Université de Mons
Faculté des Sciences
Département d'Informatique

Dealing with Inconsistencies in Knowledge Bases

Horacio Tellez Perez

A dissertation submitted in fulfilment of the requirements of the degree of *Docteur en Sciences*

Advisor

Prof. Dr. Jef Wijsen	Université de Mons

Jury

Dr. Meghyn Bienvenu	CNRS, Université de Bordeaux, France
Prof. Dr. Véronique Bruyère	Université de Mons
Dr. Alexandre Decan	Université de Mons
Prof. Dr. Floris Geerts	Universiteit Antwerpen, Belgium
Prof. Dr. Olivier Kaufmann	Université de Mons
Prof. Dr. Jef Wijsen	Université de Mons

September 2021

Thanks

This period of my life is now ending and there are several people to whom I am grateful.

First, I would like to thank Jef, my supervisor and truthfully a guide during these four years. Thank you for your advice, your time, and your patience. Thank you for your unyielding help, we both know I needed it. Some people are a crucial influence on a person's life, I know you are one for me. Thank you for giving me the chance to experience all of this, it was amazing. I was really lucky to have you as my advisor.

Thanks to Meghyn Bienvenu, Véronique Bruyère, Alexandre Decan, Floris Geerts and Olivier Kauffman who took part of their time and accepted to be members of my jury. Thank you for your helpful remarks and advice.

The learning and working environment at the UMONS is wonderful and I am thankful to have had the possibility of experiencing it. Thanks to the Secretariat department, always glad to help me, from the first day nine years ago, in particular Claudia. Thanks to the professors, who took their time to answer my questions, even when I was no longer a student; a particular thanks to Christophe Troestler, whom I believed nine years ago was an infinite source of knowledge and now I am convinced. Thanks also to Thomas Brihaye, each five minutes talk with you made my day a little brighter.

I would like to thank my friends and colleagues, research is more fun when you have people you can complain to. Thank you Maximilien, the best bureau partner would have been a clone of me, but you are the next best thing surely; I almost made a tiny house because of you. Thank you Marion for introducing me to Kaamelott and board games, now I am stuck with you as a friend for life; knowing you is in my top 10^{100} experiences in life, guessing the exact number is up to you. Thank you Aline, we have walked almost the same academic path, talking to you have this pleasant feeling of talking to an old friend of fifty years, which is weird because we have known each other for only nine. Thank you Clément and Pierre for letting me interrupt your work practically every day, with the time I passed in your bureau I almost deserve a chair there; Pierre I hope you know I always let you win in chess; Clément I'm waiting the one million euros idea. Thank you Jeremy and David, tweaking a Linux distribution is a child's play now. Thank you Adrian, watching Roland Garros once was all it took to love tennis.

Thank you Danny and Samuel, with our talks I'm now sure that loving science is for life.

Thanks to my family, that is bigger than they imagine, as my life partner's family is also mine. I am blessed to have so many good-natured people around me. Thanks to my step-mother, who cares for me more than I could have ever imagined. Thanks to my father-in-law, for all of his advice and his immeasurable help, a great part of my present happiness is thanks to him. Thanks to my fathers. Thanks to my mother, you are figuratively and literally the reason I am here; you are more stressed than I am with this work, but everything should be fine.

Last, I would like to thank my partner in life, thank you Justine. Merci, il y a plus de lumière dans ma vie que je n'aurais jamais crue possible grâce à toi. Pour la stabilité actuelle dans ma vie, le mérite te revient à toi seule. Merci d'avoir su garder mes pieds sur terre, tu me fais tellement de bien et tu ne le sais même pas. Merci pour ta patience et ton support, cette thèse est ton travail autant que le mien (peut-être plus le mien quand même). Je t'aime.

Abstract

This thesis develops and studies theoretical frameworks for dealing with inconsistencies in database and knowledge base systems. A first framework defines a mapping language for expressing rules that take a relational database instance as input, and produce an ABox in some description logic (DL). Given a family of mapping rules, it is desirable that every database instance that is consistent with respect to some given integrity constraints maps to an ABox that is consistent with respect to a given TBox. While it is generally undecidable whether this and other desirable properties obtain, it is shown that decidability can be achieved under some moderate syntactic restrictions.

A second framework addresses the problem of repairing ABoxes that are inconsistent with respect to a given TBox. It introduces a novel approach for computing a numeric credibility score for each ABox assertion, by combining a user-defined initial scoring with logical arguments and counterarguments derived from the TBox. Once a credibility score has been established for each ABox assertion (or, in general, for each fact of a knowledge base), it is natural to define repairs as consistent subsets of the ABox with maximum aggregate credibility score, according to some aggregation function. It is studied how the computational complexity of recognizing such repairs depends on certain characteristics of the aggregation function.

In addition to these theoretical developments, a software system has been built that implements the computational approach underlying the second framework.

Contents

CHAPTER 1

Introduction

> “To measure is to know.” “If you cannot measure it, you cannot improve it.” “When you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind.”
>
> Lord Kelvin

In this introductory chapter, we first outline the organization and contributions of this thesis. We then provide some background on topics in database theory and description logics that are relevant for this thesis. A background on relevant topics in algebra is given in Appendix C.

1.1. Context and Contributions of the Thesis

Context In recent years, data production is growing at an almost exponential rate [2, 3]. Moreover, there is an increased need to store and exchange data. This data proliferation often happens in an uncontrolled or little controlled fashion, which leads to *data quality problems*, meaning that data can become incomplete, uncertain, contradictory, inconsistent... The challenges related to data inconsistency have been particularly addressed in two research communities: the elder community studying theoretical foundations for *(relational) database systems* [6], and the younger one studying the *Semantic Web* [102]. In database systems, integrity constraints are used to capture more of the meaning of the data, thereby defining the "consistent" data. There are two main approaches to deal with inconsistent database instances: in *data cleaning* [63], the aim is to arrive at a single consistent database; the basic assumption in *database repairing* [118] is that there may be no single best way to clean an inconsistent database, in which case one has to deal with multiple *repairs*. Reasoning about integrity constraints has been at the center of database research since the introduction of the relational model. On the other hand, the Semantic Web is supported by Description Logics [16], a family of logics tailored for knowledge representation. Description Logics (DL) are not only designed for representing information, but also for automated reasoning about this information, seeking a good balance between expressive power and complexity of reasoning. Database theory and Description Logics are the main frameworks used in this thesis to study the *inconsistency problem*. Significantly, in recent years, there have been growing research efforts to make these two frameworks work together. Notably, the paradigm of *Ontology Based Data Access* (OBDA) seeks to benefit from both worlds, by querying relational databases through an ontological language, and by enriching query answers by means of ontological reasoning. Such reasoning can also unveil conflicts and errors in the data. The first part of this thesis will develop and investigate an OBDA framework that takes into account the inconsistency problem. The second part will develop quantified approaches for dealing with inconsistency in knowledge bases. We will now describe these contributions in more detail.

Study of an ODBA framework Chapter 2 defines an OBDA mapping language for expressing rules that take a relational database instance as input, and produce an ABox in some description logic DL. It is assumed that the fixed database schema is equipped with a set Σ of integrity constraints, while the target ABox is subject to a fixed TBox $\mathcal{T}$ in some description logic. Our mapping language is designed to be user-friendly by providing an intuitive syntax that is nevertheless expressive. Given a family $\mathcal{M}$ of mapping rules, it is natural to ask questions about the relationship between the (in)consistency, with respect to Σ, of the input database instances and the (in)consistency, with respect to $\mathcal{T}$, of the ABoxes produced by the mapping. Such questions include the following.

- Is there at least one consistent (with respect to Σ), non-empty database that maps to an ABox that is consistent with respect to $\mathcal{T}$? If the answer to this question is "no," then at least one component among Σ, $\mathcal{T}$, or $\mathcal{M}$ must be incorrectly specified.

- Is there a consistent (with respect to Σ) database that maps to an ABox that is inconsistent with respect to $\mathcal{T}$? If the answer to this question is "no," then database consistency implies consistency of the produced ABox.

- Conversely, is there an inconsistent (with respect to Σ) database that maps to an ABox that is consistent with respect to $\mathcal{T}$?

While these problems are generally undecidable, it is shown that decidability can be achieved under some moderate syntactic restrictions. We study how the computational complexity of these problems depends on the expressive power of the languages used for Σ and $\mathcal{T}$.

Study of a framework for ranking ABox assertions Following the OBDA study of Chapter 2, we will develop a quantified approach to the inconsistency problem in knowledge bases represented by a TBox and an ABox. In this part of the thesis, we assume that all information is already in an ontology $\langle \mathcal{T}, \mathcal{A} \rangle$, ignoring the database component. In practice, such an ABox could be explicitly given or be the result of applying an OBDA mapping on a database instance, as defined in Chapter 2. Chapter 3 introduces a framework that addresses the problem of assessing the quality of assertions in ABoxes that are inconsistent with respect to a given TBox. It introduces a novel approach for computing a numeric credibility score for each ABox assertion, by combining a user-defined initial scoring with logical arguments and counter-arguments derived from the TBox. Informally, starting from the user-defined base score, the quality of an assertion should be increased if it is supported by other high-quality assertions, and should be decreased if it is refuted by other high-quality assertions. This supporting and refuting evidence for assertions is modeled by a system of linear equations. We study the conditions under which this system has a solution. Moreover, we discuss how to pick a "stabilized" ranking if multiple solutions exist. Significantly, we have developed a software tool rustoner that implements the proposed framework. Rustoner is discussed in Chapter 5.

Study of weighted repairs Most existing approaches to database repairing define a repair as a consistent database that is maximally close, according to some fixed distance measure, to the original, inconsistent database. Common distance measures state that the symmetric difference between repairs and the original database should be minimal with respect to set inclusion or cardinality. Although these distance measures are theoretically elegant, they are often unsatisfactory in practice because they are largely agnostic about the meaning of the data. We believe that it is worthwhile to develop capabilities for further restricting the set of repairs—in the same way as integrity constraints restrict the set of possible databases.

Once a credibility score has been established (in Chapter 3) for each ABox assertion (or, in general, for each fact of a knowledge base), it is natural to define repairs as consistent subsets of the ABox with maximum aggregate credibility score, according to some aggregation function. Such an approach is investigated in Chapter 4. We study how the computational complexity of recognizing aggregate-based repairs depends on certain characteristics of the used aggregation function, and present some desirable properties of the aggregation function that lead to polynomial-time complexity.

This thesis is deliberately written in such a way that each chapter is self-contained and can be read on its own. To achieve this, some small overlap among chapters was unavoidable. We believe that the chapters together form a coherent whole with three successive blocks, as described next. First, an OBDA mapping language is defined to map databases to ontologies. Such mapping defines an ABox which can be materialized or remain virtual. Second, an ABox ranking allows for the quantification of "quality" or "trustfulness" of ABox assertions. Third, once data is quantified, it can be used in our study of weighted repairs.

1.2. Background from Database Theory

This part follows the theory and notations defined in [6]. In this thesis, we use the *relational database model*. Informally, a database in this model is a set of tables.

Example 1.1
We provide a simple database for storing users and books in a library.

USERS	*Id*	*Last*	*First*	*Since*
	0012	Smith	Rob	12/12/2015
	1004	Jones	Tom	02/08/2013

BOOKS	*Id*	*Title*	*Category*
	BE10	Coming Back	Romance
	BE10	The Foundation	SF

$\triangleleft$

We assume three disjoint, countably infinite sets: a set **att** of *attributes*, a set **relname** of *relation names*, and a set **dom** of *constants*. We assume a total order $\leq_{\mathbf{att}}$ on **att**. We also assume a total function *sort* with domain **relname** that maps every relation name to a finite set of attributes. In Example 1.1, *USERS* is a relation name with $sort(USERS) = \{Id,\ Last,\ First,\ Since\}$, a set of attributes.

Let U be a finite set of attributes. A *tuple over* U is a total mapping from U to **dom**. For example, the following set is a tuple over $\{Id, Last, First, Since\}$:

$$\{Id : 0012, Last : \text{Smith}, First : \text{Rob}, Since : 12/12/2015\}.$$

If the attributes in a tuple are ordered according to $\leq_{\mathbf{att}}$, then attributes can be omitted without ambiguity, as follows:

$$(0012, \text{Smith}, \text{Rob}, 12/12/2015)\,.$$

The latter representation is often referred to as the *unnamed perspective.* A *relation over* U is a finite set of tuples over U.

A *database schema* $\mathbf{S}$ is a finite set of relation names. A *database instance over* $\mathbf{S}$ (or simply, *database over* $\mathbf{S}$) is a total mapping with domain $\mathbf{S}$ that associates, to each relation name R in $\mathbf{S}$, a relation over $\mathit{sort}(R)$. If $\mathbf{db}$ is a database instance, then $R^{\mathbf{db}}$ denotes the relation associated to R. In the unnamed perspective, if $(c_1, \dots, c_\ell)$ is a tuple in $R^{\mathbf{db}}$, then we also say that $R(c_1, \dots, c_\ell)$ is a *fact* of $\mathbf{db}$. It is often convenient to represent a database instance as the set of its facts. Example 1.1 shows a database over the schema $\{\mathit{USERS}, \mathit{BOOKS}\}$.

Integrity Constraints, Inconsistency, and Repairs

A database schema $\mathbf{S}$ is commonly extended with a set Σ of integrity constraints that restrict the set of allowed database instances. In this thesis, we assume that all integrity constraints are domain-independent sentences [6, Definition 5.3.7] in predicate logic. Note that this excludes first-order sentences that are not domain-independent, for example, $\forall x R(x)$.

A database $\mathbf{db}$ is *consistent* with respect to Σ, denoted $\mathbf{db} \models \Sigma$, if it satisfies all integrity constraints in Σ; otherwise $\mathbf{db}$ is *inconsistent.* Note that satisfaction in database theory has two particularities compared to standard predicate logic: firstly, constant symbols are interpreted as themselves, and secondly, since integrity constraints are domain-independent, the truth of a sentence is the same for every universe of discourse that contains all constants occurring in $\mathbf{db}$ or Σ.

Example 1.2
The following integrity constraint expresses that no two distinct tuples in a *BOOKS* relation can agree on the attribute *Id*.

$$\forall x \forall y_1 \forall y_2 \forall z_1 \forall z_2 \left(\begin{pmatrix} \mathit{BOOKS}(x, y_1, z_1) \\ \wedge\, \mathit{BOOKS}(x, y_2, z_2) \end{pmatrix} \rightarrow (y_1 = y_2 \wedge z_1 = z_2) \right).$$

Note that the database of Example 1.1 is inconsistent with respect to this integrity constraint.

◁

We allow database instances **db** that are inconsistent with respect to a set of integrity constraints. Informally, a *repair* [12] of such a database instance **db** is a consistent database instance that can be obtained from **db** by means of some minimal change. The concept of "minimal change" can be formalized in many different ways. For example, we may restore consistency by deleting a minimal (with respect to set inclusion) set of tuples, without inserting new tuples or modifying existing tuples. This gives rise to *subset repairs*, which are *inclusion-maximal consistent subsets of* **db**. A more in-depth overview of database repairing can be found in [118].

