N_C} \cap \mathsf{Sig}(\mathcal{T})$. A run of Subprocedure calculateSubsumees with input A calculates the set $\{M \mid \mathcal{T} \models M \sqsubseteq A\}$.*

Proof. To shorten notation, let $subsumees$ be the set returned by *Procedure calculateSubsumees* and $T = \{M \mid \mathcal{T} \models M \sqsubseteq A\}$.

$subsumees \supseteq T$: Let $M \in T$, i.e. $\mathcal{T} \models M \sqsubseteq A$. Then, either M was added to $subsumees$ in line 5, or the procedure continues to line 7. Assume there is $M' \in eliminated$ s.t. $M \subseteq M'$. This implies that $\mathcal{T} \not\models M' \sqsubseteq A$. Let I be an interpretation. By applying semantics, it holds that $M' \not\subseteq A^I$, i.e. there is some $e \in M'$ s.t. $e \notin A^I$. From Proposition 3, we obtain $M' \subseteq M$. Then, the same e exists in M, which is a contradiction to $\mathcal{T} \models M \sqsubseteq A$. Consequently, there can be no such M', and M is added to $subsumees$ in line 11.

$subsumees \subseteq T$: Let $M \in subsumees$. If M was added to $subsumees$ in line 5, there is some $C \in M$ s.t. $\mathcal{T} \models C \sqsubseteq A$. Let $M_c = \{C\}$. Obviously, we have $\mathcal{T} \models M_c \sqsubseteq A$ and $M_c \subseteq M$. By Proposition 3, it holds that $\mathcal{T} \models M \sqsubseteq M_c$, and hence $\mathcal{T} \models M \sqsubseteq A$. Consequently, we have $M \in T$.

Otherwise, M was added to $subsumees$ in line 11. Thus, it holds that $\mathcal{T} \models M \sqsubseteq A$, and hence $M \in T$. □

Given that the only difference between the procedures *Procedure calculateS2 -* $\mathcal{ELH}$ and is *Procedure calculateS2 -* $\mathcal{ELHI}_\bot$ is how the set $subsumees$ is computed at line 8, it is easy to see that the proof for Lemma 5 can be adopted for *Procedure S2 -* $\mathcal{ELHI}_\bot$ using Lemma 11.

Lemma 11. *Let $\mathfrak{R}$ a 2RPQ, $\mathfrak{T}$ a distortion transducer, $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ an $\mathcal{ELHI}_\bot$ KB. Let spa^* be the state of spa after Procedure SPA finishes line 2. For all $M \in \mathsf{T}(\mathcal{U}_\mathcal{K})$, the following holds:*
After a run of Procedure S2 - $\mathcal{ELHI}_\bot$, we have $spa^{**}[(s,t),(s',t'),M] = w$ iff $spa^*[(s,t),(s',t'),M]$ was updated with w after applying S2 to $((s,t),(s',t'),C)$.

4.3.3. Calculating S3 over $\mathcal{ELHI}_\bot$ KBs

In section 4.2.3, a *Procedure S3* is presented to compute the same results as obtained from an exhaustive application of rules S3. As Lemma 6 and Lemma 7 hold for both $\mathcal{ELH}$ and $\mathcal{ELHI}_\bot$ KBs, the same procedure can be used for both DLs. We note that the complexity of building the Graphs G_C and finding the cost of all shortest paths depends only on the size of $\mathfrak{R}$ and $\mathfrak{T}$. However, given the presence of an $\mathcal{ELHI}_\bot$ KB, the size of $\mathsf{T}(\mathcal{U}_\mathcal{K})$ is exponential in the size of Sig($\mathcal{T}$). Thus, we obtain that a single run of *Procedure S3* requires exponential time in the combined size of $\mathfrak{R}$, $\mathfrak{T}$ and $\mathcal{K}$.

4.4. Calculating spa

Finally, we are ready to present our final algorithm to compute the entire loop table spa using *Procedures S1, S2* and *S3*. Depending on whether an $\mathcal{ELH}$ or $\mathcal{ELHI}_\bot$ KB is used, the respective variants of *S1* and *S2* have to be used.

The general idea follows the nondeterministic *Procedure SPA* from [FT21]. Additionally, the algorithm exploits Lemma 3 to ensure that the input to *Procedure S1* is the minimal fragment spa^* required to compute the correct results. This is made possible by the fact that *Procedures S1* and *S3* keep track of which entries of spa have been updated during their respective runs. We present the entire algorithm first before continuing to show the results are correct.

Algorithm 1 Construction of spa

Input: An $\mathcal{ELH}$ or $\mathcal{ELHI}_\bot$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, a 2RPQ $\mathfrak{R}$, a dt $\mathfrak{T}$
Output: loop table spa

1: initialize empty table spa^*
2: $spa := S2()$
3: $spa^{**} := S3(spa)$
4: update spa with spa^{**}
5: set $spa[(s,t),(s,t),C] := 0$ for all $((s,t),C) \in (\delta_\mathfrak{R} \times \delta_\mathfrak{T}) \times \mathsf{T}(\mathcal{U}_\mathcal{K})$
6: set $spa^* := spa$
7: **repeat**
8: $\quad (spa, spa^*) := f(spa, spa^*)$
9: **until** spa^* is empty
10: **return** spa

function f
$spa^* := S1(spa^*)$
update spa with spa^*
$spa^{**} := S3(spa)$
update spa and spa^* with spa^{**}
return (spa, spa^*)
end function

Expressions of the form *"update spa with spa^* "*, where spa^* is a partial loop table, have the following meaning: For every entry $spa^{**}[(s,t),(s',t'),C]$ with value $v \in \mathbb{N}$, update $spa[(s,t),(s',t'),C]$ with v. The algorithm uses *Procedure S2* to calculate the initial state of spa, which is then gradually updated. In addition, a partial table spa^* is used to keep track of updates to spa calculated during one iteration of f, and is used as input to *Procedure S1*.

Lines 5 and 6 guarantee that, for the first iteration of f, all entries $spa^*[(s,t),(s,t),C] = 0$ are present. These can never be updated (and thus need not to be considered as input for *Procedure S1*), but must be present during the first execution of *Procedure S1*[7].

Given an $\mathcal{ELH}$ KB $\mathcal{K}$, a 2RPQ $\mathfrak{R}$ and a distortion transducer $\mathfrak{T}$, we can use Lemma 4, Lemma 5 and Lemma 7 to show the equivalence between calls to procedures *S1, S2 and S3* and the corresponding calculations required in *Procedure SPA*. Under the presence of an $\mathcal{ELHI}_\bot$ KB, the same correspondence can be found using Lemma 9, Lemma 11 and Lemma 7. It is not hard to see that *Algorithm 1* computes the same results as *Procedure SPA*.

Lemma 12. *Let $\mathfrak{R}$ a 2RPQ, $\mathfrak{T}$ a distortion transducer, $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ an $\mathcal{ELH}$ or $\mathcal{ELHI}_\bot$ KB. Then,* Algorithm 2 *computes the same results as Procedure SPA.*

Note that procedures *S2* and *S3* are executed once during the initial steps. Then, each iteration of f requires one execution of procedures *S1* and *S3*, and the maximum amount of iterations is $(|Q_\mathfrak{R}| \cdot |Q_\mathfrak{T}|)^2 \cdot |\mathsf{T}(\mathcal{U}_\mathcal{K})|$. For $\mathcal{ELH}$, a run of procedures *S1, S2* and *S3* is feasible in at most polynomial time for combined complexity, and the size of $\mathsf{T}(\mathcal{U}_\mathcal{K})$ is polynomial in the size of $\mathcal{K}$. Thus, for $\mathcal{ELH}$, we obtain that *Algorithm 1* is computable in at most polynomial time in combined complexity.

For $\mathcal{ELHI}_\bot$, a run of procedures *S1*, *S2* and *S3* requires exponential time, and the size of $\mathsf{T}(\mathcal{U}_\mathcal{K})$ is exponential in the size of $\mathcal{K}$. In summary, we obtain that a run of *Algorithm 1* requires at most exponential time for combined complexity.

[7] Our implementation works slightly different and never actually stores these entries, but uses an equivalent way to process them during the first iteration.

Together with Lemma 12 we obtain that *Algorithm 1* is a deterministic procedure that correctly calculates spa. As a side result, we obtain an upper bound for combined complexity to calculate spa over $\mathcal{ELHI}_\bot$ KBs.

Lemma 13. *Given a 2RPQ $\mathfrak{R}$, a distortion transducer $\mathfrak{T}$, and an $\mathcal{ELHI}_\bot$ KB $\mathcal{K}$, the relation spa is computable in at most exponential time for combined complexity.*

4.5. calculating sp

With the construction of spa in place, we can proceed to show how the table sp can be obtained. Recall that sp is used to store information on available paths in $G_{\mathcal{U}_\mathcal{K}}$, starting and ending at nodes with same $\mathcal{A}$-Box individual.

This information is used to construct a graph $G^*_{\mathcal{U}_\mathcal{A}}$, which finally can be used to answer approximate queries by finding shortest paths in this graph. We assume the reader is familiar with the general procedure, and omit giving an explanation here. A detailed discussion on how the table sp and the graph $G^*_{\mathcal{U}_\mathcal{A}}$ are used to answer approximate queries can be found in [FT21] and Appendix B.

Note that there is a slight difference on how the tables sp are designed depending on whether an $\mathcal{ELH}$ or $\mathcal{ELHI}_\bot$ KB is considered. For an $\mathcal{ELH}$ KB, sp is a relation with entries of the form

$$sp[(s,t),(s',t'),C]$$

where $(s,t),(s',t') \in (\delta_\mathfrak{R} \times \delta_\mathfrak{T})^2$ and $D \in \mathsf{T}(\mathcal{U}_\mathcal{K})$. Each such an entry is associated with a value $v \in \mathbb{N} \cup \infty$, and represents the following information: If, for an individual $a \in \mathsf{N_I}$, it holds that $D(a)$, there is path of minimal cost from (s,t,a) to (s',t',a) in $G_{\mathcal{U}_\mathcal{K}}$ with cost v that does not visit any $(\_,\_,a)$ expect for the first and last node.

The value for such an entry is obtained using an expression:

$$sp[(s,t),(s',t'),C] \ll w_1 + spa[(s_1,t_1),(s_2,t_2),A] + w_2, \text{ if } C1^* \text{ holds}$$

We notice that this expression is identical to rule S1 as used for the construction of spa.

For $\mathcal{ELHI}_\bot$, a slightly different form of sp is introduced in Appendix B[8]. Here, entries are of the form

$$sp[(s,t),(s',t'),a]$$

where $(s,t),(s',t') \in (\delta_\mathfrak{R} \times \delta_\mathfrak{T})^2$ and $a \in \mathsf{Ind}(\mathcal{A})$. Again, each entry is associated with a value $v \in \mathbb{N} \cup \infty$. Very similar to the case above, such an entry represents the information that there is path of minimal cost from (s,t,a) to (s',t',a) in $G_{\mathcal{U}_\mathcal{K}}$ with cost v that does not visit any $(\_,\_,a)$ in between.

[8] The reason is that the construction presented in Appendix B was based on a previous version of [FT21], and the representation of sp was changed later on. However, it is not hard to see that both representations can be transformed into each other.

Here, the value stored in $sp[(s,t),(s',t'),a]$ corresponds to the minimum value obtained from an expression of the form $w_1 + spa[(s_1,t_1),(s_2,t_2),M_1] + w_2$ s.t. condition $C2^*$ is satisfied. Again, we notice that this is almost identical to applications of rule S1 for $\mathcal{ELHI}_\bot$. The only difference is that condition $C2^*$ is used, which is the variant of $C2$ where we additionally require $\mathcal{K} \models M(a)$.

Due to the correspondence between applications of rule S1 and the expressions to calculate the values in sp, it is not hard to see that sp can be constructed for both $\mathcal{ELH}$ and $\mathcal{ELHI}_\bot$ using slightly adapted versions of procedures *S1* - $\mathcal{ELH}$ and *S1* - $\mathcal{ELHI}_\bot$ respectively. The adjusted procedures *SP* - $\mathcal{ELH}$ and *SP* - $\mathcal{ELHI}_\bot$ can be found in the appendix.

Therefore, the construction of sp requires only two steps: First, calculate spa using Algorithm 2. Then, calculate sp by running *Procedure SP* with input spa once.