Example 1.3
For our running example, the following are the two subset repairs of the *BOOKS* relation:

$\mathbf{r}_1 =$

BOOKS	*Id*	*Title*	*Category*
	BE10	Coming Back	Romance

$\mathbf{r}_2 =$

BOOKS	*Id*	*Title*	*Category*
	BE10	The Foundation	SF

◁

1.3. Background from Description Logics

This subsection follows the treatment in [16]. Description Logics are a family of logics ranging from fairly simple logics (e.g., *DL-Lite* [31]) to quite expressive ones (e.g., $\mathcal{SHIQ}$ [60]). We will discus notions that are common to most DLs.

As for any formal logic, expressions in DL must be syntactically well-defined. We begin by specifying three countably infinite pairwise disjoint sets: the set of *concept names* $\mathbf{C}$, the set of *role names* $\mathbf{R}$, and the set of *individuals* $\mathbf{I}$. Concept names and role names are to be interpreted by, respectively, unary and binary relations. Individuals are interpreted by constants.

We now discuss how constructs are inductively built. For the base case, we have that *atomic concepts* are concept names in $\mathbf{C}$, and *atomic roles* are role names in $\mathbf{R}$. In what follows, let C, D be valid atomic or complex concepts; and let r, s be valid atomic or complex roles. Then the following are all valid constructs:

- $\perp$ which corresponds to *nothing* or the empty set;
- $\top$ which corresponds to *all* or the set that equals the universe of interpretation;
- $C \sqcap D$ concept conjunction;
- $C \sqcup D$ concept disjunction;
- $\neg C$ concept negation;
- $\neg r$ role negation;
- r^- inverse of a role;
- $r \cap s$ role conjunction;
- r^* transitive closure of a role;
- $\forall r.C$ universal restriction; and
- $\exists r.C$ existential restriction.

While the syntax differs from first-order logic, the intended meaning is closely related to first-order semantics. A particular DL is obtained by allowing only a subset of constructors. For example, *DL-Lite* is restricted to to the following constructs:

- $\perp$;
- $\top$;
- A a basic concept;
- r a basic role;

- r^-;
- $\neg s$ where s can be atomic or inverted;
- $\exists s.\top$ where s can be atomic or inverted; and
- $\neg C$ where C can be atomic or of the form $\exists s.\top$.

Moreover, some DL add extra constructs not shown before. For example, $\mathcal{SHIQ}_{\sqcap,\sqcup,\neg(\text{full}),*}$ allows for all of the precedent constructs and some more.

TBox and ABox

Knowledge represented by means of a Description Logic is called an *ontology*. Such knowledge in an ontology is stored in two sets: the terminology box called *TBox*, and the assertion box called *ABox*. The TBox specifies general knowledge about the domain of study, including the interaction among concepts, while the ABox stores more concrete information pertaining to individuals.

Example 1.4
We present a simple ontology for the taxonomy of *canidae*. The TBox is denoted by $\mathcal{T}$ and the ABox by $\mathcal{A}$.

$$\mathcal{T} = \left\{ \begin{array}{l} \mathsf{Species} \sqsubseteq \mathsf{Genus}, \\ \mathsf{Order} \sqsubseteq \mathsf{Phylum} \sqcap \mathsf{Class} \end{array} \right\}$$

$$\mathcal{A} = \left\{ \begin{array}{l} \mathsf{chordata} : \mathsf{Phylum}, \\ \mathsf{carnivora} : \mathsf{Order}, \\ \mathsf{canis\ lupus} : \mathsf{Species}, \\ (\mathsf{canis\ lupus}, \mathsf{canis\ latrans}) : \mathit{sameGenus} \end{array} \right\}$$

$\triangleleft$

The new symbol $\sqsubseteq$ appearing in the TBox models that a concept is a specialization of another. This allows us to structure our knowledge, saying that the concept $\mathsf{Species}$ is a specialization of the concept Genus (or, conversely, that Genus is a generalization of $\mathsf{Species}$).

The ABox $\mathcal{A}$ contains specific assertions. The assertion $\mathsf{carnivora} : \mathsf{Order}$ expresses that $\mathsf{carnivora}$ is an Order, while the assertion $(\mathsf{canis\ lupus}, \mathsf{canis\ latrans}) :$

sameGenus expresses that both $\mathsf{canis\ lupus}$ and $\mathsf{canis\ latrans}$ are of the same Genus. More formally, assertions in an ABox can be of two forms:

- $\mathsf{a} : \mathsf{C}$
- $(\mathsf{a}, \mathsf{b}) : s$

where a, b are individual names, C is a concept, and s is a role. ABoxes are finite sets of ABoxes assertions.

In the field of Description Logics, an ontology $\langle \mathcal{T}, \mathcal{A} \rangle$ is also called a *knowledge base*, and often denoted by the symbol $\mathcal{K}$.

Remark 1.4
Some logics that allow *role inclusions*, like *sameGenus* $\sqsubseteq$ *sameFamily*, do not put such axioms in the TBox, but rather in an *RBox* $\mathcal{R}$.

$\triangleleft$

Syntactic Restrictions Syntactic restrictions not only apply to the construction of complex concepts and roles, but also to what expressions are allowed in the TBox and the ABox. For example, in the logic $\mathcal{ALC}$, the expression $C \sqsubseteq D$ is syntactically valid whenever C and D are atomic or complex concepts. On the other hand, in *DL-Lite*, the left-hand expression C must be a non-negated concept. Likewise, in a concept expression $a : E$, the expression E must be a basic concept in *DL-Lite*, but the logic $\mathcal{ALC}$ allows for more flexibility, sometimes allowing for E to be a complex construct.

Interpretation

Like in first-order logic, the semantics of an ontology is defined by interpreting its symbols. An *interpretation* is a pair $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ where $\Delta^{\mathcal{I}}$ is a non-empty interpretation domain, and $\cdot^{\mathcal{I}}$ is a function that maps symbols in $\langle \mathcal{T}, \mathcal{A} \rangle$ to $\Delta^{\mathcal{I}}$, as follows:

- $\bot^{\mathcal{I}}$ equals $\emptyset$;
- $\top^{\mathcal{I}}$ equals $\Delta^{\mathcal{I}}$;
- for every concept name A, $A^{\mathcal{I}}$ is a subset of $\Delta^{\mathcal{I}}$;

- for every role name r, $r^{\mathcal{I}}$ is a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$; and
- for every individual name a, $a^{\mathcal{I}}$ is an element of $\Delta^{\mathcal{I}}$.

The function $\cdot^{\mathcal{I}}$ naturally extends to complex constructs, for example:

- $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$;
- $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$;
- $(\exists r.C)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \text{there is } b \in \Delta^{\mathcal{I}} \text{ such that } (a,b) \in r^{\mathcal{I}} \text{ and } b \in C^{\mathcal{I}}\}$.

An interpretation can satisfy or violate TBox axioms and ABox assertions, in particular:

- an interpretation $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ satisfies the TBox axiom $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$;
- an interpretation $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ satisfies an ABox assertion $a : C$ if $a^{\mathcal{I}}$ is contained in $C^{\mathcal{I}}$.

Finally, we say that an interpretation $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is a *model* of an ontology $\langle \mathcal{T}, \mathcal{A} \rangle$ if $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is a valid interpretation that satisfies all axioms in $\mathcal{T}$ and all assertions in $\mathcal{A}$.

Reasoning

Standard reasoning tasks in Description Logics concern satisfiability and logical implication, which involve questions of the following kind:

- Given an ontology $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, is $\mathcal{K}$ satisfiable, i.e., is there a model $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of $\mathcal{K}$?
- Given an ontology $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ and two concepts C, D, does $\mathcal{K}$ logically imply $C \sqsubseteq D$, i.e., is it true that every model $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of $\mathcal{K}$ satisfies $C \sqsubseteq D$?
- Given an ontology $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ and an assertion α, does $\mathcal{K}$ *entail* α, that is, is every model of $\mathcal{K}$ also a model of α? A variant of this problem will play a key role in Chapter 3.

The computational complexity of these tasks depends on which constructs are (dis)allowed in the Description Logic under consideration, as nicely exposed in [16].

CHAPTER 2

Connecting Databases to Ontologies

Remark 2.0
The content of this chapter has been presented at DL 2019 [89].

◁

2.1. Motivation

The literature contains many proposals for mapping relational databases to ontologies. A major motivation for these proposals is *ontology-based data access* (OBDA) [119], i.e., the capability of interrogating databases by using an ontological vocabulary. The current study, however, started with a different purpose, which can be coined as *ontology-based database repairing* or *ontology-based database cleaning*. Database repairing [118] and data cleaning [63] are approaches for dealing with dirty data, where dirtiness refers to the violation of integrity constraints or, more abstractly, the non-conformity to rules that the data should obey. Ideally, all such data rules should be declared at database design time and subsequently enforced by the database management system. In practice, however, we seldom dispose of an exhaustive declaration of all data rules: some rules were overlooked when the database schema was conceived, while others were hidden in procedural programming code. Moreover, in the course of time, new rules may emerge because of new legislation (e.g., GDPR), while existing rules may be invalidated. Now let us assume that we have access to an ontology that talks about objects and relations that also exist in some presumably dirty database. Our hypothesis is that data quality problems may become more visible when we succeed in connecting or mapping the database to the ontology, enabling us to confront the stored data with the ontological "ground truth." It should be mentioned here that an ontologically based approach to data quality is not a new idea: it already appeared in [116], was formalized in [40], and is mentioned in [119] as an important direction for future research.

An OBDA setting consists of several components. It comprises a relational database schema (i.e., a set of relation names), a description logic vocabulary (i.e., a set of unary and binary predicate names, called concept names and role names), and a TBox in some description logic. A final component is a database-to-ontology mapping. Such a mapping takes, as input, a database instance over the fixed database schema, and returns, as output, an ABox in the description logic. If $\mathcal{M}$ denotes such a mapping and $\mathbf{db}$ denotes a database instance that serves as input to $\mathcal{M}$, then we write $\mathcal{M}(\mathbf{db})$ for the resulting ABox. In a data cleaning context, we may be interested to know, for example, whether the knowledge base $\langle \mathcal{T}, \mathcal{M}(\mathbf{db}) \rangle$ is consistent, and if not, what data in $\mathbf{db}$ causes inconsistency.

In this chapter, we introduce and study a language for specifying such mappings $\mathcal{M}$, seeking a good balance between expressiveness and complexity. Four design considerations are as follows.

- First, we work in a perspective where columns in relations are not only numbered, as in mathematical logic, but also named with *attributes*. We will not assume that real-world entities have unique identifiers. Instead, we will use tuples with attributes to identify entities. This allows us, for example, to distinguish between the actress $\{$*Lastname* : `Hilton`, *Firstname* : `Paris`$\}$ and the entity $\{$*Hotel* : `Hilton`, *City* : `Paris`$\}$, which is a hotel in Paris.

- Second, the language for mapping databases to ontologies will be a subset of relational algebra. This leads to a succinct syntax without first-order variables. A major convenience for end-users is that any syntactically correct combination of the algebra operators is allowed in our mapping language. This would not be achievable in predicate logic, where end-users would be troubled with syntactic restrictions like safeness and guardedness. The omission of variables is similar to description logics capturing fragments of first-order logic without using first-order variables, in a syntax that is friendly to end-users.

- Third, like with description logics, a major concern in the design of our mapping language is to find a good balance between expressiveness and complexity. For expressiveness considerations, we allow negation in our mapping language, which is often considered useful [20]. On the other hand, we cannot allow the full expressive power of predicate logic, because this would lead to the undecidability of some basic reasoning problems.
- Fourth, relational database schemas are often obtained from a conceptual schema expressed in the Entity-Relationship model [35] or some variant of it. In such database schemas, most database tables correspond to either an entity type or a relationship type in the conceptual schema. Intuitively, concept names and role names in description logics also correspond, respectively, to entity types and relationship types.[1] These resemblances have motivated some design choices of our mapping language. In particular, for mappings that generate concept assertions in the ABox, we have opted for including negation in our mapping language at the price of giving up on arbitrary joins. The idea is that in a well-designed database, the same real-world entity will generally not be spread out over multiple database tables, thus reducing the need for arbitrary joins. On the other hand, negation may be co“mmonly needed (for example, to compute foreign students as all students except Belgian citizens).

The main results in this chapter can be summarized as follows.

- In terms of expressiveness, our language is incomparable with the commonly used language of GLAV mappings [32]. In particular, we allow

[1] A mathematical logician may object that concept names and role names are just strings in some vocabulary.

negation but disallow arbitrary conjunctive queries at the left-hand sides of mapping rules. We thus obtain an expressive language, while important reasoning problems remain decidable in EXPTIME. It should be noted here that this complexity is in terms of the size of database constraints, mapping rules, and TBox axioms. This is often called *schema complexity*, as opposed to *data complexity*.

- Our mapping language is based on *GF*, the guarded fragment of first-order logic. Nevertheless, end-users can define mapping rules without knowing the guarded fragment. In fact, we propose a user-friendly algebraic language that is contained in the guarded fragment.

- We propose a solution to overcome the mismatch between value-based keys used in databases and abstract individual names in description logic, which was illustrated by the previous "Paris Hilton" example.

This chapter is organized as follows. The next section discusses related work. Section 2.3 illustrates the concepts of this chapter by means of a simple example. Section 2.4 introduces some preliminary definitions. Section 2.5 introduces *Entity-expressions* and *Relationship-expressions*, which are the building blocks for our mapping rules that are introduced in Section 2.6. The decidability of some important reasoning problems is established in Section 2.7. Section 2.8 concludes the chapter. All proofs are available in Appendix B.

2.2. Related Work

Recent years have seen active research on disclosing relational databases to ontologies or the semantic Web [100, 101, 104, 119]. The most commonly used rules used for mapping relational databases to ontologies have the form

$$\forall \vec{x}\,(\varphi(\vec{x}) \rightarrow \exists \vec{y}\psi(\vec{x},\vec{y}))\,, \tag{2.1}$$

where the *left-hand side* φ is a conjunction of atoms over the database schema, and the *right-hand side* ψ is a conjunction of atoms over the vocabulary (concept names and role names) of the ontology. A closed formula of the form (2.1) is called a *GLAV mapping* or, in the database literature, a *tuple-generating* *dependency (tgd).* A tgd is *full* if no existential quantifier occurs in it. A *GAV tgd* is a full tgd whose right-hand side is a single atom. A *LAV tgd* is a tgd whose left-hand side is a single atom. Bienvenu [21] uses $\mathrm{GAV}^{\neg,\neq}$ tgds, which extend GAV tgds by allowing negated atoms and inequalities in the left-hand side. In [94], the left-hand side is allowed to be an arbitrary SQL query.