Algorithm 2 Construction of sp

Input: An $\mathcal{ELH}$ or $\mathcal{ELHI}_\bot$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, a 2RPQ $\mathfrak{R}$, a dt $\mathfrak{T}$
Output: relation sp
1: calculate spa using *Algorithm 1*
2: calculate sp using *Procedure SP*
3: return sp

For $\mathcal{ELH}$, a run of *Algorithm 1* requires at most polynomial time in the size of $\mathfrak{R}$, $\mathfrak{T}$ and $\mathcal{K}$, and a run of *Procedure SP* is as well feasible in polynomial time. For $\mathcal{ELHI}_\bot$, the complexity for both steps increases to EXP-time.

Lemma 14. *For $\mathcal{K}$ an $\mathcal{ELH}$ (resp. $\mathcal{ELHI}_\bot$) KB,* Algorithm 2 *computes sp in at most polynomial (exponential) time for combined complexity.*

This concludes this chapter. Once we have sp in place, the construction of $G_{\mathcal{U}_\mathcal{A}}$ and $G^*_{\mathcal{U}_\mathcal{A}}$ used in our implementation is a straight-forward adoption of the procedure explained in [FT21]. The reasoning problems that can be answered by our implementation using $G^*_{\mathcal{U}_\mathcal{A}}$ are explained in the next chapter.

5. Implementation

In this chapter, we discuss the implementation of the algorithms presented in chapter 4. Our final implementation is called Tin*DL* and can be seen as an extension of the work presented in a preceeding masters thesis. The software is implemented using the programming language Kotlin, which provides full interoperability with existing Java sources. It is provided as a stand-alone application using a MySQL database and Spring Boot as framework. The entire stack of services required to run the application is provided as a docker image.

5.1. Overview of the implementation

This section will give an overview of our implementation by explaining the overall structure and the way different components interact with each other. To this end, we will start by explaining how input files are processed and handed to the backend logic services.

5.1.1. External frameworks

5.1.1.1. OWL API

To process OWL ontologies, our implementation uses the OWL API[1] [HB11] in the version 5.5.0. The OWL API framework is being released under both LGPL and Apache licencing, allowing the usage and integration of the source code into our application. It is one of two major frameworks for working with OWL Semantics, the other being Apache Jena[2], which as well is an open-source Java implementation.
Development of the OWL API goes back all the way to 2003, and was originally developed during the WonderWeb project at University of Manchester [BVL04]. Being an open-source project, current releases have seen contributions by various developers from different universities and nationalities, including the University of Ulm, known for their participating in the development of reasoner such as HermiT and ELK.
Our implementation makes heavy use of the data structures and methods provided by the framework, such as retrieving entities included in the ontology, building OWL expressions and accessing the employed reasoner.

[1] `https://github.com/owlcs/owlapi`
[2] `https://jena.apache.org/`

5.1.1.2. OWL2 Reasoners

Processing of the DL reasoning tasks introduced in chapter 2, such as *answering subsumption* and *calculating subsumees* is achieved by employing an OWL Reasoner. While a number of different reasoner implementations exist, not all of them see active development to remain compatible with current releases of the OWL API [Abi23]. Here, we give a short overview of some reasoners that were successfully employed within our application:

- HermiT : HermiT [Gli+14] is an OWL2 Reasoner originally developed at the University of Oxford and University of Ulm in 2014. It uses a hypertableau calculus ([MSH09]) and is able to handle all features of the OWL2 specification. Unfortunately, the project is not seeing active development anymore. The last official release is from 2013 and provides compatibility with OWL API 3.4.3. However, there exist some more forks which aim to provide compatibility with more recent versions of the OWL API. The most recent version found are maintained in a closed-source repository and released to MavenCentral. The most recent version 1.4.5.519 was released in 2020, and we were successfully able to use this version alongside OWL API 5.5.0 within our implementation.

- ELK: The ELK Reasoner [KKS14] is being developed at the University of Ulm and is written in Java. It is under ongoing development and currently supports reasoning over a fragment of the OWL2 EL[3] profile. The aim of the project is to eventually cover the full OWL2 EL profile. Current releases are maintained in an open-source repository alongside a comprehensive documentation on the current state of development.

  ELK is compatible with recent versions of the OWL API, and we were successfully able to include the most current release `elk-owlapi 0.6.0` alongside OWL API 5.5.0 within our implementation. Unfortunately, ELK currently does not support all reasoning tasks required by our implementation, and therefore cannot be used to actually obtain correct results. However, we hope that the missing features will be added to ELK at some point, and thus have included support for ELK in our implementation.

Evaluation of OWL2 reasoners has been subject to a numerous amount of publications in the past years. It is beyond the scope of this work to evaluate the performance of our implementation using different reasoners. However, one can expected that a reasoner designed specifically for tractable extensions of $\mathcal{EL}$ (corresponding to the OWL2 EL profile), such as *ELK*, provides better performance when dealing with $\mathcal{ELH}$ KBs as opposed to reasoners that are capable of handling intractable extensions or the full OWL2 Profile, such as HermiT .

Within the current version of *TinDL*, accessing the services provided by such a reasoner is done via the generalized OWL Reasoner Interface provided by the OWL API. Thus, our implementation is agnostic of the actual reasoner being used. We want to point out that our implementation has no way to determine the correctness of the results provided by the reasoner. Thus, it is imperative that a suitable reasoner is chosen to guarantee finding correct results. Further, we noticed there are slight differences in how the interfaces are implement by HermiT and *ELK* (mainly due issues in ELK), but were able to compensate using a dedicated wrapper class when using *ELK*. We expect similar effort would be required in order to use other reasoners.

[3] `https://github.com/liveontologies/elk-reasoner/wiki/OwlFeatures`

5.2. Input data

As the purpose of this implementation is to answer approximate 2RPQs over DL KBs, it naturally processes 3 types of inputs: 2RPQs, Distortion Transducers, and DL KBs, the latter given by the means of ontologies according to the OWL2 Standard. RPQs and Transducers are expected to be given as input files using a specific format, as described below.

5.2.1. Regular Path Queries and Distortion Transducers

RPQs and Transducers are expected to be given as text files using a specific mixture of keywords and DL vocabulary. The input format is designed to represent the form of finite automatons. Note that, while giving 2RPQs using regular expressions might be desirable depending on the use case, this is currently not supported by our implementation.

- file extension and encoding: The expected file format is an UTF-8 encoded text file. Supported file extensions are either `.txt` or the dedicated `.tinput`.
- functional keywords - `nodes`, `edges`: These terms mark the beginning of a section containing the respective data.

Such an input file contains two sections, each opened using the respective keyword. The section `nodes` contains all nodes present in the query automaton. Each such node is separated by a new line, and each such line must comply to the following format:

```
(node identifier), (initial node?), (final node?)
```

where:

- `node`: a string representing the unique node identifier, consisting of at least 1 character.
- `initial node?`: a string being either: `true` if the node is an initial node, `false` otherwise
- `final node?`: a string being either: `true` if the node is a final node, `false` otherwise

Accordingly, the section `edges` contains all edges present in the automaton. The format of these lines depends on whether the input file represents an RPQ or a transducer.

2RPQs:

2RPQ edges are given by lines of the following format:

```
(sourceNode), (targetNode), (label)
```

where `sourceNode` and `targetNode` are node identifiers introduced in the `nodes` section, and `label` is a string representing either a *Concept assertion* from $\{A? \mid A \in \mathsf{N_C}\}$ or a *DL role name* from $\mathsf{N_R^{\pm}}$. Inverse roles are stated by using `inverse(r)` with $r \in \mathsf{N_R}$. An example is given in figure 5.1.

```
nodes
s0, true,  false
s1, false, false
s2, false, true
edges
s0, s1, serves
s1, s1, Vegan?
s1, s2, inverse(serves)
```

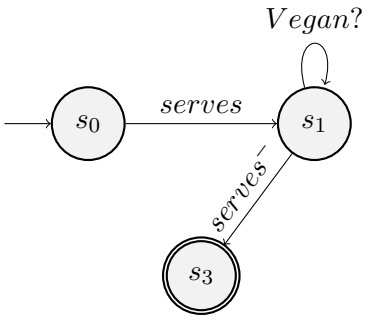

Figure 5.1.: Example 2RPQ input file and its automaton representation q_1

```
nodes
t0, true, true
edges
t0, t0, Vegan?, Vegan?, 0
t0, t0, serves, contains, 2
t0, t0, inverse(serves), Vegan?, 5
```

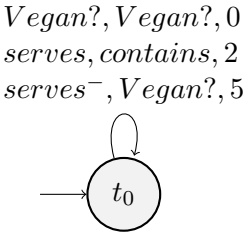

Figure 5.2.: Example transducer input file and its automaton representation T_1

Distortion Transducers:

Transducer edges differ from 2RPQ edges in the way that they use an additional outputLabel as well as the distortion cost:

```
(source node), (target node),
(inputLabel), (outputLabel), (distortionCost)
```

Here, `inputLabel` and `outputLabel` are strings in a similar format as `label` is for query edges. Additionally, `distortionCost` is an Integer value representing the distortion cost. An example is given in figure 5.2.

5.2.2. OWL2 Ontologies

As means to process DL KBs, OWL Ontologies according to the OWL2 specification [Gra+08] are used. The information stored in such an ontology is access using the OWL API. *TinDL* supports all (even optional) syntax variants as specified for OWL2[4]. The following file formats are supported:

- RDF/XML (`.rdf`): The standard file format for OWL2 Ontologies. Support for this extension which support is mandatory for all OWL2-compatible services.
- OWL/XML (`.owl`): A different XML specification aimed to provide easier processing using XML tools[5]
- OBO format (`.obo`): A syntax format developed by the Open Biological and Biomedical Ontologies (OBO) Foundry, specifically designed for the biomedical domain. The format is compatible with other OWL2 Syntax specifications ([Tir+11]).

[4] `https://www.w3.org/TR/owl2-overview/`
[5] `https://www.w3.org/TR/2012/REC-owl2-xml-serialization-20121211/`

- Manchester Functional Syntax (`.omn`): A representation using Manchester Syntax [6], a user-friendly syntax designed to be easily read- and writeable by humans.
- Functional-styl Syntax (`.ofn`): A syntax variant aimed to allow expressions that resemble the structural specification of OWL2 Ontologies[7].
- Turtle (`.ttl`): Terse RDF Triple Language - an optional syntax specification for OWL Ontologies[8].

Our implementation uses IRIs (Internationalized Resource Identifiers, see IETF Standard RFC3987[9]) to uniquely identify ontologies and their contained entities. Most ontologies follow the proposed naming conventions for OWL 2, using entity IRIs that usually consist of two parts: The base ontology IRI, and some suffix denoting the entities unique identifier within the ontology. The suffix is separated from the ontology IRI by either the symbol # (`ontologyIRI#suffix`), or is the final sequment of a path-like IRI (`ontology/iri/suffix`). Examples for such IRIs are `http://snomed.info/id/58222006` (SNOMED) or `http://swat.cse.lehigh.edu/onto/univ-bench.owl#Professor` (LUBM).

While entity IRIs are required to be unique within an ontology, they are usually quite long and rather inconvient to handled by an user. In order to allow a human-friendly vocabulary for the input query and transducer, our implementation uses a specific *short form* of an IRI to find matching terms within the query and transducer vocabulary. Usually, this short form corresponds to the suffix explained above. The exact way how these short forms are generated depends on the entities IRI, and is explained in the OWL API documentation[10]. Note that these short forms are guaranteed to be unique only if all entities within an ontology use an appropriate naming convention. Our implementation does not verify the uniqueness of short forms. In order to avoid ambiguity, the user must make sure the input ontology uses appropriate entity IRIs.

These short forms could still lead to unexpected results if when importing multiple ontologies. To prevent this, *TinDL* does not use the import closure of the provided ontology. This is not a restriction, but it does require the user to provide ontology files that do not rely on imports. This, for example, can be done by using Protegé to merge all imports into one ontology and saving the result as a new file.

5.3. Data modeling and representation

In this section, we will discuss how certain data structures are modeled within the implementation and explain why the given representation was chosen.