Most studies in OBDA have adopted the relational database model; recent notable exceptions are [24, 29, 82] which also consider NoSQL databases.

As explained in the introduction, our incentive for studying OBDA is that it can provide an ontologically based approach to data quality. This involves identifying inconsistency and redundancy in OBDA mappings, as well as testing for other (un)desirable properties [40, 75, 94]. A recent survey on OBDA [119] mentions data quality as an important research direction.

When mapping relational databases to ontologies, a difficulty is that the relational database model uses value-based primary keys to identify tuples, while description logics use abstract individual names to refer to objects, possibly in combination with the Unique Name Assumption. For example, in a database setting, a fact $R(\underline{a_1, \ldots, a_k}, \vec{b})$ may represent an entity (e.g., a student, a teacher, a course) in the real world. The relation name R together with the underlined primary-key value uniquely identify this entity. Such a primary-key value can be composite, i.e., $k \geq 1$. If we want to represent the same entity in the DL setting, we have to create a unique, atomic individual name for it. This issue is nicely discussed in [94, p. 149], where a solution is proposed that uses ordered tuples of database constants for individual names. Our approach resembles the latter solution, with one significant extension: we also use attributes, as illustrated by the "Paris Hilton" example in Section 2.1.

Another problem that often emerges in data integration is that a same real-world entity may be recorded multiple times in a database with different identifiers, in which case a database-to-ontology mapping should involve some unification [33, 120]. This unification problem, however, is outside the scope of our study.

2.3. Introductory Example

Before starting the technical development, we introduce our mapping language by means of a simple example. A fact $\mathit{ENROLLED}(c, f, \ell, p, y)$ in our example database means that student (f, ℓ) is currently enrolled in course c and took the prerequisite course p in the year y. A fact $\mathit{TAUGHT\text{-}BY}(c, f, \ell, h, s)$ means that the course c is taught by (f, ℓ) and takes place at every hour h during semester s. The same course can be taught more than once in a week.

ENROLLED

Course	*First*	*Last*	*Prerequisite*	*Year*
CS402	Tom	Jones	CS311	2008
CS402	Tom	Jones	CS401	2009

TAUGHT-BY

Course	*First*	*Family*	*Hour*	*Semester*
CS402	David	Maier	Mon. 10am	Spring
CS402	David	Maier	Tue. 10am	Spring

We will identify all persons by their first and last names, using the attributes *First* and *Last*. The operator $\pi_{\mathit{First},\mathit{Last}}$ takes the projection on *First* and *Last*. Since the table *TAUGHT-BY* uses the attribute *Family* for last names, we rename that attribute by means of the renaming operator $\delta_{\mathit{Family}\to\mathit{Last}}$. Let

$$S \coloneqq \pi_{\mathit{First},\mathit{Last}}\mathit{ENROLLED} \text{ and } T \coloneqq \pi_{\mathit{First},\mathit{Last}}(\delta_{\mathit{Family}\to\mathit{Last}}\mathit{TAUGHT\text{-}BY}).$$

Thus, S is the set of persons that are students, and T is the set of persons that are teachers. We will identify all courses by the attribute *Course*, which necessitates the use of the renaming operator $\delta_{\mathit{Prerequisite}\to\mathit{Course}}$. Let

$$\begin{aligned} C \coloneqq\ \pi_{\mathit{Course}}\mathit{ENROLLED}\ &\cup\ \pi_{\mathit{Course}}(\delta_{\mathit{Prerequisite}\to\mathit{Course}}\mathit{ENROLLED}) \\ &\cup\ \pi_{\mathit{Course}}\mathit{TAUGHT\text{-}BY} \end{aligned}$$

Thus, C is the set of all courses. We are now ready to give three mapping rules for populating concept names $\mathsf{Student}$, $\mathsf{Teacher}$, and Course:

$$S : \mathsf{Student}, \qquad T : \mathsf{Teacher}, \qquad C : \mathsf{Course}.$$

Since S and T use the same attributes, it is possible that some students are also teachers. This would result in a knowledge base falsifying $\mathsf{Teacher} \sqsubseteq$

$\neg$Student. Our mapping language captures negation by means of the difference operator $-$. For example, one could declare

$$S - T : \mathsf{PersonWhoDoesNotTeach}.$$

Finally, we show a mapping rule for roles. Assume we are in the spring semester, and we are interested in who attends which course in the current semester. We show a mapping rule for the role name *attends*:

$$\big[S,\ C \ltimes \big(\sigma_{Semester=\texttt{Spring}} \mathit{TAUGHT\text{-}BY}\big),\ \ \mathit{ENROLLED}\big] : \mathit{attends} \tag{2.2}$$

S gets all students. Next, $C \ltimes \sigma_{Semester=\texttt{Spring}}(\mathit{TAUGHT\text{-}BY})$ gets all courses that take place in the spring semester. Technically, $\ltimes$ is the semijoin operator, whose effect is to return those courses in C that join with some tuple in the selection $\sigma_{Semester=\texttt{Spring}} \mathit{TAUGHT\text{-}BY}$. Then, the third argument *ENROLLED* specifies that a student in the first argument has to be related to a course in the second argument if they occur together in a same tuple of *ENROLLED*. The last argument, *attends*, specifies the role name for student-course pairs so obtained.

The following mapping rule is equivalent to (2.2), but applies the semester condition on the third argument:

$$\big[S,\ C,\ \ \mathit{ENROLLED} \ltimes \pi_{Course}\big(\sigma_{Semester=\texttt{Spring}}(\mathit{TAUGHT\text{-}BY})\big)\big] : \mathit{attends}$$

2.4. Preliminaries

2.4.1 Preliminaries from Database Theory

We assume a denumerable set **att** of *attributes*, a denumerable set **dom** of *constants*, and a denumerable set **relname** of *relation names*. We assume a total order $\leq_{\mathbf{att}}$ on **att**. We assume a total function *sort* with domain **relname** that maps every relation name to a finite set of attributes.

Let U be a finite set of attributes. A *tuple over* U is a total mapping from U to **dom**. A *relation over* U is a finite set of tuples over U. An *attribute renaming for* U is a total injective function from U to **att**. We write $A_1, A_2, \ldots, A_n \to B_1, B_2, \ldots, B_n$ for the attribute renaming f such that

$f(A_i) = B_i$ for $i \in \{1, \ldots, n\}$ and f is the identity on other attributes. If t is a tuple over U and f is an attribute renaming, then $f(t)$ denotes the tuple s over $\{f(A) \mid A \in U\}$ such that for every $A \in U$, $s(f(A)) = t(A)$. For example, if $t = \{A : a,\ B : b,\ C : c\}$ and $f = AB \to BD$, then $f(t) = \{B : a,\ D : b,\ C : c\}$.

A *database schema* is a finite set of relation names. In the rest of the chapter we will suppose a fixed database schema. A *database instance* $\mathbf{db}$ associates, to each relation name R, a finite relation over $sort(R)$, denoted $R^{\mathbf{db}}$. A database instance is also called a *database*.

2.4.2 Relational Algebra

The operations of the relational algebra [6] are selection σ, projection π, (natural) join $\bowtie$, semijoin $\ltimes$, renaming δ, union $\cup$, and difference $-$. Conditions in selections can be equalities between attribute values and constants, that is, $\sigma_{A=B}$ and $\sigma_{A=c}$. A projection $\pi_X E$ takes the projection of E on the set X of attributes. A renaming $\delta_f E$, where f is an attribute renaming, applies f to all tuples in E. A join $E \bowtie F$ returns all tuples that can be constructed by taking the union of two tuples, one from E and one from F, that agree on their common attributes. A semijoin $E \ltimes F$ returns every tuple of E that agrees with some tuple of F on their common attributes. In the full relational algebra, $\ltimes$ is not a primitive operator, because it can be expressed as a projection of a join: $E \ltimes F \equiv \pi_{sort(E)}(E \bowtie F)$. However, semijoin is a primitive operator in the *semijoin algebra*, which allows semijoins but disallows joins. The formal semantics of all operators, which is given in Appendix A, defines $eval(E, \mathbf{db})$, the relation to which an algebra expression E on a database $\mathbf{db}$ evaluates.

2.4.3 Specialization of Predicate Logic to Database Theory

We recall some slight differences between the uses of predicate logic in mathematical logic and in database theory, particularly the fact that interpretations are always Herbrand interpretations and that function symbols do not appear in the vocabulary, except in the form of constants. These differences will be relevant in the technical treatment.

Let $\varphi(x_1, \ldots, x_n)$ a first-order formula with free variables $x_1, \ldots, x_n$. Let ν be a valuation over $\{x_1, \ldots, x_n\}$ such that for every $i \in \{1, \ldots, n\}$, $\nu(x_i) = a_i$. If we write $\mathbf{db} \models \varphi(a_1, \ldots, a_n)$, then we mean that $\mathfrak{A}, \nu \models \varphi$, using standard first-order logic semantics for $\models$, for the following structure $\mathfrak{A}$:

- The universe of discourse $\mathbf{A}$ is any set that contains all values that appear in the database or in φ. We will require that our formulas are domain-independent [6, Definition 5.3.7], which means that it does not matter what are the other values in $\mathbf{A}$.
- Every relation name R is interpreted by $R^{\mathbf{db}}$, that is, $R^{\mathfrak{A}} = R^{\mathbf{db}}$.
- Constant symbols are interpreted as themselves, that is, $c^{\mathfrak{A}} = c$ for every constant symbol c. This is also called a Herbrand interpretation of constants.

Note that our vocabularies have no function symbols of arity 1 or greater.

2.4.4 The Guarded Fragment of First-Order Logic

The *guarded fragment* of first-order logic [11, 53], denoted by GF, was first introduced by Andréka et al. as a a generalization of some nice reasoning properties of modal logics. Since Description Logics are closely related to modal logics [16], the guarded fragment provides a nice setting to study Description Logics. The guarded fragment satisfies attractive properties: the problem of logical implication is decidable in 2EXPTIME, and in EXPTIME if the vocabulary is assumed to be fixed; GF has the finite model property and the tree model property. It should also be noted that GF is closely related to the semijoin algebra [73], a language that will be used in our setting.

We define GF as the following restriction of predicate calculus, with equality:

- every quantifier-free formula belongs to GF;
- if $\varphi(\vec{x}, \vec{y})$ belongs to GF and $R(\vec{x}, \vec{y})$ is a relation atom in which all free variables of φ actually occur, then the formulas $\exists \vec{y}\,(R(\vec{x}, \vec{y}) \wedge \varphi(\vec{x}, \vec{y}))$ and $\forall \vec{y}\,(R(\vec{x}, \vec{y}) \rightarrow \varphi(\vec{x}, \vec{y}))$ belong to GF; and

- GF is closed under $\wedge$, $\vee$, $\neg$, $\rightarrow$, $\leftrightarrow$.

A first-order formula is called *guarded* if it belongs to GF.

The guarded fragment GF can capture several modal logics and most Description Logics. On the other hand, it cannot express some common database constraints like transitivity or functional dependencies.

2.5. Entity-Expressions and Relationship-Expressions

In this section, we introduce Entity-expressions and Relationship-expressions, which will be used in Section 2.6 to construct mapping rules. The following definitions are relative to a fixed database schema and description logic vocabulary (i.e., a finite set of concept names and role names).

Definition 2.1 (Entity-Expression).
Entity-expressions (EEs) are recursively defined as follows:

1. Every relation name R is an EE with sort $sort(R)$.

2. If E is an EE and $X \subseteq sort(E)$, then $\pi_X E$ is an EE with $sort(\pi_X E) = X$.

3. If E is an EE and f is an attribute renaming for $sort(E)$, then $\delta_f E$ is an EE with $sort(\delta_f E) = \{f(A) \mid A \in sort(E)\}$.

4. If E is an EE, $A, B \in sort(E)$, and $c \in \mathbf{dom}$, then $\sigma_{A=c}E$ and $\sigma_{A=B}E$ are EEs with $sort(\sigma_{A=c}E) = sort(\sigma_{A=B}E) = sort(E)$.

5. If E_1 and E_2 are EEs such that $sort(E_1) = sort(E_2)$, then $E_1 \cup E_2$ and $E_1 - E_2$ are EEs with $sort(E_1 \cup E_2) = sort(E_1 - E_2) = sort(E_1)$.

6. If E_1 and E_2 are EEs, then $E_1 \ltimes E_2$ is an EE with $sort(E_1 \ltimes E_2) = sort(E_1)$.

$\triangleleft$

Example 2.1
Let **db** be the following database.

$$R \begin{array}{|ccc} A & B & C \\ \hline a & b & c \\ d & e & f \end{array} \qquad S \begin{array}{|ccc} B & C & D \\ \hline b & c & e \\ b & b & m \end{array} \qquad T \begin{array}{|ccc} C & E & F \\ \hline l & e & d \\ f & g & g \end{array}$$

We show next the interpretation of two Entity-Expressions:

$$\sigma_{B=C}S \begin{array}{|ccc} B & C & D \\ \hline b & b & m \end{array}$$

and

$$(\pi_C R \cup \pi_C S) - \pi_C T \begin{array}{|c} C \\ \hline b \\ c \end{array} .$$

$\triangleleft$

Note that Entity-expressions cannot use the join operator $\bowtie$. The fragment of relational algebra that replaces the join operator $\bowtie$ with the semijoin operator $\ltimes$ is known as the *semijoin algebra*. An important result by Leinders et al. [73] states that the semijoin algebra is contained in *GF*. Our setting slightly differs from this earlier work because we have attribute renamings δ_f and selections of the form $\sigma_{A=c}$, both of which are not present in [73]. The proof of the following Theorem 2.1 translates Entity-expressions in domain-independent formulas in the guarded fragment. It differs from [73] in that it admits constants and does not use the formulas $\mathbb{G}_k(x_1, \ldots, x_k)$ introduced by Leinders et al. for defining the guarded k-tuples of a structure. In the translation from the semijoin algebra to first-order logic, the expression $\mathbb{G}_k(x_1, \ldots, x_k)$ is used as a guard. Our translation avoids the use of such a guard by first rewriting algebra expressions in union normal form, which is conceptually simple but comes at the price of an exponential blowup.

Theorem 2.1. *For every Entity-expression E with sort$(E) = \{A_1, \ldots, A_n\}$, it is possible to construct a domain-independent formula $\varphi(x_1, \ldots, x_n)$ in GF such that for every database* **db**, *for all $a_1, \ldots, a_n \in$* **dom**, *$\{A_1 : a_1, \ldots, A_n : a_n\} \in eval(E, \mathbf{db})$ if and only if $\mathbf{db} \models \varphi(a_1, \ldots, a_n)$.*

Since decidability of of logical implication in *GF* carries over the semijoin algebra, we obtain the following corollaries of Theorem 2.1.

Definition 2.2 (Satisfiable Entity-Expression).
An Entity-expression E is called *satisfiable* if and only if there exists a database instance $\mathbf{db}$ such that $eval(E, \mathbf{db}) \neq \emptyset$.

$\triangleleft$

Corollary 2.2. *The following problem is decidable: Given an Entity-expression E, is E satisfiable?*

The join operator $\bowtie$ can be used in Relationship-expressions, capturing a common intuition that relationships are places where entities "join."

Definition 2.3 (Relationship-Expression).
Relationship-expressions (REs) are recursively defined as follows:

- Every Entity-expression is an RE.
- If E_1 and E_2 are REs, then $E_1 \bowtie E_2$ is an RE.
- The set of REs is closed under the operators $\sigma_{A=c}$, $\sigma_{A=B}$, δ_f, $\cup$, and $-$.

$\triangleleft$

Note that the set of Relationship-expressions is not closed under projection or semijoin; for example, $T \ltimes (R \bowtie S)$ and $\pi_{AB}(R \bowtie S)$ are not Relationship-expressions. That is, Relationship-expressions cannot project out attributes. Intuitively, Relationship-Expressions express relationships among entities that result from Entity-expressions; the inability to use projection ensures that such entities will not be truncated and become unrecognizable. In this way, Theorem 2.1 remains valid if we replace "Entity-expression" with "Relationship-expression" in its statement. A corollary is that satisfiability of Relationship-expressions is also decidable.

Corollary 2.3. *The following problem is decidable: Given a Relationship-expression E, is E satisfiable?*

For every fixed finite vocabulary, the satisfiability problem for *GF* is in EXPTIME. It is proved in [73, Theorem 9] that this complexity upper bound carries over to the semijoin algebra. It should be noted that our translation from semijoin algebra to *GF* first rewrites expressions in union normal form, which comes at the price of an exponential blowup. Nevertheless, we believe that this blowup can be avoided by an approach similar to [73].

2.6. The Mapping Language

We will now introduce the notion of *Database-to-ABox Dependency (DAD)*. From here on, all definitions are relative to a database schema **S**, a description logic vocabulary $\mathbf{C} \cup \mathbf{R}$ (i.e., a set of concept names and role names), and a description logic DL. A DAD can be of two sorts: a *Concept DAD (CDAD)* takes as input a database and returns a set of concept assertions; a *Role DAD (RDAD)* takes as input a database and returns a set of role assertions.

CDAD

The following definition introduces the syntax and semantics for a Concept DAD. The semantics of CDAD relies on a function ι from the set of all tuples to **I**, the set of individual names.

Definition 2.4 (Concept DAD).
A *Concept DAD (CDAD)* is an expression $E : C$ where E is an Entity-expression (over the database schema **S**) and C is a concept name in **C**.

We assume a denumerable set **I** of *individual names*. We assume an injective function ι from the set of all tuples (taken over all finite subsets of **att**) to the set of individual names.

Let **db** be a database. The *set of concept assertions generated by* $E : C$ *from* **db** is the following:

$$\{ \ \iota(t) : C \ \ \mid t \in \mathit{eval}(E, \mathbf{db}) \ \}.$$

◁

RDAD

The syntax for a Role DAD is slightly more complex: it is a sequence of two Entity-expressions, one Relationship-expression, and a role name r in **R**.

Informally, given a database, such a Role DAD generates a role assertion $(a, b) : r$ whenever a and b belong, respectively, to the result of the first and the second Entity-expression, and together fit the Relationship-expression. We give an example, and then provide a formal definition.

Example 2.2
Consider the following data from the Mathematics Genealogy Project at `http://www.genealogy.ams.org`.

PHD	*First*	*Last*	*Year*	*AdvisorFirst*	*AdvisorLast*
	Jan	Chomicki	1990	Tomasz	Imielinski
	Tomasz	Imielinski	1981	Witold	Lipski
	Witold	Lipski	1968	Wiktor	Marek

Assume that no two distinct persons in this database agree on their first and last names. Let f be a renaming such that $f(\mathit{First}) = \mathit{AdvisorFirst}$ and $f(\mathit{Last}) = \mathit{AdvisorLast}$. Thus, the inverse of f, denoted f^{-1}, maps *AdvisorFirst* and *AdvisorLast* to, respectively, *First* and *Last*. The following Entity-expression P gets first and last names of all persons in the database:

$$P \coloneqq \pi_{\{\mathit{First},\mathit{Last}\}}\mathit{PHD} \cup \delta_{f^{-1}}\big(\pi_{\{\mathit{AdvisorFirst},\mathit{AdvisorLast}\}}\mathit{PHD}\big).$$

It is significant to note that Wiktor Marek will be added with attributes *First* and *Last*, even though he does not appear with these attributes in the *PHD* table. The following RDAD populates the role name *SupervisedBy*:

$$[P, P/f, \mathit{PHD}] : \mathit{SupervisedBy}. \tag{2.3}$$

Given a database $\mathbf{db}$, this rule will add a role assertion $(s, t) : \mathit{SupervisedBy}$ to the ABox whenever $s, t \in \mathit{eval}(P, \mathbf{db})$ such that some tuple of $\mathit{eval}(\mathit{PHD}, \mathbf{db})$ includes both s and $f(t)$. For example, if $s_0 = \{\mathit{First} : \texttt{Witold}, \mathit{Last} : \texttt{Lipski}\}$ and $t_0 = \{\mathit{First} : \texttt{Wiktor}, \mathit{Last} : \texttt{Marek}\}$, then $(s_0, t_0) : \mathit{SupervisedBy}$ will be added to the ABox because $s_0 \cup f(t_0) = \{\mathit{First} : \texttt{Witold}, \mathit{Last} : \texttt{Lipski}, \mathit{AdvisorFirst} : \texttt{Wiktor}, \mathit{AdvisorLast} : \texttt{Marek}\}$ is included in the last tuple of the *PHD* table.

$\triangleleft$

Definition 2.5 (Role DAD).
A *Role DAD (RDAD)* is an expression of the form

$$[E_1/f_1, E_2/f_2, E] : r$$

where E_1 and E_2 are Entity-expressions, f_1 and f_2 are attribute renamings, E is a Relationship-expression such that $sort(\delta_{f_1} E_1) \cup sort(\delta_{f_2} E_2) \subseteq sort(E)$, and r is a role name in $\mathbf{R}$.

If f_1 or f_2 is the identity, it can be omitted. Such an RDAD is called *join-free* if $\bowtie$ does not occur in it (but $\ltimes$ can occur). For example, the RDADs (2.2) and (2.3) are both join-free.

To define the semantics, let $\mathbf{db}$ be a database. The *set of role assertions generated by* $[E_1/f_1, E_2/f_2, E] : r$ *from* $\mathbf{db}$ is the following:

$$\{\ (\iota(t_1), \iota(t_2)) : r \ \mid \ \begin{array}{l} t_1 \in \mathit{eval}(E_1, \mathbf{db}),\ t_2 \in \mathit{eval}(E_2, \mathbf{db}), \\ \text{and } f_1(t_1) \cup f_2(t_2) \subseteq t \text{ for some } t \in \mathit{eval}(E, \mathbf{db}) \end{array} \ \}.$$

$\lhd$

2.7. Reasoning Problems

We now move from a single CDAD or a single RDAD to sets of CDADs and RDADs, and introduce some reasoning problems.

Definition 2.6 (The ABox $\mathcal{M}(\mathbf{db})$).
Let $\mathbf{db}$ be a database. Let $\mathcal{M}$ be a set of CDADs and RDADs. We write $\mathcal{M}(\mathbf{db})$ for the smallest ABox that contains all concept and role assertions generated from $\mathbf{db}$ by the CDADs and RDADs in $\mathcal{M}$.

A CDAD is said to be *active on* $\mathbf{db}$ if it generates at least one concept assertion from $\mathbf{db}$; an RDAD is *active on* $\mathbf{db}$ if it generates at least one role assertion from $\mathbf{db}$.

$\lhd$

In the following definition, one may think of Σ as a set of database constraints. However, when studying problems like SATISFIABILITY (see below), we may add to Σ some desirable properties, like $\exists \vec{x} R(\vec{x})$ if we are asking for a satisfying database in which R is nonempty.

Definition 2.7 (DB2KB specification).
Fix two vocabularies, τ_1 and τ_2, of first-order logic without function symbols. A *DB2KB* (or *OBDA specification*) is a triple $(\Sigma, \mathcal{M}, \mathcal{T})$ where

- Σ is a set of closed first-order formulas over the database schema, using the vocabulary τ_1;
- $\mathcal{M}$ is a set of CDADs and RDADs, where Entity-Expressions and Relationship-Expressions are defined over τ_1, and concept and role constructs are defined over τ_2; and
- $\mathcal{T}$ is a DL TBox, using the vocabulary τ_2.

$\triangleleft$

Console and Lenzerini [40] introduced the notions of *faithfulness* and *protection* for characterizing data quality in OBDA. These notions are recalled next, together with the well-known notion of *satisfiability*. For aesthetic reasons, we state all questions in the form "Is there a database such that...?". Therefore, we are asking for the complement of faithfulness and protection as defined in [40]. This difference is not fundamental, because we are aiming at decidability results, and the class of decidable languages is closed under complement. Finally, the notion of *global-consistency* appeared in [75].

INPUT: A DB2KB $(\Sigma, \mathcal{M}, \mathcal{T})$.

QUESTIONS:

- Satisfiability: Is there a database $\mathbf{db}$ such that $\mathbf{db} \models \Sigma$ and the knowledge base $\langle \mathcal{T}, \mathcal{M}(\mathbf{db}) \rangle$ is consistent?
- Non-Faithfulness: Is there a database $\mathbf{db}$ such that the knowledge base $\langle \mathcal{T}, \mathcal{M}(\mathbf{db}) \rangle$ is consistent but $\mathbf{db} \not\models \Sigma$?
- Non-Protection: Is there a database $\mathbf{db}$ such that $\mathbf{db} \models \Sigma$ but the knowledge base $\langle \mathcal{T}, \mathcal{M}(\mathbf{db}) \rangle$ is inconsistent?
- Global-Consistency: Is there a database $\mathbf{db}$ such that $\mathbf{db} \models \Sigma$, all CDADs and RDADs of $\mathcal{M}$ are active on $\mathbf{db}$, and the knowledge base $\langle \mathcal{T}, \mathcal{M}(\mathbf{db}) \rangle$ is consistent?

Informally, a "yes"-answer to Non-Protection tells us that the ontology has some constraints not implied by Σ. Recall from Section 2.1 that the discovery of such constraints may be significant in data quality assessments.

As mentioned just in front of Definition 2.7, Σ may contain desirable properties in addition to database constraints. For example, for Satisfiability, we can use Σ to express that the database **db** must be nonempty, as follows. For every $R \in \mathbf{S}$, define $\varphi_R \coloneqq \exists \vec{x} R(\vec{x})$, and add to Σ the formula $\bigvee_{R \in \mathbf{S}} \varphi_R$, or even stronger, $\bigwedge_{R \in \mathbf{S}} \varphi_R$. These are unlikely to be database constraints, because most database applications start with an empty database, which thus has to be legal. However, database relations are not intended to remain empty at all times, which can thus be expressed as a desirable property in Σ. A knowledge-worker might even use Σ to assert knowledge like

$$\neg \exists h \exists z \, \mathit{TAUGHT\text{-}BY}(\mathtt{CS402}, \mathtt{Jeffrey}, \mathtt{Ullman}, h, z)$$

and then run Satisfiability to check consistency of her knowledge.

The above problems can be shown to be undecidable in general [40, Theorem 1]. The following theorem shows their decidability under some restrictions on the input, which will be discussed after the theorem. A technical crux in the proof of Theorem 2.4 concerns the switch from database tuples to DL individual names. As discussed in Section 2.2, entities that are represented by tuples on the database side must be mapped to atomic, individual names on the ontology side. We solve this issue by choosing an appropriate encoding for the injective function defined in Definition 2.4.

Theorem 2.4. Satisfiability, Non-Faithfulness, Non-Protection, *and* Global-Consistency *are decidable problems if their inputs are restricted to DB2KBs* $(\Sigma, \mathcal{M}, \mathcal{T})$ *with the following properties:*

- $\mathcal{T}$ *can be effectively expressed in GF;*
- *every formula in* Σ *is in GF; and*
- *all RDADs in* $\mathcal{M}$ *are join-free.*

The satisfiability problem for *GF* with constants is EXPTIME-complete when the arities of all relation names are fixed [111]. The EXPTIME-hard lower bound obviously carries over to the problems in Theorem 2.4. The EXPTIME-upper bound does not, insofar as Theorems 2.1 and 2.4 use exponential translations of CDADs and RDADs in *GF*. However, it is a plausible conjecture that membership in EXPTIME can be obtained along the lines of the proof of [73, Theorem 9].

We briefly discuss the restrictions in the statement of Theorem 2.4. The restriction that the input TBoxes $\mathcal{T}$ must be expressible in *GF* may be automatically fulfilled by the description logic DL under consideration. Indeed, many expressive description logics can be expressed in *GF* [15,112], an example being $\mathcal{ALC}$ [16, p. 46]. The requirement that Σ is in *GF* still allows expressing many interesting properties and database constraints, like non-emptiness of relations and inclusion dependencies, as well as all Boolean combinations of these (because *GF* is closed under Boolean combinations). On the other hand, *GF* does not include common database constraints like primary keys or functional dependencies. Theorem 2.4 imposes no restrictions on CDADs, but RDADs are restricted to be join-free. Our example RDADs (2.2) and (2.3) are join-free. Informally, join-freeness demands that whenever an RDAD puts two entities together in a role, then these entities should already occur together in some database relation. This restriction is plausibly satisfied for database schemas that are obtained from Entity-Relationship diagrams that already capture such roles by relationships. The restriction can be prohibitive though if one wants to combine in a role two entities that are unrelated in the Entity-Relationship diagram.

Finally, we show a result telling us that database constraints can be obtained from an ontology, given a mapping $\mathcal{M}$. As we argued in Section 2.1, this may be of interest in data cleaning applications where some database constraints may be missing.

Theorem 2.5. *Let $(\Sigma, \mathcal{M}, \mathcal{T})$ be a DB2KB such that $\Sigma = \emptyset$ and $\mathcal{T}$ is a DL-Lite$_{core}$ TBox. It is possible to construct a finite set Σ' of closed first-order formulas such that for every database* **db**, $\mathbf{db} \models \Sigma'$ *if and only if $\langle \mathcal{T}, \mathcal{M}(\mathbf{db}) \rangle$ is a consistent knowledge base. Moreover, if every RDAD in $\mathcal{M}$ is join-free, then Σ' is in GF.*

2.8. Conclusion

The language of CDADs and RDADs allows expressing database-to-ontology mappings in a user-friendly way. The language is based on the semijoin algebra, which is embedded in the guarded fragment of first-order logic. This results in decidability of some important reasoning problems. Since CDADs and RDADs allow full negation, they can express mappings that are not GLAV mappings. On the other hand, the GLAV mapping

$$\forall x \forall y \forall z \, (R(x,y) \wedge R(y,z) \wedge R(z,x) \rightarrow C(x))$$

is not guarded and cannot be expressed as a CDAD. In future research, we plan to explore in more depth the practice of using ontological knowledge in database repairing, database cleaning, and consistent query answering. We also plan to investigate complexity bounds for the decidable problems in Theorem 2.4.