5.3.1. Conjunctions of concept names

The presence of an $\mathcal{ELHI}_\bot$ KB requires dealing with conjunctions of concept names, which usually are represented using sets of concept names. The size of these sets is limited only by the amount of concept names present in the KB. In order to deal with such sets efficiently, our implementation uses a binary representation. This is done by assigning a unique index

[6] `https://www.w3.org/TR/owl2-manchester-syntax/`
[7] `https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/`
[8] `https://www.w3.org/TeamSubmission/turtle/`
[9] `https://www.ietf.org/rfc/rfc3987.txt`
[10] *TinDL* uses an instance of SimpleShortFormProvider, see `http://owlcs.github.io/owlapi/apidocs_5/org/semanticweb/owlapi/util/SimpleShortFormProvider.html`

to each concept name, and using a binary array to denote, for each index, whether the concept name is contained within the set.

The following example shows how these sets are implemented:
Let $K = (\mathcal{T}, \mathcal{A})$ be an $\mathcal{ELHI}_\bot$ KB with $|Sig(\mathcal{T}) \cap \mathsf{N_C}| = n$, i.e. there are n distinct concept names present in $\mathcal{T}$. Assign to each concept name c a unique, fixed value $i(c)$ from the interval $[0, n)$ s.t. $i(c) \neq i(c')$ for all $c, c' \in Sig(\mathcal{T}) \cap \mathsf{N_C}$.
Let $M \subseteq (Sig(\mathcal{T}) \cap \mathsf{N_C})$. Now, let $b = (0)^n$ be a n-ary array of binary values and $b(k)$ denote the value at the k-th position ($0 \leq k < n$). Now, for each $c \in M$, set $b(i(c)) = 1$. By interpreting b as a binary number, we obtain a value $v(b) \in \mathbb{N}, v(b) < 2^n$ that uniquely represents the set.

Our implementation uses the Kotlin **ULong** data type, a primitive 64-bit unsigned Long Integer implementation that supports bitwise operations. While we note that this restricts the amount of concept names to 64 when processing $\mathcal{ELHI}_\bot$ KBs, this restriction is situated well above other limitations on amount of concept names for $\mathcal{ELHI}_\bot$ KBs that can reasonably be processed by our implementation, as explained in section **??** *INSERT REFERENCE*.

The binary representation of these sets features two major advantages for our implementation: Any conjunction can be saved using only 64 bits of memory, while at the same time allowing for very efficient checking of equality and containment between two such sets: Assuming a 64-bit JVM runtime is used to run *TinDL*, checking whether $M \subseteq M'$ using their binary representation requires only a single bitwise comparison and is thus feasible in constant time. The latter is heavily used during the construction of the loop tables.

For $\mathcal{ELH}$ KBs, concept names can be represent in a similar fashion by simply using their index value. It is more than enough to store these using 32-bit integers. This way, a theoretical amount of 2^{32} different concept names can be handled - a limit well beyond any currently existing ontology that we are aware of.

5.3.2. Loop tables

For the construction of sp and spa as explained in chapter 4, our implementation needs to store all information contained within these tables during the entire answering process. Considering that the amount of entries in spa can be very large, especially for $\mathcal{ELHI}_\bot$ KBs, the representation of this data is a crucial aspect of the implementation.

For both $\mathcal{ELH}$ and $\mathcal{ELHI}_\bot$, the tables are stored using the Kotlin HashMap implementation. This is an unordered collection of Key-Value pairs, where the keys are referenced by a numeric hash value, and the value is a 32-bit integer. Recall that an entry in spa is a tuple $((s,t),(s',t'),C)$, where s, s', t, t' are states from $\mathfrak{R}$ resp. $\mathfrak{T}$, and C is either a concept name (for $\mathcal{ELH}$) or a conjunction of concept names (for $\mathcal{ELHI}_\bot$). States from $\mathfrak{R}$ and $\mathfrak{T}$ have unique string identifier obtained from the input file, and C uses a unique, binary representation. We use a 32-bit hash values, computable in constant time, to encode each of these components, and the sum of these hash values to reference a table entry.

It is important to note that HashMaps in Kotlin are *collision-save*, i.e. in case there are multiple keys with the same hash value, a specific comparison function is used to decide which key was referenced when accessing the map. Our implementation makes sure there is such a function to correctly discriminate all entries. This comparison uses the unique string iden-

tifiers for query and transducer states, and the binary representation of the tails. Therefore, accessing an entry in spa only requires constant time, independent of the number of entries. This is an important property of our implementation in order to respect the upper bounds for combined complexity obtained for the construction of sp and spa.

Considering the vast amount of read and write operations to the entries in spa required, using external memory to store spa is not an option. Unfortunately, this leads to the amount of entries in spa being the main limiting factor for scalability of our implementation. Assuming a 64-bit JVM runtime and a generous amount of 16GB of memory, we can quickly determine an upper bound on the size of spa that can be stored:

Recall that the amount of entries in spa is $|Q_{\mathfrak{R}}| \cdot |Q_{\mathfrak{T}}| \cdot |\mathsf{T}(\mathcal{U}_{\mathcal{K}})|$. For each entry in spa, we need to store a total of 64 bits (8 bytes). Using a maximum of 16GB of memory, the maximum amount of values that can be stored is approximately 2^{31}.

Now, consider an example 2RPQ and transducer with 4 states each. Then, the maximum size of $\mathsf{T}(\mathcal{U}_{\mathcal{K}})$ that can be stored is 2^{27}. For an $\mathcal{ELHI}_{\bot}$ KB, the size of $\mathsf{T}(\mathcal{U}_{\mathcal{K}})$ corresponds to $2^{|\mathsf{Sig}(\mathcal{T}) \cap \mathsf{N_C}|}$ - leaving us with a maximum of 27 different concept names in our KB. Looking at existing ontologies, one quickly realizes that this restriction severely limits the use cases of our implementation.

The Leight University Benchmark Ontology (LUBM) [GPH05] is one of the smallest representatives of $\mathcal{ELHI}_{\bot}$ ontologies frequently used for benchmarking and evaluating OWL tools. The original version of LUBM contains 43 different class names. For *TinDL* to be able to handle this using our 4-state query and transducer, one would, in theory, require at least 1.5 Exabyte of memory. This absurd example shows a severe limitation of the approach presented in this work when dealing with $\mathcal{ELHI}_{\bot}$ KBs.

The situation looks far more promising when looking at $\mathcal{ELH}$ KBs. Here, the size of $\mathsf{T}(\mathcal{U}_{\mathcal{K}})$ is limited by the amount of concept names in the $\mathfrak{T}$-Box. The famous *SNOMED* ontology, one of the largest ontologies present and often regarded as the ultimate challenge when evaluation OWL tools, currently contains approx. 360.000 classes. If we consider again a limit of 16GB of memory, this leaves us with a factor of 6000 for $|Q_{\mathfrak{R}}| \cdot |Q_{\mathfrak{T}}|$. Looking at current consumer-level hardware, we therefore argue that the memory requirement to store spa is unlikely to be a limiting factor in terms of scalability when dealing with $\mathcal{ELH}$ KBs.

5.4. Description of the answering process

The answering procedure is initiated by providing the input files along with a *run configuration* containing additional input data. *TinDL* provides a REST-API for uploading input files and submitting run configurations. Documentation on how to use the provided endpoints and how to submit *run configurations* is included with the source code.

After parsing the input query and transducer files, the ontology is loaded using the OWL API framework. Along with loading the ontology, a set of essential services for processing the ontology data are initialized. This includes the OWL Reasoner, embedded in a caching utility which allows us to reuse reasoning results and to incrementally calculate the role hierarchy using a pay-as-you-go principle. This utility also supports 'prewarming' the reasoning cache, offering a significant performance improvement at the expense of preliminary processing time and memory usage.

Once the ontology is loaded, the tables spa and sp are calculated using an implementation of the algorithms explained in chapter 4. Subsequently, a graph structure representing $G^*_{\mathcal{U}_\mathcal{A}}$ is constructed using the information in sp. The construction process differs slightly depending on whether the ontology conforms to $\mathcal{ELH}$ or $\mathcal{ELHI}_\bot$, but in both cases, the resulting graph is uniform and independent of the underlying description logic.

Finally, the graph $G^*_{\mathcal{U}_\mathcal{A}}$ is used to obtain the appropriate results. The final step depends on type of reasoning problem that was specified in the *run configuration*. In the following, it is described how $G^*_{\mathcal{U}_\mathcal{A}}$ is used to answer the reasoning problems defined in section 3.3.

- *cost computation*: Answering the *cost computation problem* is achieved by running a single-source Dijkstra algorithm on the graph structure $G^*_{\mathcal{U}_\mathcal{A}}$. Two identifiers a, b are obtained from the *run configuration*. The set of starting nodes consist of all nodes (s, t, a) in $G^*_{\mathcal{U}_\mathcal{A}}$ s.t. $s \in I_\mathfrak{R}$ and $t \in I_\mathfrak{T}$. Accordingly, the set of final nodes consists of all nodes (s', t', b) s.t. $s' \in F_\mathfrak{R}$ and $t' \in F_\mathfrak{T}$. The answer is obtained by running a single-source Dijkstra implementation from each starting node to each final node. The minimum value obtained from such a run is returned as result.

- *τ-entailment*: For answering τ-entailment, the run configuration is required to contain two identifiers a, b and a threshold value μ. The result is obtained in a similar way to the cost computation problem, but the computation is stopped as soon as a path from (s, t, a) to (s', t', b) with a cost lower than μ is found. The result is a boolean value indicating whether such a path was found.

- *query answering*: To compute all certain approximate answers, a Floyd-Warshall implementation is used to find the cost of a shortest path between all nodes in $G^*_{\mathcal{U}_\mathcal{A}}$. Then, a filtering operation is used to find all tuples $(a, b, \eta_{a,b})$ s.t. $a, b \in \mathrm{Ind}(\mathcal{A})$ and $\eta_{a,b}$ is the minimum cost of a path from some (s, t, a) to some (s', t', b) s.t. $s \in I_\mathfrak{R}, s' \in F_\mathfrak{R}, t \in I_\mathfrak{T}$ and $t' \in F_\mathfrak{T}$. The set containing all such tuples is returned.

- *threshold query answering*: A threshold value μ is obtained from the *run configuration*. The answering procedure is similar to the *query answering problem*, but a set containing only such tuples $(a, b, \eta_{a,b})$ where $\eta_{a,b} \leq \mu$ is returned.

5.5. Testing and Evaluation

The implementation process followed the principles of Test-Driven Development (TDD). Correctness of results was validated using a set of small, hand-crafted ontologies, along with corresponding small queries and transducers. Due to the lack of any existing implementation that could be used to verify the correctness of our prototype implementation, we did manual calculations and used these to ensure the correctness of the individual steps of the answering process. However, the variety of inputs we could verify in this way remains rather limited.

We conclude this chapter by presenting the results of some experiments used to evaluate our implementation in terms of computation time required to construct spa and sp. A comprehensive evaluation covering all variants of input variables poses a very challenging task itself, and unfortunately was beyond the scope of this thesis. However, we expect that there are many ways to improve our results, and hope that further work on this topic can pave the way to more robust implementations of this kind.

We start by explaining the methodology behind our evaluation. All experiments were conducted using an $\mathcal{ELH}$ and $\mathcal{ELHI}_\bot$ variant of a small benchmark ontology (16 concept names, 54 axioms, 3 individuals). As input queries, the experiments used randomly generated 2RPQs $\mathfrak{R}$ with a fixed amount of states $Q_\mathfrak{R}$ and a fixed amount of edges $\delta_\mathfrak{R}$. As input transducers, we used automatically generated transducers with a single state representing the *word edit distance* between the string representation of the elements in the query alphabet and the concept and role names present in the ontology.

We used our internal benchmarking tool to track the time required to construct the tables spa and sp. Additionally, the amount of entries in the final table spa with values $v \neq \infty$ were tracked. The results for $\mathcal{ELH}$ are shown in Table 5.1. For $\mathcal{ELHI}_\bot$, the results are shown in table 5.2. The size of $Q_\mathfrak{R}, \delta_\mathfrak{R}$ and $\delta_\mathfrak{T}$ used as input is specified in the first 3 columns. The column 'Time spa' shows minimum, maximum and average time required to construct the table spa. Similar results for the construction of sp are shown in the next column. The final column denotes the amount of entries in contained in spa with a value other than ∞ after the construction was finished. Note that the time values in table 5.1 is given in milliseconds, whereas the time values in table 5.2 are given in seconds.

The result show the expected increase in computational effort when using the $\mathcal{ELHI}_\bot$ version of the benchmark ontology. For a random query with $\mathfrak{R}$ with $|Q_\mathfrak{R}| = 5$ and $|\delta_\mathfrak{R}| = 15$, the construction of table spa required an average of 4.73ms for $\mathcal{ELH}$, whereas an input of the same size required an average of 50.11s for $\mathcal{ELHI}_\bot$ - almost 10^4 times as long.

			Time spa (ms)			Time sp (ms)			Size spa		
$\lvert Q_\mathfrak{R}\rvert$	$\lvert\delta_\mathfrak{R}\rvert$	$\lvert\delta_\mathfrak{T}\rvert$	min	max	avg	min	max	avg	min	max	avg
5	15	80	1.11	300.15	4.73	1.49	10.69	1.49	16	320	99
5	25	120	1.85	214.97	7.06	1.61	15.41	3.53	16	320	172
5	50	172	7.57	300.93	17.38	5.17	30.89	11.14	80	320	27

Table 5.1.: Experimental results for queries of different size for $\mathcal{ELH}$. Each experiment was repeated 200 times to obtain the average values.

			Time spa (s)			Time sp (s)			Size spa		
$\lvert Q_\mathfrak{R}\rvert$	$\lvert\delta_\mathfrak{R}\rvert$	$\lvert\delta_\mathfrak{T}\rvert$	min	max	avg	min	max	avg	min	max	avg
5	15	80	35.35	77.88	50.11	1.21	8.30	5.83	783,354	1,304,568	1,251,066
5	25	120	87.71	155.26	115.31	9.04	30.21	17.21	974,921	1,399,851	1,303,035

Table 5.2.: Experimental results for queries of different size for $\mathcal{ELHI}_\bot$. Here, each experiment was repeated 10 times.

6. Conclusions

In this thesis, we have developed and implemented a practical procedure for answering Two-Way Regular Path Queries over $\mathcal{ELH}$ and $\mathcal{ELHI}_\bot$ Knowledge Bases under approximate semantics. To this end, we have shown that the approximate semantics proposed in [FT21] can be extended to cover the more expressive DL $\mathcal{ELHI}_\bot$, and have explained how to adapt the answering procedure presented therein. Based on these results, we have developed a practical algorithm that constructs the relations sp and spa, which are crucial components of the answering procedure. Our algorithm includes several optimizations to enhance performance across a range of input scenarios.

For $\mathcal{ELH}$, we have shown that our algorithm constructs sp and spa in at most polynomial time for combined complexity, and thus matches the upper bound obtained from [FT21]. For $\mathcal{ELHI}_\bot$, we already inherit EXP-Time completeness from answering 2RPQs under classical semantics [BO15], but we again obtain matching complexity results for our algorithm. Finally, we explain several details of the implementation, including how the input data is processed and how key data structures are represented.

To the best of our knowledge, our implementation is the first of its kind. Moreover, we are not aware of an implementation that supports answering 2RPQs over $\mathcal{ELH}$ and $\mathcal{ELHI}_\bot$ KBs under classical semantics. Lacking any comparable system, we were not able to evaluate our implementation in a competitive sense. Instead, we performed a basic evaluation, measuring computation times for a small benchmark ontology with varying input query and transducer automaton sizes.

Naturally, these experiments do not paint the complete picture. However, a comprehensive evaluation of the implementation poses a very challenging task itself, and was beyond the scope of this thesis. This can not be done by simply varying the size of the inputs. Due to the very complex interaction between the input data, it is not trivial to determine how much effort the single computation steps require, and how much benefit is gained from the employed optimizations. A particularly interesting task for future evaluation involves testing the implementation on 'realistic' inputs, i.e. non-trivial queries over ontologies that see practical usage. However, we must acknowledge that the implementation presented here is severely restricted in its ability to process large ontologies, mainly due to the rather large amount of memory required. This is especially the case for computing answers over $\mathcal{ELHI}_\bot$ KBs.

Finally, we hope that our implementation sees some practical usage and can act as a foundation for further improvements.

7. Bibliography

[Abi23] Konrad Abicht. “OWL Reasoners still useable in 2023”. In: *ArXiv* abs/2309.06888 (2023). url: https://api.semanticscholar.org/CorpusID:261705956.

[BBL05] F. Baader, S. Brandt, and C. Lutz. “Pushing the EL Envelope”. In: *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence IJCAI-05*. Edinburgh, UK: Morgan-Kaufmann Publishers, 2005.

[BBL08] Franz Baader, Sebastian Brandt, and Carsten Lutz. “Pushing the EL Envelope Further”. In: *In Proceedings of the OWLED 2008 DC Workshop on OWL: Experiences and Directions*. Ed. by Kendall Clark and Peter F. Patel-Schneider. 2008.

[BO15] Meghyn Bienvenu and Magdalena” Ortiz. “Ontology-Mediated Query Answering with Data-Tractable Description Logics”. In: *Reasoning Web. Web Logic Rules: 11th International Summer School 2015, Berlin, Germany, July 31- August 4, 2015, Tutorial Lectures*. Ed. by Wolfgang Faber and Adrian Paschke. Cham: Springer International Publishing, 2015, pp. 218–307. isbn: 978-3-319-21768-0. doi: 10.1007/978-3-319-21768-0_9. url: https://doi.org/10.1007/978-3-319-21768-0_9.

[BOŠ13] Meghyn Bienvenu, Magdalena Ortiz, and Mantas Šimkus. “Conjunctive regular path queries in lightweight description logics”. In: *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*. IJCAI ’13. Beijing, China: AAAI Press, 2013, pp. 761–767. isbn: 9781577356332.

[BVL04] Sean Bechhofer, Raphael Volz, and Phillip Lord. “Cooking the semantic web with the OWL API”. In: vol. 2870. July 2004. isbn: 978-3-540-20362-9. doi: 10.1007/978-3-540-39718-2_42.

[Flo62] Robert W. Floyd. “Algorithm 97: Shortest path”. In: *Commun. ACM* 5.6 (June 1962), p. 345. issn: 0001-0782. doi: 10.1145/367766.368168. url: https://doi.org/10.1145/367766.368168.

[FT21] Oliver Fernández Gil and Anni-Yasmin Turhan. “Answering Regular Path Queries Under Approximate Semantics in Lightweight Description Logics”. In: *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI’21)*. AAAI Press, 2021, pp. 6340–6348.

[Gli+14] Birte Glimm et al. “HermiT: An OWL 2 Reasoner”. In: *Journal of Automated Reasoning* 53 (2014), pp. 245–269. url: https://api.semanticscholar.org/CorpusID:15513227.

[GPH05] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. "LUBM: A benchmark for OWL knowledge base systems". In: *Journal of Web Semantics* 3.2 (2005). Selcted Papers from the International Semantic Web Conference, 2004, pp. 158–182. issn: 1570-8268. doi: https://doi.org/10.1016/j.websem.2005.06.005. url: https://www.sciencedirect.com/science/article/pii/S1570826805000132.

[Gra+08] Bernardo Cuenca Grau et al. "OWL 2: The next step for OWL". In: *Journal of Web Semantics* 6.4 (2008). Semantic Web Challenge 2006/2007, pp. 309–322. issn: 1570-8268. doi: https://doi.org/10.1016/j.websem.2008.05.001. url: https://www.sciencedirect.com/science/article/pii/S1570826808000413.

[GT05] Gösta Grahne and Alex Thomo. "Regular path queries under approximate semantics". In: *Annals of Mathematics and Artificial Intelligence* 46 (2005), pp. 165–190.

[HB11] Matthew Horridge and Sean Bechhofer. "The OWL API: A Java API for OWL ontologies". English. In: *Semantic Web* 2.1 (2011), pp. 11–21. issn: 1570-0844. doi: 10.3233/SW-2011-0025.

[KKS14] Yevgeny Kazakov, Markus Krötzsch, and František Simančík. "The Incredible ELK: From Polynomial Procedures to Efficient Reasoning with $\mathcal{EL}$ Ontologies". In: *J. Autom. Reasoning* 53.1 (2014), pp. 1–61. doi: 10.1007/s10817-013-9296-3.

[MSH09] Boris Motik, Rob Shearer, and Ian Horrocks. "Hypertableau Reasoning for Description Logics". In: *Journal of Artificial Intelligence Research* 36 (2009), pp. 165–228.

[Tir+11] Syed Tirmizi et al. "Mapping between the OBO and OWL ontology languages". English. In: *Journal of Biomedical Semantics* 2(Suppl 1).S3 (2011), pp. 1–16. issn: 2041-1480. doi: 10.1186/2041-1480-2-S1-S3.

[TRM21] Santiago Timón-Reina, Mariano Rincón, and Rafael Martínez-Tomás. "An overview of graph databases and their applications in the biomedical domain". In: *Database* 2021 (May 2021), baab026. issn: 1758-0463. doi: 10.1093/database/baab026. eprint: https://academic.oup.com/database/article-pdf/doi/10.1093/database/baab026/37952095/baab026.pdf. url: https://doi.org/10.1093/database/baab026.

[War62] Stephen Warshall. "A Theorem on Boolean Matrices". In: *J. ACM* 9.1 (Jan. 1962), pp. 11–12. issn: 0004-5411. doi: 10.1145/321105.321107. url: https://doi.org/10.1145/321105.321107.

Appendix

A. Algorithms and Proofs

A.1. Calculating rule S1 for $\mathcal{ELHI}_\bot$ KBs

A.1.1. Stage 1b: Extracting transducer edges

Similar to the $\mathcal{ELH}$ case, the first stage extracts transducer edges of the form (t, u, R', w_1, t_1) and (t_2, u', R'', w_2, t') that have a corresponding query edge. To account for the presence of inverse roles in the ontology, R' and R'' can be chosen arbitrarily from $\mathsf{N}_\mathsf{R}^\pm$.

Stage 1b: Filtering transducer edges

Input: A 2RPQ $\mathfrak{R}$, a dt $\mathfrak{T}$, $(s,t),(s_1,t_1),(s_2,t_2),(s',t') \in (Q_\mathfrak{R} \times Q_\mathfrak{T})^4$
Output: Two sorted sets $edgesDown \subseteq \delta_\mathfrak{T}$ and $edgesUp \subseteq \delta_\mathfrak{T}$

2: set $edgesDown := \{(t,u,R',w_1,t_1) \mid (s,u,s_1) \in \delta_\mathfrak{R}, (t,u,R',w_1,t_1) \in \delta_\mathfrak{T}, R' \in \mathsf{N}_\mathsf{R}^\pm\}$
3: set $edgesUp := \{(t_2,u',R'',w_2,t') \mid (s_2,u',s') \in \delta_\mathfrak{R}, (t_2,u,R'',w_2,t') \in \delta_\mathfrak{T}, R'' \in \mathsf{N}_\mathsf{R}^\pm\}$
4: sort $edgesDown$ and $edgesUp$ by w in ascending order

A.1.2. Stage 2b: Extracting feasible roles

Stage 2b: Extracting feasible roles

Input: An $\mathcal{ELHI}_\bot$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, sorted sets $edgesDown \subseteq \delta_{\mathfrak{T}}$ and $edgesUp \subseteq \delta_{\mathfrak{T}}$ from Stage 1b

Output: A set $candidateRoles \subseteq (\mathsf{N_R} \cap Sig(\mathcal{T}) \times \mathbb{N})$

6: initialise $candidateRoles = \{\}$
7: **for all** $R \in \mathsf{N_R} \cap Sig(\mathcal{T})$ **do**
8: initialise $downCost := \infty$, $upCost := \infty$
9: **for all** $(t, u, R', w_1, t_1) \in edgesDown$ **do**
10: **if** $T \models R \sqsubseteq R'$ **then**
11: set $downCost := w_1$
12: exit and continue at line 14
13: **end if**
14: **end for**
15: **if** $downCost = \infty$ **then**
16: discard and continue with next r
17: **end if**
18: **for all** $(t_2, u, R'', w_2, t') \in edgesUp$ **do**
19: **if** $T \models R^- \sqsubseteq R''$ **then**
20: set $upCost := w_2$
21: exit and continue at line 23
22: **end if**
23: **end for**
24: **if** $upCost = \infty$ **then**
25: discard and continue with next r
26: **end if**
27: add $(r, downCost + upCost)$ to $candidateRoles$
28: **end for**return $candidateRoles$