CHAPTER 3

Assertion Ranking in Ontologies

Remark 3.0
The content of this chapter has been presented at DL 2020 [90].

◁

3.1. Motivation

Inconsistency is an important and recurrent problem in today's database and knowledge base systems. Data errors are practically unavoidable in systems that integrate data coming from different sources. Two approaches to tackle this problem are data cleaning [63, 117] and consistent query answering (CQA) [118]. The latter approach uses the notion of *repair*, which is a consistent knowledge base obtained by making some minimal amount of data corrections. A repair is conceptually not different from a cleaned knowledge base. However, whereas the process of data cleaning is supposed to end in a single cleaned knowledge base, the CQA approach allows for the possibility of multiple repairs (or possible worlds). In the Description Logics framework, different repairs correspond to different ABoxes, each one consistent with respect to a given fixed TBox. When we move from a single-world perspective to a possible-worlds perspective, different semantics for logical reasoning become possible [22, 25, 30, 54, 74, 79]. Eventually, all these semantics address the following central problem: Given a logical sentence φ, assign some degree of truthfulness to φ. Is φ true in some, most, or all repairs? What is the probability of φ being true? Is there strong support for—or resistance against—the sentence φ? Throughout this chapter, we use the term "truthfulness" in a loose and informal way; we could have used other terms instead: credibility, veracity, accuracy...

In this chapter, we present a new principled framework for evaluating the relative truthfulness of ABox assertions (i.e., the sentence φ is an ABox assertion in our framework). By relative, we mean that we are merely interested in ranking ABox assertions: Given two ABox assertions α and β, is α more, less, or equally truthful than β? Our framework is different from CQA in that it does not use the notion of repair. The framework assumes that there is some initial weight function over the ABox assertions, called *credibility function*, which models the opinion of domain experts concerning the truthfulness of assertions. In their assessments, however, experts may have difficulties to fully understand and take into account the logic that is expressed by, possibly extensive, TBoxes. To overcome this difficulty, we propose an automated mathematical method for adjusting the experts' initial credibility function by taking into account the ontological logic of the TBox: strong logical support for an assertion α should increase its credibility, while strong logical support for $\neg\alpha$ should decrease α's credibility. Eventually, our method results in a ranking of ABox assertions in terms of truthfulness, taking into account both the experts' credibility function and the TBox logic. In our framework, we assume that the TBox has been validated by a multitude of experts and is therefore error-free.

This chapter is organized as follows. Section 3.2 presents a guiding example. Section 3.3 discusses related work. Section 3.4 introduces our general theoretical framework, and Section 3.5 presents a particular instantiation of this framework. Section 3.6 shows that our method guarantees some desirable properties. Section 3.7 studies in more detail some theoretical questions and computational tasks raised by this instantiation. Section 3.8 discuses about the user credibility function and aggregate operators. Finally, Section 3.9 concludes the chapter.

3.2. Motivating Example

Consider the following knowledge base $\mathcal{K}_0 = \langle \mathcal{T}_0, \mathcal{A}_0 \rangle$.

$$\mathcal{T}_0 = \left\{ \begin{array}{rcl} \mathsf{Professor} & \sqsubseteq & \mathsf{Person}, \\ \mathsf{Student} & \sqsubseteq & \mathsf{Person}, \\ \mathsf{Person} & \sqsubseteq & \neg\mathsf{Course}, \\ \mathsf{Student} & \sqsubseteq & \neg\mathsf{Professor}, \\ \exists teaches & \sqsubseteq & \mathsf{Professor}, \\ \exists attends & \sqsubseteq & \mathsf{Student}, \\ \exists teaches^- & \sqsubseteq & \mathsf{Course}, \\ \exists attends^- & \sqsubseteq & \mathsf{Course} \end{array} \right\} \qquad \mathcal{A}_0 = \left\{ \begin{array}{rcl} \mathsf{John} & : & \mathsf{Professor} \\ \mathsf{Ava} & : & \mathsf{Student} \\ \mathsf{DB2} & : & \mathsf{Course} \\ \mathsf{KR} & : & \mathsf{Course} \\ (\mathsf{John}, \mathsf{DB2}) & : & teaches \\ (\mathsf{John}, \mathsf{KR}) & : & attends \\ (\mathsf{Ava}, \mathsf{IA}) & : & attends \\ (\mathsf{Bob}, \mathsf{KR}) & : & attends \end{array} \right\}$$

This knowledge base is inconsistent, because from $(\mathsf{John}, \mathsf{KR}) : attends$, we can infer $\mathsf{John} : \mathsf{Student}$ by means of the axiom $\exists attends \sqsubseteq \mathsf{Student}$. Then, by means of the axiom $\mathsf{Student} \sqsubseteq \neg\mathsf{Professor}$, we can infer $\mathsf{John} : \neg\mathsf{Professor}$, which obviously contradicts the first assertion in $\mathcal{A}_0$. In conclusion, $(\mathsf{John}, \mathsf{KR}) : attends$ and $\mathsf{John} : \mathsf{Professor}$ contradict one another. The vertices in the directed graph of Fig. 3.1 are the assertions of $\mathcal{A}_0$. A red-colored edge from α to β means that α *refutes* β. On the other hand, a green-colored edge from α to β means that α *supports* β. For example, from $(\mathsf{John}, \mathsf{DB2}) : teaches$ we can infer $\mathsf{John} : \mathsf{Professor}$ by means of the axiom $\exists teaches \sqsubseteq \mathsf{Professor}$. Therefore, $(\mathsf{John}, \mathsf{DB2}) : teaches$ supports $\mathsf{John} : \mathsf{Professor}$. In this example, we assumed that domain experts esteemed that assertions in the ABox were equally credible, represented by a value of 1. Then, our proposed method updates values by balancing the value of each assertion α with respect to the values of α's refuters and supporters. Refuters and supporters of α, respectively, lower and increase α's value by an amount that is proportional to their own value. In Fig. 3.1, we see that $\mathsf{John} : \mathsf{Professor}$ is attributed a higher value than $(\mathsf{John}, \mathsf{KR}) : attends$ because it has more supporters and less refuters.

Figure 3.2 shows the effect of adding $\mathsf{Ava} : \mathsf{Professor}$ to $\mathcal{A}_0$. One can observe that the values of the assertions $(\mathsf{Ava}, \mathsf{IA}) : attends$ and $\mathsf{Ava} : \mathsf{Student}$ have gone down because of conflicts with the new assertion.

Since supporters and refuters are single assertions in this simple exam-

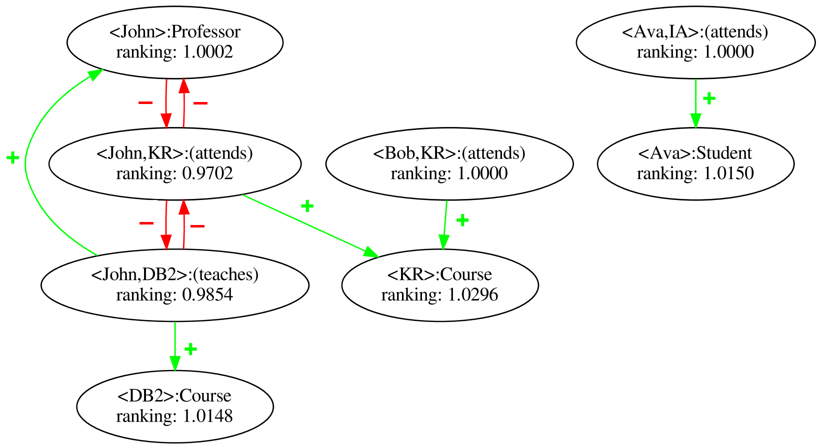

Figure 3.1: ABox assessment obtained by a practical application of our framework.

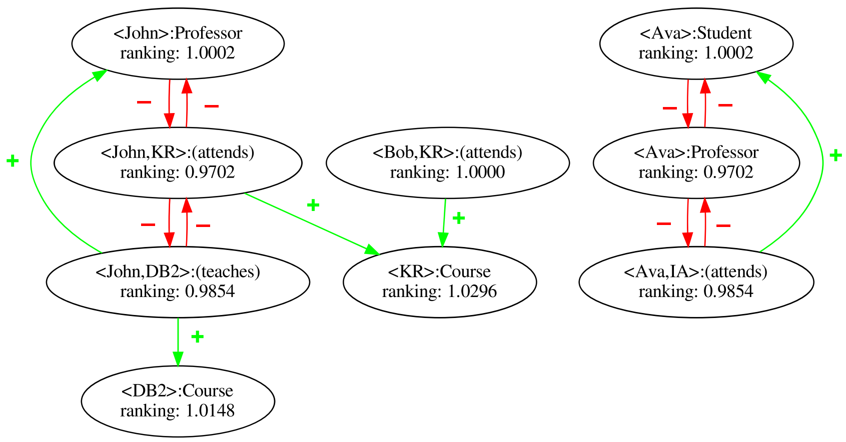

Figure 3.2: New ABox assessment after adding $\mathsf{Ava} : \mathsf{Professor}$.

ple, they can be represented in a directed graph. In our general framework, however, supporters and refuters will be *sets* of assertions. For example, if $A \sqcap B \sqsubseteq \neg C$ is a TBox assertion, then the set $\{(\mathsf{i}, \mathsf{A}), (\mathsf{i}, \mathsf{B})\}$ is a refuter of (i, C), but neither (i, A) nor (i, B) is a refuter on its own.

3.3. Related Work

In Description Logics, different repairs correspond to different ABoxes, each one consistent with respect to a given fixed TBox. When we move from a single ABox to a set of possible ABoxes, different semantics for logical reasoning become possible, including brave semantics [25], ABox Repair (AR) and Intersection ABox Repair (IAR) semantics [74]. These semantics can be enriched by taking into account notions of cardinality [79], preference [22], or probability [30,54]. Some works [45,110] in the Description Logics community have already investigated the problems of finding the best repairs according to some criteria, and of extracting consistent information that best complies with specific needs. Sik Chun Lam et al. have proposed logic-based methods for repairing TBox axioms of unsatisfiable ontologies [70], as well as numerical methods for rewriting and ranking problematic axioms [39].

Ranking the nodes of some sort of knowledge graph in terms of interest, quality, or preference is a recurrent problem in many disciplines. Solving these problems requires to somehow quantify the nodes and their interactions. A quantitative approach, rather than a qualitative one, has been used in argumentation frameworks [9,27,28,62,96], belief revision [103], the Web [68,85], and social networks [43,113].

3.4. Theoretical Framework

Throughout this chapter, we will assume that TBoxes are satisfiable. That is, for every TBox $\mathcal{T}$, the knowledge base $\langle \mathcal{T}, \emptyset \rangle$ is consistent. If a knowledge base $\langle \mathcal{T}, \mathcal{A} \rangle$ is inconsistent, we will assume that the inconsistency is caused by one or more assertions in $\mathcal{A}$. When humans are faced with such an inconsistent knowledge base $\langle \mathcal{T}, \mathcal{A} \rangle$, they may not be able to quickly pinpoint the "wrong" assertion(s) in $\mathcal{A}$, because the inconsistency may only become apparent after an involved reasoning process that uses many axioms and assertions. We present a method for ranking assertions such that higher-ranked assertions are more "truthful" than lower-ranked assertions. Significantly, the ranking only requires some superficial input from the user, which is modeled by the notion of a *credibility function*, a mapping from $\mathcal{A}$ to $\mathbb{R}$. Such a credibility function will be combined with TBox assertions to result in the desired ranking. All definitions that follow are relative to a fixed knowledge base $\langle \mathcal{T}, \mathcal{A} \rangle$ in some fixed Description Logic.

3.4.1 Refuters and Supporters

We define what it means for a set B of assertions to support or refute an assertion α not in B. In our framework, assertions will be higher ranked if they have strong supporters and no strong refuters.

Definition 3.1 (Refuters and Supporters).
The following definitions are relative to a knowledge base $\langle \mathcal{T}, \mathcal{A} \rangle$, and an assertion $\alpha \in \mathcal{A}$.

A *refuter of* α is an inclusion-minimal subset B of $\mathcal{A}$ with the property that $\langle \mathcal{T}, B \rangle$ is consistent and $\langle \mathcal{T}, B \cup \{\alpha\} \rangle$ is inconsistent.

A *supporter of* α is an inclusion-minimal subset B of $\mathcal{A}$ with the property that $\langle \mathcal{T}, B \rangle$ is consistent, $\alpha \notin B$, and $\langle \mathcal{T}, B \rangle \models \alpha$.

$\lhd$

We recall that a logic is *monotonic* if whenever Σ_1 and Σ_2 are two sets of expressions and Σ_1 entails φ, then $\Sigma_1 \cup \Sigma_2$ also entails φ. We can also express the monotonicity of a logic by saying that if φ is an inconsistent expression, then $\varphi \wedge \phi$ is also inconsistent for every ϕ.

Proposition 3.1. *Let $\langle \mathcal{T}, \mathcal{A} \rangle$ be a knowledge base in a monotonic Description Logic, and let $\alpha \in \mathcal{A}$. Then,*

1. *distinct refuters of α are not comparable by set inclusion;*
2. *distinct supporters of α are not comparable by set inclusion; and*
3. *if B is a refuter of α and B' is a supporter of α, then B and B' are not comparable by set inclusion.*

From here on in this chapter, we will assume that all Description Logics considered are monotonic. This is not a big restriction for our framework, since most Description Logics can be defined in terms of first-order logic, which is a monotonic logic.

3.4.2 Aggregated Credibility

As explained in the beginning of Section 3.4, we assume a *credibility function* mapping every assertion α in $\mathcal{A}$ to a real number that can be thought of as the truthfulness associated by an expert to the assertion α. In our setting we assume that higher numeric values are synonym of more quality or more trustworthiness.

We will assume that the credibility function induces a function f from $2^{\mathcal{A}}$ to $\mathbb{R}$ by means of some aggregation operator $\oplus$. Examples of $\oplus$ are MIN, MAX, AVG, SUM. In the theoretical development, we will abstract away from the aggregation operator $\oplus$ and treat f as a basic construct. To keep our framework flexible and general, we do not impose any restrictions on the choice of the credibility function, which can be instantiated and adapted later on to fit a particular setting, for example, as a probability distribution. Nevertheless, in Section 3.8, we propose some concrete realization of the notion of aggregation.

3.4.3 ABox Assessment

An *ABox assessment* for an ABox $\mathcal{A}$ is a total function $\nu : \mathcal{A} \to \mathbb{R}$. Informally, our goal is to define ν such that if two ABox assertions are logically conflicting, then the assertion with the higher ν-value is preferred.