A.2. *Procedure S2 -* $\mathcal{ELHI}_\bot$

Procedure S2 - $\mathcal{ELHI}_\bot$

Input: An $\mathcal{ELHI}_\bot$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, a 2RPQ $\mathfrak{R}$, a dt $\mathfrak{T}$
Output: relation spa containing all updated entries

initialize empty table spa
for all $(s,t),(s',t') \in (\delta_\mathfrak{R} \times \delta_\mathfrak{T})^2$ do
    if $s = s'$ and $t = t'$: then
        Skip and continue with next $(s,t),(s',t')$
    end if
    set $edges := \{(t,u,A?,w,t') \mid (s,u,s') \in \delta_\mathfrak{R}, (t,u,A?,w,t') \in \delta_\mathfrak{T}, A \in \mathsf{N_C}\}$

    for all $(t,u,A?,w,t') \in edges$ do
        Calculate $subsumers := calculateSubsumers(A)$
        for all $M \in subsumers$ do
            set $spa[(s,t),(s',t'),M] := min(spa[(s,t),(s',t'),M],\ w)$
        end for
    end for
end for

return spa

A.3. Procedure SP

Procedure SP - $\mathcal{ELH}$

Input: An $\mathcal{ELH}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, a 2RPQ $\mathfrak{R}$, a dt $\mathfrak{T}$, loop table spa
Output: relation sp

initialize $sp[\mathfrak{p},\mathfrak{q},C] = \infty$ for all $(\mathfrak{p},\mathfrak{q},C) \in (\delta_\mathfrak{R} \times \delta_\mathfrak{T})^2 \times \mathsf{T}(\mathcal{U}_\mathcal{K})$
for all $\mathfrak{p},\mathfrak{q},\mathfrak{p}_1,\mathfrak{p}_2 \in (\delta_\mathfrak{R} \times \delta_\mathfrak{T})^4$ do

    set $(down, up) := filterEdges(\mathfrak{p},\mathfrak{q},\mathfrak{p}_1,\mathfrak{p}_2)$
    set $candidateRoles := filterRoles(down, up)$

    for all $(r,w) \in candidateRoles$ do
        for all $A \in \mathsf{T}(\mathcal{U}_\mathcal{K})$ do
            Calculate $subsumees := \{C \mid C \in \mathsf{T}(\mathcal{U}_\mathcal{K}), C \sqsubseteq \exists r.A\}$
            for all $C \in subsumees$ do
                if $sp[\mathfrak{p},\mathfrak{q},C] > w + spa[\mathfrak{p}_1,\mathfrak{p}_2,A]$ then
                    set $sp[\mathfrak{p},\mathfrak{q},C] := w + spa[\mathfrak{p}_1,\mathfrak{p}_2,A]$
                end if
            end for
        end for
    end for
end for

return sp

Procedure SP - $\mathcal{ELHI}_\bot$

Input: An $\mathcal{ELHI}_\bot$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, a 2RPQ $\mathfrak{R}$, a dt $\mathfrak{T}$, loop table spa
Output: relation sp

1: initialize empty relation sp
2: **for all** $(\mathfrak{p}, \mathfrak{q}, \mathfrak{p}_1, \mathfrak{p}_2 \in (\delta_\mathfrak{R} \times \delta_\mathfrak{T})^4$ **do**
3: calculate $(down, up) := filterEdges(\mathfrak{p}, \mathfrak{q}, \mathfrak{p}_1, \mathfrak{p}_2)$
4: calculate $candidateRoles := filterRoles(down, up)$
5: **for all** $a \in$ Ind$(\mathcal{A})$ **do**
6: set $v := \infty$
7: Calculate $M := \{C \mid \mathcal{K} \models C(a)\}$
8: **for all** $(r, w) \in candidateRoles$ **do**
9: initialise $eliminated := \{\}$
10: **for all** $M_1 \in \mathsf{T}(\mathcal{U}_\mathcal{K})$[1] **do**
11: **if** $v \leq w + spa[\mathfrak{p}_1, \mathfrak{p}_2, M_1]$ **then**
12: Discard and continue with next M_1
13: **end if**
14: **if** $M_1 \subseteq M'$ for some $M' \in eliminated$ **then**
15: Discard and continue with next M_1
16: **end if**
17: **if** $T \models M \sqsubseteq \exists r.M_1$ **then**
18: set $v := w + spa[\mathfrak{p}_1, \mathfrak{p}_2, M_1]$
19: **else**
20: add M to $eliminated$
21: **end if**
22: **end for**
23: **end for**
24: set $sp[\mathfrak{p}, \mathfrak{q}, a] = v$
25: **end for**
26: **end for**

[2] elements are required to be processed in a specific order, see Remark 2

A.4. Proof of Lemma 9

Lemma. *Let $\mathfrak{R}$ a 2RPQ, $\mathfrak{T}$ a distortion transducer, $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ an $\mathcal{ELHI}_\bot$ KB, spa a loop table and spa^* a fragment of spa containing all updated entries since the previous iteration of f. For all $M \in \mathsf{T}(\mathcal{U}_\mathcal{K})$, the following holds:*
After a run of Procedure S1 - $\mathcal{ELHI}_\bot$*, we have $spa^{**}[(s,t),(s',t'),M] = v$ iff v is the minimal[2] updated value after applying rule S1 to $((s,t),(s',t'),M)$.*

Proof. From section 4.3.1.2, we obtain that lines 11-14 of *Procedure S1 - $\mathcal{ELHI}_\bot$* do not change the results in spa^{**}, and therefore are not considered here.

($\Leftarrow$) Let $M \in \mathsf{T}(\mathcal{U}_\mathcal{K})$. We consider two cases: First, if $spa[(s,t),(s',t'),M]$ was not updated during an application of S1, there is no $(t, u, R', w_1, t_1), (t_2, u', R'', w_2, t') \in \delta_\mathfrak{T}$ and $M_1 \in \mathsf{T}(\mathcal{U}_\mathcal{K})$ s.t. Condition $C2^*$ is satisfied and $w_1 + spa[(s_1,t_1),(s_2,t_2),M_1] + w_2 < spa[(s,t),(s',t'),M]$.

[2] We can assume that v is the minimum value amongst all non-deterministic S1-Applications to $((s,t),(s',t'),M)$

By Lemma 2, it holds that $candidateRoles = \varnothing$, and thus spa^{**} contains no entries.

For the second case, assume $spa[(s,t),(s',t'),M]$ was updated with value v by an S1-application. Then, there is $(t,u,R,w_1,t_1),(t_2,u',R'',w_2,t') \in \delta_{\mathfrak{T}}, (s,u,s_1),(s_2,u',s') \in \delta_{\mathfrak{R}}$, $r \in \mathsf{N}_{\mathsf{R}}^{\pm}$ and $A \in \mathsf{N}_{\mathsf{C}}$ s.t. Condition $C2^*$ is satisfied, i.e. it holds that:

(i) $v = w_1 + spa[(s_1,t_1),(s_2,t_2),M_1] + w_2$,

(ii) $\mathcal{T} \models r \sqsubseteq R'$, $\mathcal{T} \models r^- \sqsubseteq R''$, and

(iii) $\mathcal{T} \models M \sqsubseteq \exists r.M_1$.

Let $c = spa[(s_1,t_1),(s_2,t_2),M_1]$ and $w = w_1 + w_2$. By Lemma 2, we have $(r,w) \in candidateRoles$. Additionally, we obtain $spa^*[(s_1,t_1),(s_2,t_2),M_1] = c$ from Lemma 3, and therefore $(M_1,c) \in fillers$. This means there is some execution of line 11 which computes the set $updates := \{N \mid \mathcal{T} \models N \sqsubseteq \exists r.M_1, spa[(s,t),(s',t'),N] > v\}$ according to Lemma 8. As v can be used to update $spa[(s,t),(s',t'),M]$, it holds that $spa[(s,t),(s',t'),M] > v$. Together with (iii), it follows that $M \in updates$. As a consequence, we obtain $spa^{**}[(s,t),(s',t'),C] \leq v$ after processing line 13.

Assume $spa^{**}[(s,t),(s',t'),M] < v$. Then, there must exists some $(s,w') \in candidateRoles$ and $(M_2,c') \in fillers$ s.t. $\mathcal{T} \models M \sqsubseteq \exists s.M_2$ and $w' + c' < v$. By Lemma 2, this implies there are $e_1 = (t,u,S',w_1',t_1)$ and $e_2 = (t_2,u',S'',w_2',t') \in \delta_{\mathfrak{T}}$ with $w' = w_1' + w_2'$ s.t. $\mathcal{T} \models s \sqsubseteq S'$, $\mathcal{T} \models s \sqsubseteq S''$, and $c' = spa^*[(s,t),(s',t'),M_2]$ s.t. $\mathcal{T} \models C \sqsubseteq \exists s.M_2$. Then, Condition $C2^*$ is satisfied using e_1, e_2 and B. Consequently $spa[(s,t),(s',t'),M]$ could have been updated with $w_1' + c' + w_2' < v$, which is a contradiction to v being the minimal updated value. It follows that $spa^{**}[(s,t),(s',t'),M] = v$.

□

B. Construction of sp and spa for $\mathcal{ELHI}_\bot$ KBs

In order to lift the concepts from ([FT21]) to the $\mathcal{ELHI}_\bot$ setting, we need to extend the definitions of sp and spa. The computation of these tables follows the general idea from the $\mathcal{EL}$ and DL_{Lite} settings, with adjustments accounting for the different domain of $\mathcal{U}_\mathcal{K}$:

Similar to the $\mathcal{EL}$ and DL_{Lite} settings, the table sp contains the minimal costs c^* of an a-path from (s,t,a) to (s',t',a) in $G_{\mathcal{U}_\mathcal{K}}$ using entries of the form $[(s,t),(s',t'),a]$, with $(s,t),(s',t') \in Q_{\mathfrak{R}} \times Q_{\mathfrak{T}}$ and $a \in \mathsf{Ind}(\mathcal{A})$. The table spa uses entries of the form $[(s,t),(s',t'),M]$, with $M \subseteq N_C$, which contain the minimal cost of an e-path from (s,t,e) to (s',t',e), where $e \in \Delta^{\mathcal{U}_\mathcal{K}} \setminus \mathsf{Ind}(A)$ and $tail(e) = M$.

By definition of U_K and G_{U_K} , an a-path from (s,t,a) to (s',t',a) in G_{U_K} must be of the form:

$$(s,t,a)\ w_1\ (s_1,t_1,aRM)\ \gamma\ (s_2,t_2,aRM)\ w_2\ (s',t',a),$$

where $(s_1,t_1,aRM)\ \gamma\ (s_2,t_2,aRM)$ is an aRM-path.

The form of such an a-path tells us that each value $sp[(s,t),(s',t'),a]$ corresponds to the minimal value of an expression of the form $w_1 + spa[(s_1,t_1),(s_2,t_2),M] + w_2$ with $M = \{C_1,...C_n\}$ a conjunction of concepts in set notation, such that the following condition is satisfied:

C1. $M \subseteq \mathsf{N}_C, \mathcal{T} \models M_0 \sqsubseteq \exists R.M\ \mathcal{K} \models M_0(a), \mathcal{T} \models R \sqsubseteq R', \mathcal{T} \models R^- \sqsubseteq R'', (s,u,s_1) \in \delta_{\mathfrak{R}}$, $(t,u,R',w_1,t_1) \in \delta_{\mathfrak{T}}$, $(s_2,u',s') \in \delta_{\mathfrak{R}}$ and $(t_2,u',R'',w_2,t') \in \delta_{\mathfrak{T}}$.

Accordingly, we present the adjusted rules for the computation of spa:

S1 $spa[(s,t),(s',t'),M] \ll w_1 + spa[(s_1,t_1),(s_2,t_2),M_1] + w_2$, if C1* holds.

S2 $spa[(s,t),(s',t'),M] \ll w$, if $\mathcal{T} \models M \sqsubseteq A$, $(s,u,s') \in \delta_{\mathfrak{R}}$, and $(t,u,A?,w,t') \in \delta_{\mathfrak{T}}$,

S3 $spa[(s,t),(s',t'),M] \ll spa[(s,t),(s'',t''),M] + spa[(s'',t''),(s',t'),M]$.

C1* is the variant of C1 with $\mathcal{K} \models M_0(a)$ omitted:

C1* $M_1 \subseteq \mathsf{N}_C, \mathcal{T} \models M \sqsubseteq \exists R.M_1, \mathcal{T} \models R \sqsubseteq R', \mathcal{T} \models R^- \sqsubseteq R'', (s,u,s_1) \in \delta_{\mathfrak{R}}, (t,u,R',w_1,t_1) \in \delta_{\mathfrak{T}}$, $(s_2,u',s') \in \delta_{\mathfrak{R}}$ and $(t_2,u',R'',w_2,t') \in \delta_{\mathfrak{T}}$.

The following procedure SPA describes how the relation spa can be build using the construction rules presented above:

Procedure SPA

Input: An $\mathcal{ELHI}_\bot$ KB $\mathcal{K} = (\mathcal{T},\mathcal{A})$, a distortion Transducer $\mathfrak{T}$ and a conjunct $\mathfrak{R}(t,t')$ with $\mathfrak{R} = (Q_{\mathfrak{R}}, \Sigma, \delta_{\mathfrak{R}}, I, F)$.

Output: Relation spa for $\mathcal{K}, \mathfrak{T}, \mathfrak{R}$

Initialize $spa[p,p,M] = 0$
Initialize $spa[p,q,M] = \infty$ (if $p \neq q$)
Apply rule S2 to all $(p,q,M) \in (Q_{\mathfrak{R}} \times Q_{\mathfrak{T}})^2 \times \mathcal{P}(Sig(\mathcal{T}))$
Apply rule S3 until spa does not change;
repeat
    $spa := f(spa)$
until spa does not change

function f
    Apply rule S1 to all $(p,q,M) \in (Q_{\mathfrak{R}} \times Q_{\mathfrak{T}})^2 \times \mathcal{P}(N_C)$
    Apply rule S3 until spa does not change;
end function

B.1. Correctness of procedure SPA

Given an $\mathcal{ELHI}_\bot$ KB K, a distortion transducer $\mathfrak{T}$ and a conjunct $\mathfrak{R}(t,t')$ of a C2RPQ, this section describes that the procedure SPA constructs the relation spa as defined. We use $G_{\mathcal{U}_\mathcal{K}}$ to denote the graph $G_{\mathfrak{R}\times\mathfrak{T}\times\mathcal{U}_\mathcal{K}}$.

Recall the definition of an e-path from ([FT21]): Let $e \in \Delta^{\mathcal{U}_\mathcal{K}}$. An e-path in $G_{\mathcal{U}_\mathcal{K}}$ is of the form $(s,t,e)\gamma(s',t',e)$ such that:

- $\gamma \in \mathbb{N}$ implies $e \in \Delta^{\mathcal{U}_\mathcal{K}} \setminus \mathsf{Ind}(A)$, and
- γ only visits vertices (s'',t'',e') such that $e' \in \Delta^{\mathcal{U}_\mathcal{K}} \setminus \mathsf{Ind}(A)$ and $e' \in T_e$.

Now, let $a \in \mathsf{Ind}(\mathcal{A})$ and π be an a-path from (s_1, t_1, a) to (s_2, t_2, a) s.t. it does not visit any $a' \in \mathsf{Ind}(\mathcal{A})$, i.e. π is a loop through the anonymous subtree rootet in a. The cost c_i of this path corresponds to the cost of all edges visited by π.

We will show that the construction rules presented here for spa are correct by expanding the corresponding lemmas from ([FT21]):

Lemma 15. *Let $(s,t),(s',t') \in Q_{\mathfrak{R}} \times Q_{\mathfrak{T}}$, $e \in \Delta^{\mathcal{U}_\mathcal{K}} \setminus Ind(A)$ and $tail(e) = M$. After SPA executes line 4, $spa[(s,t),(s',t'),M]$ contains the minimum cost c^* of an e-path of depth 0 from (s,t,e) to (s',t',e).*

Proof. The proof follows the same general idea as the proof of lemma 10 in ([FT21]).

An e-path of depth 0 is of the form:

$$(s_1, t_1, e)w_1(s_2, t_2, e)w_2...w_{n-1}(s_n, t_n, e),$$

where $n > 1, \{s,t\}_1 = \{s,t\}$ and $\{s,t\}_n = \{s',t'\}$. If $G_{\mathcal{U}_\mathcal{K}}$ does not contain a path from (s,t,e) to (s',t',e), neither S2 nor S3 can be applied. Thus, we have $spa[(s,t),(s',t'),M] = c^* = \infty$. Otherwise, let π be an e-path of depth 0 from (s,t,e) to (s',t',e) in $G_{\mathcal{U}_\mathcal{K}}$.

$\Rightarrow$: $spa[(s,t),(s',t'),M] \leq c^*$. We use induction on the length $n-1$ of π to show that

$$spa[(s_1,t_1),(s_n,t_n),M] \leq c(\pi).$$

The base case considers paths of length 1, i.e. $n = 2$. Such paths are of the form $(s_1,t_1,e)w(s_2,t_2,e)$ with cost $c(\pi) = w$. The existence of such a path in G_{U_K} implies that there is some $u \in \mathsf{N}_\mathsf{R}^\pm \cup \{A? \mid A \in \mathsf{N}_C$ and $A \in \mathsf{N}_C$ s.t. $(s_1,u,s_2) \in \delta_{\mathfrak{R}}$, $(t_1,u,A?,w,t_2) \in \delta_{\mathfrak{T}}$ and $e \in A^{U_K}$. It follows that $A \in M$ and $\mathcal{T} \models M \sqsubseteq A$. Thus, S2 can be applied to obtain $spa[(s_1,t_1),(s_2,t_2),M] \leq w$. For paths of length $n > 2$, we follow the same argument as ([FT21]) in Lemma 10: Assume that $n > 2$ and let π_1 be the sub-path $(s_1,t_1,e)...(s_{n-1},t_{n-1},e)$ and π_2 the sub-path $(s_{n-1},t_{n-1},e)w_{n-1}(s_n,t_n,e)$. The application of induction yields $spa[(s_1,t_1),(s_{n-1},t_{n-1}),M]$ $\leq c(\pi_1)$ and $spa[(s_{n-1},t_{n-1}),(s_n,t_n),M] \leq c(\pi_2)$. Hence, since S3 is applied exhaustively, it must be that $spa[(s_1,t_1),(s_n,t_n),M] \leq c(\pi)$. Thus, since π is chosen arbitrarily, it follows that $spa[(s,t),(s',t'),M] \leq c^*$.

$\Leftarrow$: $c^* \leq spa[(s,t),(s',t'),M]$. Consider an execution of line 3, followed from an exhaustive application of S3 in line 4. Note that an execution of line 3 can induce multiple updates of $spa[(s,t),(s',t'),M]$.
Let $\mu_1,..,\mu_j$ be the sequence of updates performed along the exhaustive execution of line 3, and $\mu_{j+1},..,\mu_k$ the updates performed along the execution of line 4. We assume the sequence is in order, i.e., μ_1 is the first update and μ_k is the last one. In addition, we denote as $spa[(s^i,t^i),(s'^i,t'^i),M^i]$ the entry corresponding to the update μ_i and v_i the updated value. Let now $e^i \in \Delta^{U_K} \setminus \mathsf{Ind}(A)$ such that $tail(e^i) = M^i$. We show by induction on i that there exists an e^i -path of depth 0 in G_{U_K} of the form $\pi_i = (s^i,t^i,e^i)...(s'^i,t'^i,e^i)$ such that $c(\pi_i) \leq v_i$. We consider two cases:

- $i \leq j$, i.e. μ_i corresponds to an application of S2. As an application of S2 is independent of the current loop table entries, the following holds: An application of S2 implies that $\mathcal{T} \models M^i \sqsubseteq A$, $(s^i,u,s^{i'}) \in \delta_{\mathfrak{R}}$, and $(t^i,u,A?,w,t^{i'}) \in \delta_{\mathfrak{T}}$. Since $tail(e^i) = M^i$, it holds

that $e^i \in A^{U_K}$. By definition of G_{U_K}, $(s^i, t^i, e^i), w, (s^{i'}, t^{i'}, e^i)$ is an edge in G_{U_K}. Thus, $\pi = (s^i, t^i, e^i)w(s^{i'}, t^{i'}, e^i)$ is an e^i-path of depth 0 in G_{U_K} with $c(\pi) = w = v_i$.

- $i > j$, i.e. μ_i corresponds to an application of S3. For the corresponding entry $spa[(s^i, t^i), (s'^i, t'^i), M$ we have entries $spa[(s^m, t^m), (s'^m, t'^m), M]$ and $spa[(s^n, t^n), (s'^n, t'^n), M]$ s.t. $(s'^m, t'^m) = (s^n, t^n)$, $(s^m, t^m) = (s^i, t^i)$ and $(s'^n, t'^n) = (s'^i, t'^i)$. If either of these entries was added during the execution of line 1, we would have $v_i = \infty$, which contradicts the fact that an update was performed. Thus, we can assume that $1 \leq m, n < i$, i.e. the entries were added during the previously performed applications of S2 or S3. As S3 uses the entries currently present in the loop table, we can further assume that there is no update μ_g with matching entries and $g < m, n$ s.t. $v_g \leq v_m$ or $v_g \leq v_n$. Thus, we have $v_i = v_m + v_n$ the updated value. By induction, there are e^i-paths π_1 and π_2 of depth 0 in $G_{\mathcal{U}_\mathcal{K}}$ of the form $(s^i, t^i, e^i)...(s'^m, t'^m, e^i)$ and $(s'^m, t'^m, e^i)...(s'^i, t'^i, e^i)$ s.t. $c(\pi_1) \leq v_m$ and $c(\pi_2) \leq v_n$. Combining these paths, we obtain $\pi = (s^i, t^i, e^i)...(s'^i, t'^i, e^i)$ with $c(\pi) = c(\pi_1) + c(\pi_2) \leq v_i$.

□

We have shown that, after SPA executes line 4, each entry $spa[(s, t), (s', t'), M]$ either contains the minimum cost of an e-path of depth 0 from (s, t, e) to (s', t', e) in $G_{\mathcal{U}_\mathcal{K}}$, or contains ∞ if no such path exists.

We will continue by showing that these results extend to all depths $\mathfrak{d} > 0$ w.r.t. $\mathfrak{d}$ applications of f.

Lemma 16. *Let $\mathfrak{d} \geq 1$, $(s, t), (s', t') \in Q_\mathfrak{R} \times Q_\mathfrak{T}$, $e \in \Delta^{\mathcal{U}_\mathcal{K}} \setminus Ind(A)$ and $tail(e) = M$. After $\mathfrak{d}$ applications of f, $spa[(s, t), (s', t'), M]$ contains the minimal cost c^* of an e-path of depth at most $\mathfrak{d}$ from (s, t, e) to (s', t', e).*

Proof. We show by induction on the number of iterations $\mathfrak{d}$ that $spa[(s, t), (s', t'), M] = c^*$. We obtain the base case for $\mathfrak{d} = 0$ from Lemma 1. For the induction step, we show both directions:
$\Rightarrow$: $spa[(s, t), (s', t'), M] \geq c^*$. If $spa[(s, t), (s', t'), M]$ never gets updated during the exectutions of f, the base case applies. Otherwise, suppose that $spa[(s, t), (s', t'), M]$ is updated by applications of S1 or S3.

- S1: If S1 was applied, all conditions for $C1^*$ were met for $spa[(s, t), (s', t'), M]$. Therefore, we have some $M_1 \subseteq \mathsf{N}_C$ and $R, R', R'' \in \mathsf{N}_\mathsf{R}^\pm$ s.t. $\mathcal{T} \models M \sqsubseteq \exists R.M_1$, $\mathcal{T} \models R \sqsubseteq R'$, $\mathcal{T} \models R^- \sqsubseteq R''$, $(s, u, s_1) \in \delta_\mathfrak{R}$, $(t, u, R', w_1, t_1) \in \delta_\mathfrak{T}$, $(s_2, u', s') \in \delta_\mathfrak{R}$ and $(t_2, u', R'', w_2, t') \in \delta_\mathfrak{T}$. By the definition of $G_{\mathcal{U}_\mathcal{K}}$, we know that two edges exists in $G_{\mathcal{U}_\mathcal{K}}$:

  $$(s, t, e)w_1(s_1, t_1, e') \text{ and } (s_2, t_2, e')w_2(s', t', e)$$

  with $e \neq e'$, $tail(e') = M'$, $(e, e') \in R'^{\mathcal{U}_\mathcal{K}}$ and $(e', e) \in R''^{\mathcal{U}_\mathcal{K}}$. By Induction, the minimum cost of an e'-path from (s_1, t_1, e') to (s_2, t_2, e') of depth $\mathfrak{d} - 1$ corresponds to the entry in $spa[(s_1, t_1), (s_2, t_2), M']$. Therefore, we have $c^* \leq w_1 + spa[(s_1, t_1), (s_2, t_2), M'] + w_2$. As the updated value of $spa[(s, t), (s', t'), M]$ is $w_1 + spa[(s_1, t_1), (s_2, t_2), M'] + w_2$, we have $spa[(s, t), (s', t'), M] \geq c^*$.