Although credibility functions and ABox assessments are both mappings from $\mathcal{A}$ to $\mathbb{R}$, it is important to understand that they are conceptually distinct. In particular, ABox assessments improve credibility functions by taking into account the axioms of the TBox, via the notions of refuter and supporter defined previously.

Informally, we want that for every $\alpha \in \mathcal{A}$, its value under ν is greater if α has strong supporters, and smaller if α has strong refuters. Here, the strength of a supporter or refuter is recursively determined by the aggregated

ν-values of its elements. This can be expressed as follows, where $\sim$ denotes some proportionality that will be made explicit later on:

$$\nu(\alpha) \sim \sum_{\substack{B \text{ is a} \\ \text{supporter} \\ \text{of } \alpha}} \left(f(B) * \sum_{\beta \in B} \nu(\beta) \right) - \sum_{\substack{B \text{ is a} \\ \text{refuter} \\ \text{of } \alpha}} \left(f(B) * \sum_{\beta \in B} \nu(\beta) \right). \tag{3.1}$$

The expression at the righthand of $\sim$ will be abbreviated $\Sigma^{\pm}(\alpha)$. Informally, $\Sigma^{\pm}(\alpha)$ sums over all supporters and refuters of α. The formulas for supporters and refuters are signed positively and negatively respectively. The magnitude of each supporter's or refuter's contribution is proportional to its aggregated credibility and ABox assessment values. It is significant that (3.1) is recursive, in the sense that ν appears at both the lefthand and righthand of $\sim$.

To shorten notations, we define mappings $T : \mathcal{A} \to 2^{\mathcal{A}}$ and $F : \mathcal{A} \to 2^{\mathcal{A}}$, as follows:

$$\begin{aligned} T(\alpha) &:= \{B \subseteq \mathcal{A} \mid B \text{ is a supporter of } \alpha\}; \\ F(\alpha) &:= \{B \subseteq \mathcal{A} \mid B \text{ is a refuter of } \alpha\}. \end{aligned}$$

Informally, one can think of T and F as *True* and *False* respectively. $B \in T(\alpha)$ is shorthand for *"B is a supporter of α,"* and $B \in F(\alpha)$ is shorthand for *"B is a refuter of α"*. Obviously, the complexity of computing T and F will depend on the underlying Description Logic.

To have a more compact form for $\Sigma^{\pm}(\alpha)$, we define an indicator function I from $\{T, F\} \times 2^{\mathcal{A}} \times \mathcal{A} \times \mathcal{A}$ to $\{0, 1\}$:

$$I(L, B, \alpha, \beta) = \begin{cases} 1 & \text{if } \beta \in B \text{ and } B \in L(\alpha) \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

Thus, $I(T, B, \alpha, \beta)$ is one if B is a supporter of α that contains β, and is zero otherwise. Likewise, $I(F, B, \alpha, \beta)$ is one if B is a refuter of α that contains β, and is zero otherwise. Note incidentally that $\alpha = \beta$ implies $I(L, B, \alpha, \beta) = 0$, because α does not belong to a supporter or a refuter of itself. Then, for $\alpha \in \mathcal{A}$,

$$\Sigma^{\pm}(\alpha) = \sum_{\beta \in \mathcal{A}} \nu(\beta) * \left(\sum_{B \subseteq \mathcal{A}} f(B) * (I(T, B, \alpha, \beta) - I(F, B, \alpha, \beta)) \right). \tag{3.3}$$

3.4.4 Ranking of ABox Assertions

An ABox assessment ν induces a ranking of an ABox, as follows: α is ranked higher than β if $\nu(\alpha) > \nu(\beta)$; and α is ranked equal to β if $\nu(\alpha) = \nu(\beta)$.

We define what it is for two assessments to be rank equivalent

Definition 3.2 (Rank-equivalent assessments).
Let $\mathcal{A}$ be an ABox and $\nu_1, \nu_2 : \mathcal{A} \to \mathbb{R}$ two assessments over $\mathcal{A}$. We say that ν_1 and ν_2 are *rank-equivalent* and write $\nu_1 \overset{\bullet}{\sim} \nu_2$ if for every pair of assertions $\alpha, \beta \in \mathcal{A}$ we have that

$$\nu_1(\alpha) < \nu_1(\beta) \Leftrightarrow \nu_2(\alpha) < \nu_2(\beta).$$

$\triangleleft$

It is easily verified that if $\nu_1 \overset{\bullet}{\sim} \nu_2$, then $\nu_1(\alpha) = \nu_1(\beta)$ implies $\nu_2(\alpha) = \nu_2(\beta)$. Obviously, $\overset{\bullet}{\sim}$ is an equivalence relation on the set of all ABox assessments. In our study, we will be primarily interested in the set of ABox assessments modulo $\overset{\bullet}{\sim}$. That is, we are not so much interested in the real numbers in the range of two different ABox assessments, but rather in their induced rankings.

Example 3.1
Let $\mathcal{A} = \{\alpha_1, \alpha_2, \alpha_3\}$. Let ν_1 and ν_2 be two assessments, as follows:

- $\nu_1(\alpha_1) = 1$, $\nu_1(\alpha_2) = 2$, $\nu_1(\alpha_3) = 3$; and
- $\nu_2(\alpha_1) = 2$, $\nu_2(\alpha_2) = 4$, $\nu_2(\alpha_3) = 8$.

Then, ν_1 and ν_2 are rank-equivalent. If we rank the assertions of $\mathcal{A}$ in ascending order, then both assessments agree that α_1 precedes α_2, and that α_2 precedes α_3.

$\triangleleft$

3.5. Framework Instantiation

We will assume $\mathcal{A} = \{\alpha_1, \ldots, \alpha_n\}$. In this section, we elaborate an instantiation of the framework in Section 3.4. With the previous abbreviations, Eq. (3.1) reads as $\nu(\alpha_i) \sim \Sigma^{\pm}(\alpha_i)$. We will instantiate $\sim$ as a linear proportionality, i.e., for some numbers a, b, c with $a \neq 0$:

$$\begin{cases} a * \nu(\alpha_1) &= b * \Sigma^{\pm}(\alpha_1) + c \\ a * \nu(\alpha_2) &= b * \Sigma^{\pm}(\alpha_2) + c \\ & \vdots \\ a * \nu(\alpha_n) &= b * \Sigma^{\pm}(\alpha_n) + c \end{cases} \tag{3.4}$$

A more natural formulation may introduce only two numbers d, e and impose $\nu(\alpha_i) = d * \Sigma^{\pm}(\alpha_i) + e$ for $1 \leq i \leq n$. This is tantamount to fixing $a = 1$. The choice for using three numbers was made for reasons of flexibility, but is not fundamental.

Since we think of (3.1) as a positive proportionality, we require that a and b have the same sign, say both positive. Let $\mathbb{A}^f$ be the square $n \times n$ matrix such that for $1 \leq i, j \leq n$,

$$\mathbb{A}^f_{i,j} \coloneqq \sum_{B \subseteq \mathcal{A}} f(B) * \left(I(T, B, \alpha_i, \alpha_j) - I(F, B, \alpha_i, \alpha_j)\right). \tag{3.5}$$

It is easily verified that for every $i \in \{1, \dots, n\}$, $\mathbb{A}^f_{i,i} = 0$.

For $i \in \{1, \dots, n\}$, we introduce a variable x_i for the unknown $\nu(\alpha_i)$. We define $X = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}^{\intercal}$. Then, using (3.3), the system of equations (3.4) can be equivalently written as follows:

$$\left(a \cdot \mathbb{1} - b \cdot \mathbb{A}^f\right) X = \begin{bmatrix} c & \cdots & c \end{bmatrix}^{\intercal}, \tag{3.6}$$

where $\mathbb{1}$ is the square identity matrix. Equation (3.6) has a unique solution if and only if the matrix $\left(a \cdot \mathbb{1} - b \cdot \mathbb{A}^f\right)$ (which does not depend on c) is non-singular (i.e., is invertible). In that case, the solution ABox assessment is given by:

$$X = \left(a \cdot \mathbb{1} - b \cdot \mathbb{A}^f\right)^{-1} \begin{bmatrix} c & c & \cdots & c \end{bmatrix}^{\intercal} \tag{3.7}$$

We should pick $c \neq 0$ to avoid the all-zero solution. Indeed, if $\left(a \cdot \mathbb{1} - b \cdot \mathbb{A}^f\right)$ is non-singular and $c = 0$, then the solution to (3.7) yields the ABox assessment ν such that $\nu(\alpha_1) = \nu(\alpha_2) = \cdots = \nu(\alpha_n) = 0$, which obviously is of no interest for ranking ABox assertions.

Then, it is easily verified that for fixed values of a and b, different choices for c lead to rank-equivalent solutions. Therefore, we can choose $c = 1$ without loss of generality, in which case in the solution, x_i is the sum of all elements in the ith row of $\left(a \cdot \mathbb{1} - b \cdot \mathbb{A}^f\right)^{-1}$. Two significant questions remain:

1. For which values of a and b is the square matrix $\left(a \cdot \mathbb{1} - b \cdot \mathbb{A}^f\right)$ non-singular, such that (3.7) gives us a unique ABox assessment? Obviously, $\left(a \cdot \mathbb{1} - b \cdot \mathbb{A}^f\right)$ is invertible if and only if $\left(\mathbb{1} - \frac{b}{a} \cdot \mathbb{A}^f\right)$ is invertible. Therefore, only the ratio between a and b matters in our subsequent analysis.

2. Are ABox assessments obtained for different values of a, b rank-equivalent?

We conclude this section by providing a graph-theoretical view on the above questions.

Graph-theoretical view From a graph-theoretical viewpoint, $\mathbb{A}^f$ can be interpreted as an edge-weighted directed graph whose vertices are the assertions of the ABox. This view is used in Figures 3.1 and 3.2. An edge from α_j to α_i ($i \neq j$) has weight $\mathbb{A}^f_{i,j}$. The task is then to assign a real number to each vertex such that the resulting graph satisfies the sort of equilibrium expressed by (3.4). The ratio between a and b determines how strongly a vertex is impacted (supported or refuted) by its incoming edges. The question arising is whether an equilibrium (3.4) can be attained for some values of a, b. It is equally important to examine the robustness of such an equilibrium (if it exists) with respect to rank-equivalence. Ideally, different ABox assessments obtained by small parameter changes should be close modulo rank-equivalence.

3.6. Properties of the Assessment

Intuitively, one expects that assessments be invariant under a reordering of the input assertions in the ABox. Also, assessments should not change if we add a new assertion that does not interact with any existing assertion. In this section, we show that our computation indeed satisfies these expected properties.

We first show that the the assessment resulting from our framework is independent of the order supposed on assertions present in the ABox. At some point in the theoretical development, we suppose $\mathcal{A}$ equal to $\{\alpha_1, \ldots, \alpha_n\}$ and base the matrix coordinates on this order. We now show that the computed assessments are independent of such order.

Proposition 3.2. *Let $\mathcal{T}$ be a TBox in some DL. Let $\mathcal{A} = \{\alpha_1, \ldots, \alpha_n\}$ be an ABox in the same vocabulary as $\mathcal{T}$. Let f be a credibility function for $\mathcal{A}$. Let $\mathbb{A}^f$ be the conflict matrix relative to $\langle \mathcal{T}, \mathcal{A} \rangle$ and f.*
Let $\rho : \{1, \ldots, n\} \to \{1, \ldots, n\}$ be a permutation and let $\mathcal{A}' = \{\beta_1, \ldots \beta_n\}$ with $\beta_i = \alpha_{\rho(i)}$ for all i, that is, $\mathcal{A}'$ is the be ABox obtained by changing the order of elements in $\mathcal{A}$ following ρ.
If ${\mathbb{A}'}^f$ denotes the conflict matrix relative to $\langle \mathcal{T}, \mathcal{A}' \rangle$ and f, then:

$$\forall 1 \leq i, j \leq n, \ {\mathbb{A}'}^f_{i,j} = \mathbb{A}^f_{\rho(i),\rho(j)}.$$

In particular, for every triple (a, b, c) of real numbers, $\langle \mathcal{T}, \mathcal{A} \rangle$ has a unique assessment ν with respect to (a, b, c) if and only if $\langle \mathcal{T}, \mathcal{A}' \rangle$ has a unique assessment ν' with respect to (a, b, c). Moreover, for such unique assessments ν and ν', it holds that for all i, we have that $\nu'(\alpha_i) = \nu(\alpha_{\rho(i)})$.

As we have seen, assertions can have both refuters and supporters. Intuitively, we expect that assertions without refuters tend to be more credible, while assertions without supporters may be less credible. In the same way, assertions that do not take part in any interaction should not influence the assessment and should not be influenced by the assessment. All these expected properties turn out to be correct. We first define what it means for an assertion to be unsupported, unrefuted, or independent.

Definition 3.3 (Unsupported, Unrefuted, Independent).
Let $\mathcal{T}$ be a TBox in some DL and let $\mathcal{A}$ be an ABox in the same vocabulary that $\mathcal{T}$. Let I be the indicator function defined in Eq. (3.2). Let α be an assertion in $\mathcal{A}$, we say that α is:

- *unsupported* if for all $B \subseteq \mathcal{A}$ and for all $\beta \in \mathcal{A}$, $I(T, B, \alpha, \beta) = 0$;
- *unrefuted* if for all $B \subseteq \mathcal{A}$ and for all $\beta \in \mathcal{A}$, $I(F, B, \alpha, \beta) = 0$; and
- *independent* if for all $B \subseteq \mathcal{A}$ and for all $\beta \in \mathcal{A}$, we have that $I(T, B, \alpha, \beta) = I(F, B, \alpha, \beta) = 0$ and that $I(T, B, \beta, \alpha) = I(F, B, \beta, \alpha) = 0$.

$\triangleleft$

Proposition 3.3. *Let $\mathcal{T}$ be a TBox in some DL. Let $\mathcal{A} = \{\alpha_1, \dots, \alpha_n\}$ be an ABox relative to $\mathcal{T}$, and f a credibility function for $\mathcal{A}$.*
Let $\alpha \in \mathcal{A}$ be such that α is independent. Then, the system defined by a triple (a, b, c) of reals induces a unique assessment ν for $\langle \mathcal{T}, \mathcal{A} \rangle$ if and only if the system defined by (a, b, c) induces a unique assessment ν' for $\langle \mathcal{T}, \mathcal{A} \backslash \{\alpha\} \rangle$.
In the case that such unique assessment exists, it holds that $\nu(\alpha) = \frac{c}{a}$ and that for every $\beta \in \mathcal{A} \backslash \{\alpha\}$, we have that $\nu(\beta) = \nu'(\beta)$;

Proposition 3.3 tells us that independent assertions can be omitted in the computations, which may be a significant optimization.