- S3: Applications of S3 are performed after all applications of S1 and do not increase the depth of the corresponding path in $G_{\mathcal{U}_\mathcal{K}}$. Therefore, we can follow the proof from Lemma 1 for paths of depth 0.

$\Leftarrow$: $spa[(s,t),(s',t'),M] \leq c^*$. An e-path of depth at most $\mathfrak{d}$ is of the form:

$$(s_1,t_1,e)\gamma_1(s_2,t_2,e)\gamma_2...\gamma_{n-1}(s_n,t_n,e) \tag{7.1}$$

with $(s_1 = s, t_1 = t, s_n = s', t_n = t')$, where for all γ_i with $1 \leq i < n$ either:

- $\gamma_i \in \mathbb{N}$: Then, $\pi_i = (s_i,t_i,e)\gamma_i(s_{i+1},t_{i+1},e)$ is an e-path of depth 0. We obtain $c(\pi_i) \geq spa[(s_i,t_i),(s_{i+1},t_{i+1}),M]$ from Lemma 1.
- $(s_i,t_i,e)\gamma_i(s_{i+1},t_{i+1},e)$ is an e-path of the form:

  $$\pi_i = (s_i,t_i,e)w'(s'_i,t'_i,e')\gamma(s''_i,t''_i,e')w''(s_{i+1},t_{i+1},e)$$

  where $e' \in T_e, tail(e') = M'$ and $\pi'_i = (s'_i,t'_i,e')\gamma(s''_i,t''_i,e')$ is an e'-path of depth at most $\mathfrak{d}-1$. The cost of π_i is $c(\pi_i) = w' + c(\pi'_i) + w''$.

  By Induction, we have $c(\pi'_i) \geq spa[(s'_i,t'_i),(s''_i,t''_i),M']$. Thus, we obtain $c(\pi_i) \geq w' + spa[(s'_i,t'_i),(s''_i,t''_i),M'] + w''$. It remains to show that $spa[(s_i,t_i),(s_{i+1},t_{i+1}),M] \leq w' + spa[(s'_i,t'_i),(s''_i,t''_i),M'] + w''$:

  As π_i is an e-path in $G_{\mathcal{U}_\mathcal{K}}$, the following edges must exists in $G_{\mathcal{U}_\mathcal{K}}$:

  $$(s_i,t_i,e), w', (s'_i,t'_i,e') \text{ and } (s''_i,t''_i,e')w''(s_{i+1},t_{i+1},e)$$

  By the definition of $G_{\mathcal{U}_\mathcal{K}}$, we have:

  $$(s_i,u,s'_i) \in \delta_\mathfrak{R}, (t_i,u,R',w',t'_i) \in \delta_\mathfrak{T} \text{ with } R' \in \mathsf{N}_\mathsf{R}^\pm, (e,e') \in R'^{\mathcal{U}_\mathcal{K}}$$

  and respectively

  $$(s''_i,u,s_{i+1}) \in \delta_\mathfrak{R}, (t''_i,u,R'',w',t_{i+1}) \in \delta_\mathfrak{T} \text{ with } R'' \in \mathsf{N}_\mathsf{R}^\pm, (e',e) \in R''^{\mathcal{U}_\mathcal{K}}$$

  Note that $e' \in T_e$ and $tail(e') = M'$. From the definition of $\mathcal{U}_\mathcal{K}$, we obtain $\mathcal{T} \models M \sqsubseteq \exists R.M'$. As $tail(e') = M'$ and $(e,e') \in R'^{\mathcal{U}_\mathcal{K}}$, there must exists some R with $R^{\mathcal{U}_\mathcal{K}} \subseteq R'^{\mathcal{U}_\mathcal{K}}$. According to the definition of $\mathcal{U}_\mathcal{K}$, this requires $\mathcal{T} \models R \sqsubseteq R'$. We follow the same argument to show that $\mathcal{T} \models R^- \sqsubseteq R''$.

  Hence, all conditions for $C1^*$ are satisfied. S1 can be applied to obtain

  $$spa[(s_i,t_i),(s_{i+1},t_{i+1}),M] \leq w' + spa[(s'_i,t'_i),(s''_i,t''_i),M'] + w''$$

  and thus

  $$c(\pi_i) \geq spa[(s_i,t_i),(s_{i+1},t_{i+1}),M].$$

  The cost of an e-path of depth at most $\mathfrak{d}$ with the form of (1) is $c = c(\pi_1) + c(\pi_2) + ... + c(\pi_{n-1})$. We have shown that, for all subpaths π_i with $1 \leq i < n$, it holds that $c(\pi_i) \geq spa[(s_i,t_i),(s_{i+1},t_{i+1}),M]$. Combining these results, we obtain $c^* \geq spa[(s,t),(s',t'),M]$.

It follows that $c^* = spa[(s,t),(s',t'),M]$. □

Using these rules, we can make an interesting observation on the values added to the table: For an entry $spa[(s,t),(s',t'),M]$, the value can only decrease if we add more concept names to M. This meets our intuition of the table spa - if an element satisfies additional concepts, we can possibly find additional (better) loops starting from that element. Loops infered by the other concepts present remain available.

Lemma 17. *After SPA executes line 3, for each entry $spa[(s,t),(s',t'),M]$, the following holds: If $M' \subseteq M$, then $spa[(s,t),(s',t'),M] \leq spa[(s,t),(s',t'),M']$.*

Proof. Assume the contrary: Let $M' \subseteq M$ and $spa[(s,t),(s',t'),M'] < spa[(s,t),(s',t'),M]$. Then, $spa[(s,t),(s',t'),M']$ was updated by an application of S2 with

$$\mathcal{T} \models M' \sqsubseteq A, \qquad (s,u,s') \in \delta_{\mathfrak{R}}, \qquad (t,u,A?,w,t') \in \delta_{\mathfrak{T}}.$$

Because $M' \subseteq M$, M is *more specific* than M', i.e. it holds that $M^{\mathcal{U}_\mathcal{K}} \subseteq M'^{\mathcal{U}_\mathcal{K}}$. We obtain $\mathcal{T} \models M \sqsubseteq M'$ as a direct consequence of the semantics. Thus, it holds that $\mathcal{T} \models M \sqsubseteq A$, and hence the same u, A as above can be used to update $spa[(s,t),(s',t'),M]$ with w. We have $spa[(s,t),(s',t'),M] \leq spa[(s,t),(s',t'),M']$, which is a contradiction. □

This property of spa is preserved during (exhaustive) applications of S1,S2 and S3:

Lemma 18. *After SPA finishes, the following holds: If $M' \subseteq M$, then $spa[(s,t),(s',t'),M] \leq spa[(s,t),(s',t'),M']$.*

Proof. By Lemma 17, the claim holds after SPA finishes Line 3. We only consider the final update to $spa[(s,t),(s',t'),M']$, which was either an application of S1 or S3. Now, assume the contrary: Let $M' \subseteq M$ and $spa[(s,t),(s',t'),M'] < spa[(s,t),(s',t'),M]$.

- S1: If S1 was applied to update $spa[(s,t),(s',t'),M']$, we have

$$\begin{array}{llll} M_1 \subseteq \mathsf{N}_C & \mathcal{T} \models M' \sqsubseteq \exists R.M_1, & \mathcal{T} \models R \sqsubseteq R', & \mathcal{T} \models R^- \sqsubseteq R'', \\ (s,u,s_1) \in \delta_{\mathfrak{R}}, & (t,u,R',w_1,t_1) \in \delta_{\mathfrak{T}}, & (s_2,u',s') \in \delta_{\mathfrak{R}}, & (t_2,u',R'',w_2,t') \in \delta_{\mathfrak{T}} \end{array}$$

  Because $M' \subseteq M$, we obtain $\mathcal{T} \models M \sqsubseteq M'$, and thus $\mathcal{T} \models M \sqsubseteq \exists R.M_1$ for the same M_1 as above. Using the same u, u', R', R'', we obtain: $spa[(s,t),(s',t'),M] \leq w_1 + spa[(s,t),(s',t'),M_1] + w_2 = spa[(s,t),(s',t'),M']$, which is a contradiction.

- S3: If S3 was applied to update $spa[(s,t),(s',t'),M']$, we have

$$spa[(s,t),(s',t'),M'] = spa[(s,t),(s'',t''),M'] + spa[(s'',t''),(s',t'),M'].$$

  Our claim yields:

$$spa[(s,t),(s'',t''),M'] \geq spa[(s,t),(s'',t''),M]$$
and
$$spa[(s'',t''),(s',t'),M'] \geq spa[(s,t),(s'',t''),M]$$

  Thus, $spa[(s,t),(s'',t''),M]$ and $spa[(s,t),(s'',t''),M]$ could have been used to update $spa[(s,t),(s',t'),M] \leq spa[(s,t),(s'',t''),M'] + spa[(s'',t''),(s',t'),M'] = spa[(s,t),(s',t'),M']$, which is a contradiction.

□

To continue, we show that spa can be constructed in finite time by reaching a fixpoint on the number of iterations of f. While $G_{\mathcal{U}_\mathcal{K}}$ can possibly be infinite, we notice that the properties inherited from the universal model allow us to restrict our attention to a finite fragment of $G_{\mathcal{U}_\mathcal{K}}$. Let us consider the amount of vertices $(s,t,e) \in Q_{\mathfrak{R}} \times Q_{\mathfrak{T}} \times \Delta^{\mathcal{U}_\mathcal{K}}$ in $G_{\mathcal{U}_\mathcal{K}}$ that differ in either s, t or $tail(e)$:

- $s \in Q_{\mathfrak{R}}$: $|Q_{\mathfrak{R}}|$ possible values
- $t \in Q_{\mathfrak{T}}$: $|Q_{\mathfrak{T}}|$ possible values
- $tail(e) \in \mathsf{T}(\mathcal{U}_{\mathcal{K}})$: $|\mathsf{T}(\mathcal{U}_{\mathcal{K}})| = |\mathcal{P}(Sig(T))| = 2^{|Sig(T)|} + 1$.

Thus, there are $m = (|Q_{\mathfrak{R}}| \cdot |Q_{\mathfrak{T}}|) \cdot |\mathsf{T}(\mathcal{U}_{\mathcal{K}})|$ different combinations of s, t, and $tail(e)$. We will use this property to show that we only have to consider paths of finite depth in $G_{\mathcal{U}_{\mathcal{K}}}$.

Lemma 19. *Let (s,t), $(s',t') \in Q_{\mathfrak{R}} \times Q_{\mathfrak{T}}$ and $e_0 \in \Delta^{\mathcal{U}_{\mathcal{K}}} \setminus Ind(A)$ such that there is an e_0-path $(s,t,e_0)...(s',t',e_0)$ in $G_{\mathcal{U}_{\mathcal{K}}}$. Then, there is one such path of minimal cost with depth at most $m = (|Q_{\mathfrak{R}}| \cdot |Q_{\mathfrak{T}}|)^2 \cdot |\mathsf{T}(\mathcal{U}_{\mathcal{K}})|$.*

The proof is the same as for Lemma 23 in ([FT21]).

B.2. Running time of SPA

We show that spa can be constructed in exponential time in the combined size of $\mathcal{T}, \mathfrak{R}$ and $\mathfrak{T}$ by using a fixpoint on the number of iterations of function f in procedure SPA. We start by looking at the complexity of applications of rules S1-S3, which are used in SPA.