Proposition 3.4. *Let $\mathcal{T}$ be a TBox in some DL. Let $\mathcal{A} = \{\alpha_1, \dots, \alpha_n\}$ be an ABox relative to $\mathcal{T}$. Let f be a credibility function such that for all $B \subseteq \mathcal{A}$ we have that $f(B) \geq 0$. Let (a, b, c) be a triple of real numbers that induces a unique assessment ν for $\langle \mathcal{T}, \mathcal{A} \rangle$ such that $\nu(\alpha) \geq 0$ for all $\alpha \in \mathcal{A}$. Then the following statements hold true:*

- *if $\alpha \in \mathcal{A}$ is unrefuted, then $\nu(\alpha) \geq \frac{c}{a}$;*
- *if $\alpha \in \mathcal{A}$ is unsupported, then $\nu(\alpha) \leq \frac{c}{a}$.*

The hypotheses in Proposition 3.4 may be easily met in practice. Indeed, the common aggregation functions $\mathsf{SUM}, \mathsf{MIN}, \mathsf{MAX}, \mathsf{AVG}$ will output positive aggregated numbers if applied on sets of positive numbers. The second hypothesis that $\nu(\alpha) \geq 0$ for all $\alpha \in \mathcal{A}$ is also mild. Indeed, this property will be automatically satisfied by our method (Theorem 3.7) whenever $c \geq 0$.

3.7. Solving the Instantiated Framework

In Section 3.5, we presented the matrix equation (3.6) whose unique solution, if it exists, yields an ABox assessment. This matrix equation has parameters a, b. A remaining problem is how to pick appropriate values for these parameters. One solution could be to choose values for a, b in an empirical fashion, relying, for instance, on information obtained from some learning experience. However, we will only focus on theoretical aspects in the current chapter. Later, in Chapter 5, we will discuss an implementation of this theory in the rustoner software program; Figures 3.1 and 3.2 were generated using this program. In

Section 3.7.1, we will show a parameterization scheme for a, b that guarantees the existence of a solution for (3.6): pick $a \gg b > 0$. What is probably more important is that when the ratio a/b grows, the ABox assessments obtained for different a, b become all rank-equivalent. Informally, the parameterization scheme converges to a unique ABox assessment modulo rank-equivalence.

3.7.1 Solution Existence

Note that all diagonal elements in $\left(a \cdot \mathbb{1} - b \cdot \mathbb{A}^f\right)$ are equal to a. The invertibility of $\left(a \cdot \mathbb{1} - b \cdot \mathbb{A}^f\right)$ can be guaranteed by (some form of) diagonal dominance, a concept that is easily grasped and has turned out to be useful in applications [115]. An example is the Levy-Desplanques theorem recalled next.

Theorem 3.5 (Levy-Desplanques)**.** *A square $n \times n$ matrix A is invertible if for every $i \in \{1, \ldots, n\}$, $|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}|$.*

From Theorem 3.5, it immediately follows that (3.7) has a unique solution if for all $i \in \{1, \ldots, n\}$, we have $|a| > |b| * \sum_{j=1}^{n} \left|\mathbb{A}^f_{i,j}\right|$. This is illustrated by Example 3.2.

Example 3.2
Assume four assertions $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ such that α_1 and α_2 support one another, while α_3 and α_4 refute one another. Take $b = 1$. If we assume $f(\{\alpha_i\}) = 1$ for $1 \leq i \leq 4$, we obtain:

$$\mathbb{A}^f = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 \end{bmatrix}.$$

Then,

$$a \cdot \mathbb{1} - b \cdot \mathbb{A}^f = \begin{bmatrix} a & -1 & 0 & 0 \\ -1 & a & 0 & 0 \\ 0 & 0 & a & 1 \\ 0 & 0 & 1 & a \end{bmatrix}.$$

It follows from Theorem 3.5 that this matrix is invertible for $a > 1$, giving the following solution:

$$X = \begin{bmatrix} \frac{1}{a-1} \\ \frac{1}{a-1} \\ \frac{1}{a+1} \\ \frac{1}{a+1} \end{bmatrix}.$$

Although distinct values for a lead to different ABox assessments, it is significant to observe that $\frac{1}{a-1} > \frac{1}{a+1}$ $(a > 1)$, and therefore all these ABox assessments ν are rank-equivalent: $\nu(\alpha_1) = \nu(\alpha_2) > \nu(\alpha_3) = \nu(\alpha_4)$. It is intuitively correct that the two assertions that mutually attack one another lose credibility.

◁

The invertibility of $\left(a \cdot \mathbb{1} - b \cdot \mathbb{A}^f\right)$ may also follow from Banach's Lemma, which is recalled next.

Theorem 3.6 (Banach's Lemma)**.** *Let M be a square matrix with the property that $\|M\| < 1$ for some operator norm $\|\cdot\|$. Then the matrix $\mathbb{1} - M$ is invertible and $(\mathbb{1} - M)^{-1} = \mathbb{1} + M + M^2 + M^3 + \cdots$.*

Therefore, if we fix $a = 1$ and pick b sufficiently small (which is analogous to fixing $b = 1$ and picking a sufficiently large), then the matrix $\mathbb{1} - b \cdot \mathbb{A}^f$ is invertible, and $\left(\mathbb{1} - b \cdot \mathbb{A}^f\right)^{-1} = \mathbb{1} + b \cdot \mathbb{A}^f + \cdots$. If we limit ourselves to the two most significant terms in this expansion, we get $X = \left(\mathbb{1} + \mathbb{A}^f\right) \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix}^\intercal$ as an approximate solution for (3.7). The ranking obtained by this solution is intuitively meaningful, because it ranks α_i based on the sum of the elements in the ith row of $\mathbb{A}^f$; in this row, supporters have a positive sign, and refuters a negative sign.

3.7.2 Convergence Towards a Fixed Ranking

So it turns out that (3.6) has a unique solution with $a, b \geq 0$ whenever the ratio a/b is sufficiently large. It is desirable that different solutions are rank-equivalent, as was the case in Example 3.2. We now show that this desirable property indeed holds true beyond some threshold value for a/b.

The notion of rank-equivalence obviously extends to any two vectors of real numbers: two such vectors u, v of the same length n are rank-equivalent if for all $1 \leq i, j \leq n$, $u_i < u_j$ if and only if $v_i < v_j$. The proof of the following theorem is in Appendix D. Since, as argued before, only the ratio between a and b matters in our analysis, we can fix b and let a increase in the following theorem.

Theorem 3.7 (Convergence)**.** *Let M be an $n \times n$ matrix whose diagonal elements are zero. Let $b, c \in \mathbb{R}$. Consider the following equation with real-number parameter a:*

$$(a \cdot \mathbb{1} - b \cdot M)\, X = \begin{bmatrix} c & \cdots & c \end{bmatrix}^{\intercal}. \tag{3.8}$$

Then there exists $a^* \in \mathbb{R}$ such that

- *for every $a \geq a^*$, the equation* (3.8) *has a unique solution; and*
- *for all $a_1, a_2 \geq a^*$, if X_1 and X_2 are solutions to* (3.8) *for parameters a_1 and a_2 respectively, then X_1 and X_2 are rank-equivalent.*

Moreover, such a bound a^* can be computed in polynomial time in the size of n.

Informally, Theorem 3.7 tells us that for all choices of b and c, there is a threshold value a^* such that all choices of a satisfying $a \geq a^*$ lead to the same ABox assessment modulo rank-equivalence.

3.7.3 Computational Complexity

We discuss the computational complexity of solving (3.6). The main difficulty is the computation of the non-diagonal elements in $\mathbb{A}^f$, which are defined by (3.5). Given α_i and β_j, this requires finding every $B \subseteq \mathcal{A}$ such that $\beta_j \in B$ and B is a supporter or refuter of α_i. In general, the number of supporters or refuters can be exponential in the size of $\mathcal{A}$, because an ABox of size $2n$ can have $\binom{2n}{n}$ supporters or refuters not comparable by set inclusion. However, in practice, a fixed upper bound on the size of supporters or refuters may be implied by the TBox, in a way captured by the following definition.

Definition 3.4 (Conflict-bounded TBox).
We say that a TBox $\mathcal{T}$ is *conflict-bounded* if there exists a polynomial-time computable positive integer m such that for every knowledge base $\langle \mathcal{T}, \mathcal{A} \rangle$, if

$\langle \mathcal{T}, \mathcal{A} \rangle$ is inconsistent, then there exists $B \subseteq \mathcal{A}$ with $|B| \leq m$ such that $\langle \mathcal{T}, B \rangle$ is already inconsistent.

◁

Conflict-boundedness arises in practice. For example, for every *DL-Lite* TBox, every supporter or refuter has cardinality ≤ 1 because each conflict in *DL-Lite* involves at most two assertions. Conflict-boundedness is also guaranteed in $\mathcal{EL}_{\perp nr}$ ($\mathcal{EL}_{\perp}$ with non-recursive empty concepts) defined in [99].

Theorem 3.8. *Let $\mathcal{T}$ be a conflict-bounded TBox such that ABox consistency with respect to $\mathcal{T}$ can be checked in polynomial time. Then, for every ABox $\mathcal{A}$, the matrix $\mathbb{A}^f$ can be computed in polynomial time (in the size of $\mathcal{A}$) if f is polynomial-time computable.*

Proof. Let $n := |\mathcal{A}|$. Recall that $\mathbb{A}^f$ is an $n \times n$ matrix given by (3.5). Since $\mathcal{T}$ is conflict-bounded, say with bound m, for every $B \subseteq \mathcal{A}$, if $|B| > m$, then $I(T, B, \alpha_i, \alpha_j) = I(F, B, \alpha_i, \alpha_j) = 0$. Therefore, the sum in (3.5) must only consider subsets $B \subseteq \mathcal{A}$ with $|B| \leq m$, which are polynomially many and polynomial-time computable. Furthermore, since ABox consistency checking is in polynomial time by the hypothesis of the theorem, it can be tested in polynomial time whether any such B containing α_j is a supporter or a refuter of α_i (or neither of both). The computation of $f(B)$ is also in polynomial time by the hypothesis of the theorem. It is now obvious that any entry of $\mathbb{A}^f$ can be computed in polynomial time.

□

Once the conflict matrix $\mathbb{A}^f$ has been created, finding an assessment relative to a triple (a, b, c) boils down to solving a linear system, which can be done in time $\mathcal{O}(n^3)$ where n is the dimension of the ABox. Finding a stabilized assessment is more time consuming, but still in polynomial time by Theorem 3.7.

An inspection of the preceding proof reveals that Theorem 3.8 remains valid if, throughout its statement, we replace polynomial time by some higher complexity upper bound (e.g., polynomial space or exponential time).

3.8. Credibility and Aggregated Credibility

So far, throughout this chapter, we have assumed an aggregate credibilty function $f : 2^{\mathcal{A}} \to \mathbb{R}$. In the theoretical development, we treated f as a basic construct, and deliberately abstracted away from aggregate operators. In this section, we will explain how f can be defined in terms of aggregation. We first define the notion of aggregate operator.

Definition 3.5 (Aggregate Operator).
An *aggregate operator* is a collection $\mathcal{G} = \{g_0, g_1, g_2, \dots\}$ of functions, where each g_n, $0 < n$, takes an n-element multiset (bag) of real numbers, and returns a number in $\mathbb{R}$. Furthermore, g_0 is constant.

◁

For example, the aggregate operator SUM will be represented as $\mathcal{G}_{\mathsf{SUM}} = \{g_0, g_1, g_2, \dots\}$, where $g_0 = 0$, and

$$g_n\left(\{\!\{r_1, \dots, r_n\}\!\}\right) = r_1 + \dots + r_n.$$

Note incidentally that aggregate operators in Definition 3.5 are only defined on finite multisets, which is sufficient for our framework.

Starting from a user-defined credibility function $w : \mathcal{A} \to \mathbb{R}$, such an aggregate operator naturally defines an aggregate credibility function $f : 2^{\mathcal{A}} \to \mathbb{R}$, as follows:

> for every finite set $S = \{\alpha_1, \alpha_2, \dots, \alpha_n\}$, $S \subseteq \mathcal{A}$, with $n \geq 0$, we define
> $$f(S) \coloneqq g_n(\{\!\{w(\alpha_1), w(\alpha_2), \dots, w(\alpha_n)\}\!\}).$$
> In particular, $f(\emptyset) \coloneqq g_0(\emptyset)$.

The function f defined in this way will be denoted by $\mathcal{G}_{\triangleright w}$.

Example 3.3
Let $\mathcal{A} = \{\alpha_1, \alpha_2, \alpha_3, \dots\}$. Let $w(\alpha_1) = 4$ and $w(\alpha_2) = w(\alpha_3) = 5$. For the aggregate operator SUM, we obtain $f(\{\alpha_1, \alpha_2, \alpha_3\}) = 4 + 5 + 5 = 14$.

◁

Obviously, different aggregate operators will lead to different credibility values for supporters and refuters, as illustrated next.

Example 3.4

Let $\mathcal{T} = \{A \sqcap B \sqsubseteq \neg C, D \sqcap E \sqcap F \sqsubseteq C,\}$ and let $\mathcal{A} = \{a : A,\ a : B,\ a : C,\ a : D,\ a : E,\ a : F\}$. Assume that an expert's credibility function w is as follows:

- $w(a : A) = w(a : B) = 2$; and
- $w(a : C) = w(a : D) = w(a : E) = w(a : F) = 1$.

Then, $S = \{a : D, a : E, a : F\}$ is the only supporter of $a : C$, and $R = \{a : A, a : B\}$ is the only refuter of $a : C$. If the aggregate operator SUM is used, then the aggregated credibilities of R and S are, respectively, 4 and 3. If the aggregate operator COUNT is used, then the aggregated credibilities of R and S are, respectively, 2 and 3.

$\lhd$

In the next chapter, we will study another aspect of aggregate operators in the context of knowledge base repairing. It will then become more clear that there is no single "best" aggregation function. Different users may prefer different aggregation functions: MIN guarantees a minimal quality for each assertion in the knowledge base, SUM is an indicator of the cumulative quality, while AVG can give the user an idea of the quality of each assertion on average.

3.9. Conclusion

In this work, we have proposed a novel framework to rank the assertions of an inconsistent ABox in terms of their degree of truthfulness. This ranking takes into account an expert's credibility function as well as the logical axioms of the TBox. The theoretical framework can work with any Description Logic, but to gain computational tractability, restrictions have to be imposed.

The framework is implementable using existing libraries for matrix operations and description logics. A prototype of the ranking has been implemented in the rustoner program [88].

A next step is to use our framework for repairing database and knowledge base systems. Our ranking of ABox assertions can be extended in several ways to a ranking of repairs, using some notion of Pareto optimality. For example, if two facts α and β contradict one another and α is ranked higher than β, then a repair containing α is preferred over a repair containing β (all other facts being equal). This brings us in the realm of prioritized database repairing [118], which we see as a promising way to enrich existing DL repair semantics (like IAR and AR). By narrowing the search space of possible repairs, we may also hope to gain efficiency compared to existing repair semantics.