Corollary 3. *An application of rule S1 is feasible in exponential time in the combined size of $\mathcal{T}, Q$ and $\mathfrak{T}$.*

Proof. An application of S1 can be done using the following steps:

- 1) Guess $M_1 \in \mathsf{T}(\mathcal{U}_{\mathcal{K}})$ and $R \in \mathsf{N}_{\mathsf{R}}^{\pm} \cup \{A? \mid A \in Sig(T)\}$
- 2) Check $\mathcal{T} \models M \sqsubseteq \exists R.M_1$
- 3) Guess R_1, R_2 s.t. $R \sqsubseteq R_1, R^- \sqsubseteq R_2$
- 4) Check $(s,u,s_1) \in \delta_{\mathfrak{R}}$, $(t,u,R_1,w,t_1) \in \delta_{\mathfrak{T}}$, $(s_1,u',s') \in \delta_{\mathfrak{R}}$, $(t_1,u',R_2,w,t') \in \delta_{\mathfrak{T}}$

1) and 3) are non-deterministic in the size of $\mathsf{T}(\mathcal{U}_{\mathcal{K}})$, 2) requires a subsumption check which requires EXP-time in $|\mathcal{T}|$ for $\mathcal{ELHI}_{\perp}$, and 4) is feasible in polynomial time. Thus, we obtain $NP^{\mathsf{EXP}} = \mathsf{EXP}$ in combined complexity. □

Corollary 4. *An application of rule S2 is feasible in exponential time in the combined size of $\mathcal{T}, Q$ and $\mathfrak{T}$.*

Proof. An application of S1 can be done using the following steps:

- 1) Guess $A \in Sig(T)$ and $u \in \Sigma_R$ s.t. $(s,u,s') \in \delta_{\mathfrak{R}}$ and $(t,u,A?,w,t') \in \delta_{\mathfrak{T}}$ and w is minimal
- 2) Check $\mathcal{T} \models M \sqsubseteq A$

1) is non-deterministic in the size of $\mathsf{T}(\mathcal{U}_{\mathcal{K}})$, 2) requires a subsumption check which in turn requires EXP-time in the size of $\mathcal{T}$ for $\mathcal{ELHI}_{\perp}$. Thus, we obtain $NP^{\mathsf{EXP}} = \mathsf{EXP}$ in combined complexity. □

Corollary 5. *An application of rule S3 is feasible in exponential time in the combined size of $\mathcal{T}, Q$ and $\mathfrak{T}$.*

Proof. An application of S3 can be done using the following steps:

- 1) For each $(s'', t'') \in Q \times \mathfrak{T}$: Look up $spa[(s,t),(s'',t''),M]$ and $spa[(s'',t''),(s',t'),M]$

The amount of (s'', t'') is $|Q| \cdot |\mathfrak{T}|$, thus the amount is polynomially bounded in combined complexity. □

The amount of entries in spa corresponds to $(|Q_{\mathfrak{R}}| \cdot |Q_{\mathfrak{T}}|)^2 \cdot |\mathsf{T}(\mathcal{U}_{\mathcal{K}})|$. As the size of $Q_{\mathfrak{R}}$ and $Q_{\mathfrak{T}}$ is polynomial and $\mathsf{T}(\mathcal{U}_{\mathcal{K}})$ is exponential in the size of $\mathcal{T}$, spa contains EXP-many entries in the combined size of $\mathcal{T}, \mathfrak{R}$ and $\mathfrak{T}$.

Next, we consider the amount of rule applications of S1, S2 and S3 at each line of spa:

- Line 3: Apply S2 to all $(p, q, M) \in (Q_{\mathfrak{R}} \times Q_{\mathfrak{T}})^2 \times \mathsf{T}(\mathcal{U}_{\mathcal{K}}))$. This requires EXP-many applications of S2 in the size if $Sig(\mathcal{T})$.
- Line 4: Apply S3 once for all $M \in \mathsf{T}(\mathcal{U}_{\mathcal{K}})$. This requires EXP-many applications of S3 in the size if $Sig(\mathcal{T})$.
- function f: Applying S1 to all $(p, q, M) \in (Q_{\mathfrak{R}} \times Q_{\mathfrak{T}})^2 \times \mathsf{T}(\mathcal{U}_{\mathcal{K}}))$ requires EXP-many applications of S1 in the size if $Sig(\mathcal{T})$. Applying rule S3 to all $M \in \mathsf{T}(\mathcal{U}_{\mathcal{K}})$ requires EXP-many applications of S3 in the size of $Sig(\mathcal{T})$

Combining these results, we obtain that an execution of SPA requires at most EXP-time in the combined size of $\mathcal{T}, \mathfrak{R}$ and $\mathfrak{T}$. By lemma 18, SPA outputs the relation spa as required. Then, we obtain the following result:

Corollary 6. *The relation spa can be computed in at most EXP-time in the combined size of $\mathcal{K}, \mathfrak{R}$ and $\mathfrak{T}$.*

Proof.

$$O(SPA) = \mathsf{EXP} \cdot \mathsf{EXP} + \mathsf{EXP} \cdot L + m \cdot (\mathsf{EXP} \cdot \mathsf{EXP} + \mathsf{EXP} \cdot L) = \mathsf{EXP}$$

where m is the maximum amount of applications of f. By Lemma 19, m is bounded by a number at most EXP-sized in the combined size of $\mathcal{K}, \mathfrak{R}$ and $\mathfrak{T}$. □

C. Approximate semantics for $\mathcal{ELHI}_\bot$

In [FT21], it is shown that, for a C2RPQ q and an $\mathcal{ELH}$ KB $\mathcal{K}$, the set of approximate certain answers $\tilde{\mathsf{cert}}_{\mathfrak{T},f}(q, \mathcal{K})$ can be characterized by only considering approximate matches in the universal model $\mathcal{U}_{\mathcal{K}}$. We will show that the same holds for an C2RPQ q and $\mathcal{K}$ an $\mathcal{ELHI}_\bot$ KB.

We start by introducing the notion of *approximate matches* for a C2RPQ q in an interpretation I. Given a dT $\mathfrak{T}$ and a p-ary combining function f, an *approximate match* for q in I, through $\mathfrak{T}$ and f is a pair $h^{q,\mathcal{I}}_{\mathfrak{T},f} = (h, h_c)$, where:

- h is a mapping $\mathsf{terms}(q) \to \Delta^{\mathcal{I}}$ s.t. $h(b) = b^I$ for all $b \in \mathsf{terms}(q) \cap \mathsf{N_I}$
- $h_c \in \mathbb{N} \cup \infty$ is the approximation cost for h with:

$$h_c := \bigsqcap_{\mathfrak{R}(t,t') \in q} \min\{c_{\mathfrak{T}}(u, v) \mid u \in \mathcal{L}(\mathfrak{R}), v \in \Sigma^*, h(t) \xrightarrow{I,v} h(t')\}$$

Given a k-tuple $\bar{d}$ of elements from $\Delta^{\mathcal{I}}$ and $\bar{x} = \mathsf{avars}(q)$, $H^{q,\mathcal{I}}_{\mathfrak{T},f}(\bar{d})$ denotes the *set of approximate matches* (h, h_c) satisfying $h(\bar{x}) = \bar{d}$. Now, recall the definition of *approximate answers* over graph databases from [FT21]:

Definition 6. Let $q(\bar{x})$ ne a C2RPQ with p atoms, $\mathfrak{T}$ a distortion transducer and f a p-ary combining function. The set of *approximate answers* of q in an interpretation $\mathcal{I}$, through $\mathfrak{T}$ and f, is defined as:

$$\tilde{\mathsf{ans}}_{\mathfrak{T},f}(q,\mathcal{I}) := \left\{(\bar{d}, \eta_{\bar{d}}) \mid \bar{d} \in \Delta^{\mathcal{I}} \text{ and } \eta_{\bar{d}} = \min\{h_c \mid (h, h_c) \in H^{q,\mathcal{I}}_{\mathfrak{T},f}(\bar{d})\}\right\}$$

We continue by showing that the definition of $\tilde{\mathsf{cert}}_{\mathfrak{T},f}(q,\mathcal{K})$ from [FT21] can be extended to the $\mathcal{ELHI}_\bot$ setting.

Definition 7. Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be an $\mathcal{ELHI}_\bot$ KB and $q(\bar{x})$ a C2RPQ with p atoms. The set of *certain approximate answers* of q w.r.t. $\mathcal{K}$, through a dT $\mathfrak{T}$ and a p-ary combining function f, is defined as:

$$\tilde{\mathsf{cert}}_{\mathfrak{T},f}(q,\mathcal{K}) := \left\{(\bar{a}, \eta_{\bar{a}}) \mid \bar{a} \in \mathsf{Ind}(\mathcal{A}) \text{ and } \eta_{\bar{d}} = \sup_{\mathcal{I} \models \mathcal{K}}\{\eta_{\bar{d}} \mid (\bar{d}, \eta_{\bar{d}}) \in \tilde{\mathsf{ans}}_{\mathfrak{T},f}(q,\mathcal{I}) \wedge \bar{d} = \bar{a}^{\mathcal{I}}\}\right\}$$

To show that $\tilde{\mathsf{cert}}_{\mathfrak{T},f}(q,\mathcal{K})$ is still well-defined for an $\mathcal{ELHI}_\bot$ KB, i.e. the supremum in Definition 7 always exists, we draw from the construction of the universal model in **??** the following observation:

Proposition 4. *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be an $\mathcal{ELHI}_\bot$ KB and $\mathcal{U}_\mathcal{K}$ the universal model of $\mathcal{K}$. For each model $\mathcal{I}$ of $\mathcal{K}$ there is a homomorphism $hom : \Delta^{\mathcal{U}_\mathcal{K}} \to \Delta^{\mathcal{I}}$ such that:*

- $hom(a^{\mathcal{U}_\mathcal{K}}) = a^{\mathcal{I}}$ *for each* $a \in \mathcal{A}$
- $e \in A^{\mathcal{U}_\mathcal{K}}$ *implies* $hom(e) \in A^{\mathcal{I}}$ *for every* $A \in N_C$
- $(e, e') \in r^{\mathcal{U}_\mathcal{K}}$ *implies* $(hom(e), hom(e')) \in r^{\mathcal{I}}$ *for every* $r \in N_R^{\pm} \cup \{A? \mid A \in N_C\}$

A corresponding proof can be found in [BO15]. Now, we argue that the following holds:

Lemma 20. *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be an $\mathcal{ELHI}_\bot$ KB, $q(\bar{x})$ a k-ary C2RPQ with p atoms, $\mathfrak{T}$ a distortion transducer and f a p-ary combining function. Further, let $(\bar{a}, \eta^*) \in \tilde{\mathsf{ans}}_{\mathfrak{T},f}(q, \mathcal{U}_\mathcal{K})$ where $\bar{a}$ is a k-ary tuple of individual names in $\mathcal{A}$.*

Then, for any model $\mathcal{I}$ of $\mathcal{K}$ we have that $(\bar{d}, \eta_{\bar{d}}) \in \tilde{\mathsf{ans}}_{\mathfrak{T},f}(q, I)$ implies that $\eta_{\bar{d}} \leq \eta^*$, where $\bar{d} = \bar{a}^{\mathcal{I}}$.

A corresponding proof for $\mathcal{ELH}$ KBs can be found in [FT21]. One can see that the notions for approximate answers and certain approximate answers for $\mathcal{ELH}$ used there match the ones given in definitions 1 and 2 for $\mathcal{ELHI}_\bot$. In fact, the only difference to the $\mathcal{ELH}$ setting that is relevant here lies in the domain of $\mathcal{U}_\mathcal{K}$. To this end, Proposition 4 is used to ensure the existence of a homomorphism hom corresponding to the one required in the argument for $\mathcal{ELH}$.

Finally, we obtain that the set $\tilde{\mathsf{cert}}_{\mathfrak{T},f}(q,\mathcal{K})$ can be characterized by only considering matches in $\mathcal{U}_\mathcal{K}$ as a direct consequence of definition 7. This is formalized in the following corollary:

Corollary 7. *Let $K = (\mathcal{T}, \mathcal{A})$ be an $\mathcal{ELHI}_\bot$ KB, $q(\bar{x})$ a C2RPQ with p atoms, $\mathfrak{T}$ a distortion transducer and f a p-ary combining function. Then, $(\bar{a}, \eta_{\bar{a}}) \in \tilde{\mathsf{cert}}_{\mathfrak{T},f}(q, K)$ iff $(\bar{a}, \eta_{\bar{a}})$ is an approximate answer of q in $\mathcal{U}_\mathcal{K}$.*