Theorem 3.7 states that in the proposed framework, we can single out a unique ranking of ABox assertions, which is obtained by choosing a sufficiently large value for the parameter a. However, smaller values for a may yield other assessments that need not to be rank-equivalent to the assessments found in Theorem 3.7. This raises some questions that merit to be studied in both theory and practice: Can the rankings obtained for two different values of a be highly uncorrelated (relative to some rank correlation coefficient)? If so, can it be argued that the "asymptotic" ranking found in Theorem 3.7 is intrinsically more desirable than rankings obtained for smaller values of a? Is there an informal reason to believe that rankings obtained with higher values for a will be more useful in practice?

In the next chapter, we will develop a quantified approach to database repairing. Nevertheless, the ABox ranking developed in the current chapter allows for an alternative approach along the lines of the framework introduced in [105] and further investigated in [47, 66, 77]. In this framework, a preference relation on database repairs is deduced from a preference relation on the set of database facts. Along the same lines, our ranking of ABox assertions establishes a preference relation which can be lifted to repairs.

For example, assume that starting from an inconsistent knowledge base $\langle \mathcal{T}, \mathcal{A} \rangle$, we find two subsets $\mathcal{A}_1, \mathcal{A}_2 \subseteq \mathcal{A}$ such that both $\langle \mathcal{T}, \mathcal{A}_1 \rangle$ and $\langle \mathcal{T}, \mathcal{A}_2 \rangle$ are consistent. Under the following condition, we can say that $\mathcal{A}_1$ is preferred over $\mathcal{A}_2$: for every $B_2 \in \mathcal{A}_2 \setminus \mathcal{A}_1$, there exists $B_1 \in \mathcal{A}_1 \setminus \mathcal{A}_2$ such that B_1 is ranked before B_2. Informally, $\mathcal{A}_1$ is preferred over $\mathcal{A}_2$ if $\mathcal{A}_1$ can be obtained from $\mathcal{A}_2$ by exchanging assertions with assertions of higher quality. A *globally optimal repair* can then be defined as a consistent ABox that is maximal with respect to the above preference relation.

CHAPTER 4

Weighted Repairs

Remark 4.0
The content of this chapter will be presented at FQAS 2021 [91].

◁

4.1. Motivation

In today's era of "big data," database management systems have to cope more and more with dirty information that is inconsistent with respect to some integrity constraints. Such integrity constraints are commonly expressed in decidable fragments of some logic, for example, as dependencies [6] or ontologies in some Description Logic [16]. The term *repair* is commonly used to refer to a consistent database that is obtained from the inconsistent database by some minimal modifications. This notion was introduced twenty years ago in a seminal paper by Arenas et al. [12], and has been an active area of research ever since. In particular, the field of *Consistent Query Answering (CQA)* studies the question of how to answer database queries if multiple repairs are possible. Two surveys of this research are [18, 118].

This chapter's aim is to contribute to the research in *preferred repair semantics*, whose goal is to capture more of the meaning of the data into the repairing process. To this end, we introduce and study *weighted repairs*. We will assume that database tuples are associated with numerical weights such that tuples with higher weights are preferred over tuples with lower weights. Then, among all possible repairs, weighted repairs are those with a maximum aggregated value, according to some aggregation function. We will study the relationship between the complexity of computing weighted repairs and certain properties of the aggregation function used.

The remainder of this section is an informal guided tour that introduces and motivates our research questions by means of a simple example. We start with a graph-theoretical view on database repairing.

A Graph-Theoretical Perspective on Database Repairing

Consider the following table *TEACHES*, in which a fact *TEACHES*(p, c, s) means that professor p teaches the course c during semester s.

TEACHES	*Prof*	*C#*	*Sem*	
	Jeff	A	fall	(f_1)
	Jeff	B	fall	(f_2)
	Ed	C	spring	(s_1)
	Rob	C	spring	(s_2)
	Rob	D	spring	(s_3)

The integrity constraints are as follows:

$$\Sigma = \{\{\mathit{Prof}, \mathit{Sem}\} \to \{C\#\}, \{C\#\} \to \{\mathit{Prof}\}\},$$

The first functional dependency expresses that no professor teaches more than one course in a given semester. The second functional dependency expresses that no course is taught by more than one professor. The relation *TEACHES* violates these integrity constraints; its conflict graph is shown in Fig. 4.1. The vertices of the conflict graph are the facts in the relation *TEACHES*; two vertices are adjacent if together they violate some functional dependency.

Given a database instance, it is common to define a *subset repair* as an inclusion-maximal subinstance that satisfies all integrity constraints. In terms of the conflict graph, every subset repair is an *inclusion-maximal independent set (IMIS)*, and vice versa. Recall that in graph theory, a set of vertices is *independent* if no two vertices of it are adjacent. It can be verified that

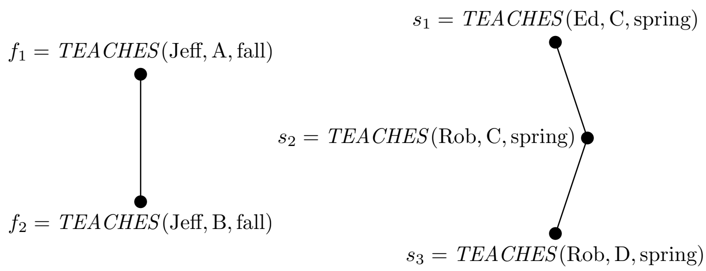

Figure 4.1: Conflict graph for the running example.

the graph of Fig. 4.1 has four IMISs: every IMIS includes either $\{f_1\}$ or $\{f_2\}$, and includes either $\{s_1, s_3\}$ or $\{s_2\}$. The term *cardinality-repair* refers to independent sets of maximum cardinality. In our running example, the cardinality-repairs are $\{f_1, s_1, s_3\}$ and $\{f_2, s_1, s_3\}$.

Maximum-Weight Independent Set (MWIS)

As in [81], we will assume from here on that every fact is associated with a nonnegative weight, where larger weights are better. In practice, such weights may occur in data integration, where facts coming from more authoritative data sources are tagged with higher weights. For example, in the next relational table, among the first two facts—which are conflicting—the second fact has a higher weight and is therefore considered better.

TEACHES	*Prof*	*C#*	*Sem*	*w*
	Jeff	A	fall	1
	Jeff	B	fall	2
	Ed	C	spring	1
	Rob	C	spring	2
	Rob	D	spring	1

It is now natural to take these weights into account, and define repairs as maximum-weight independent sets (MWIS) of the conflict graph. In graph theory, an MWIS is an independent set that maximizes the sum of the weights of its vertices. In our example, there are two MWISs, both having a total summed weight of 4:

T_1	*Prof*	*C#*	*Sem*	*w*
	Jeff	B	fall	2
	Rob	C	spring	2

and

T_2	*Prof*	*C#*	*Sem*	*w*
	Jeff	B	fall	2
	Ed	C	spring	1
	Rob	D	spring	1

.

Aggregation Functions Other than SUM

The aggregation function SUM is cooked into the definition of MWIS: among all independent sets, an MWIS is one that maximizes the *summed* weight. From a conceptual perspective, it may be natural to use aggregation functions other than SUM. For example, among the two repairs T_1 and T_2 shown above, we may prefer T_1 because it maximizes the *average* weight. Indeed, the average weights for T_1 and T_2 are, respectively, $\frac{2+2}{2}$ and $\frac{2+1+1}{3}$. Alternatively, we may prefer T_1 because it maximizes the *minimum* weight. Therefore, to capture these alternatives, we will allow other functions than SUM in this chapter, including AVG and MIN.

Computing Repairs

In data cleaning and database repairing, we are often interested in finding one or more repairs for a given database instance. Now that we have discussed weights and different aggregation functions, this boils down to the following task: given a database instance with weighted facts, return an inclusion-maximal consistent subinstance that maximizes the aggregated weight according to some fixed aggregation function. Alternatively, in graph-theoretical terms: given a vertex-weighted graph, return an inclusion-maximal independent set that maximizes the aggregated weight according to some fixed aggregation function. Since for aggregation functions other than SUM, maximality with respect to set inclusion and maximality with respect to aggregated weight may not go hand in hand, it should be made precise which criterion prevails:

- among all inclusion-maximal independent sets, return one that maximizes the aggregated weight; or
- among all independent sets that maximize the aggregated weight, return one that is inclusion-maximal.

To illustrate the difference, let $G = (V, E)$ with $V = \{a, b\}$ and $E = \emptyset$. Let $w(a) = 1$ and $w(b) = 2$, and let MIN be the aggregation function. The first task would return $\{a, b\}$, while the second task would return $\{b\}$. In this chapter, we will study the latter task, because from a quality perspective, the criterion of maximal aggregated weight seems more important than inclusion-maximality.

It is known that under standard complexity assumptions (in particular, $\mathsf{P} \neq \mathsf{NP}$), there is no polynomial-time algorithm that returns an MWIS for a given vertex-weighted graph [51]. Therefore, when the aggregation function SUM is used, it is intractable to return a repair with a maximum summed weight. In this chapter, we will ask whether this task can become tractable for other aggregation functions of practical interest. Contributions of this chapter can be summarized as follows.

- We introduce $\mathcal{G}$-repairs, generalizing existing repair notions.
- By taking a conflict hypergraph perspective on database integrity, we show that $\mathcal{G}$-repairs are a generalization of maximum-weight independent sets.
- We adapt classical decision problems to our setting, and study their computational complexity. While these problems are intractable in general, we identify classes of aggregation functions that allow for polynomial-time algorithms.

The rest of this chapter is organized as follows. Section 4.2 discusses related work. Section 4.3 introduces aggregation functions and defines the (conflict) hypergraph perspective for studying inconsistent databases. Section 4.4 defines the notion of $\mathcal{G}$-repair and the computational problems we are interested in. Section 4.5 shows computational complexity results for these problems, culminating in our main tractability theorem, Theorem 4.6. Section 4.6 shows that tractability is lost under a slight relaxation of the hypotheses of that theorem. Finally, Section 4.7 concludes the chapter.

4.2. Related Work

In their seminal paper [12], Arenas et al. define repairs of an inconsistent database as those consistent databases that can be obtained by inserting and deleting minimal (with respect to set inclusion) sets of tuples. Since then, many variants of this earliest repair notion have been introduced, several of which are discussed in [18,36,118]. For any fixed repair notion, *repair checking* is the following problem: given an inconsistent database and a candidate repair, is the candidate a true repair (i.e., does it satisfy all constraints imposed by the repair notion under consideration)? Afrati and Kolaitis [7] made important contributions to our understanding of the complexity of repair checking. For databases containing numerical attributes, repairs have also been defined as solutions to numerical constraint problems [19, 26, 48].

A notable work is [78], where the authors study also weigthed databases and where the "Most Probable Database" [55] problem relative to database cleaning and repairing is settled for functional dependencies.

The notion of *conflict hypergraph* was introduced in [37], and later extended in [106]. The relationship between repairs and inclusion-maximal independent sets was observed in [37, Proposition 3.1]. If database tuples are associated with nonnegative weights, then it is natural to generalize this relationship by viewing repairs as maximum-weight independent sets (MWIS).

As we will explain shortly, our setting can be naturally expressed in graph-theoretical terms, using weighted variants of the maximal independent set problem. Many works deal with finding, approximating, or counting maximum weighted independent sets (e.g., [42, 65, 109, 122], in simple graphs as well as hypergraphs [8, 57]. In our approach, however, we do not primarily focus on the maximum *summed* weight, but also allow for aggregation functions other than SUM. The problems we study are specifically motivated by database applications in which several other aggregation functions are sensible.

We will show that some problems that are NP-hard in general, become tractable for aggregation functions that have some desirable properties. Inspired by database theory, weight-based approaches to inconsistency have also been studied for knowledge bases in Description Logics [22, 46].

4.3. Preliminaries

Aggregation Functions over Weighted Sets

Informally, aggregation functions take as input a set of elements, each associated with a weight, and return an aggregated weight for the entire set. Examples are SUM and MIN. In this work, all weights will be nonnegative rational numbers, which we interpret as quality scores where higher values are better. These notions are formalized next.

Definition 4.1 (Weighted set).
A *weighted set* is a pair (I, w) where I is a finite set and w is a total mapping from I to $\mathbb{Q}^+$ (i.e., the set of nonnegative rational numbers). We will often assume that the *weight function* w is implicitly understood. That is, whenever we say that I is a weighted set, we mean that (I, w) is a weighted set for a mapping w that is implicitly understood.

Two weighted sets (I_1, w_1) and (I_2, w_2) are *isomorphic* if there is a bijection $\pi : I_1 \to I_2$ such that for every $x \in I_1$, $w_1(x) = w_2(\pi(x))$. Informally, two weighted sets are isomorphic if the attained numeric values as well as their multiplicities coincide.

$\triangleleft$

A standard definition of aggregate operator is Definition 3.5 of Chapter 3. We now define aggregation functions in a slightly different manner that better suits the theoretical treatment in this chapter. Instead of aggregating over *bags* of real numbers, we will aggregate over *sets* in which elements are associated with (not necessarily distinct) weights.

Definition 4.2 (Aggregation function).
An *aggregation function* $\mathcal{G}$ is a function that maps every weighted set (I, w) to a nonnegative rational number, denoted $\mathcal{G}_{\triangleright w}(I)$, such that:

- $\mathcal{G}$ is closed under isomorphism, meaning that any two weighted sets that are isomorphic are mapped to the same value; and
- the empty weighted set is mapped to 0.

We write $\mathbf{AGG}^{\mathsf{poly}}$ for the class of aggregation functions that are computable in polynomial time in $|I|$. Some well-known members of this class are denoted COUNT, SUM, MAX, MIN, AVG and PRODUCT, with their expected, natural semantics (not repeated here).

◁

By measuring the complexity of an aggregation function $\mathcal{G}$ in terms of $|I|$, we do not take into account the size of the numeric values in the image of the mapping w. This complexity is appropriate for the applications we have in mind. The requirement that an aggregation function be closed under isomorphism is tantamount to saying that for a weighted set I, the value $\mathcal{G}_{\triangleright w}(I)$ depends on, and only on, the multiset $\{\!\{w(x) \mid x \in I\}\!\}$. While it may be more common to define aggregation functions on multisets of numbers (see, e.g., [76, p. 159]), our Definition 4.2 is appropriate for the purposes we have in mind. Indeed, we will only apply aggregation on weighted sets formed by vertices of a vertex-weighted graph.

Conflict Hypergraphs

Conflict hypergraphs [37, 38] generalize the conflict graphs introduced previously in Section 4.1. To detect violations of functional dependencies, it suffices to regard two tuples at a time. However, more involved constraints may consider three or more tuples at a time. For this reason, conflict graphs are generalized to conflict hypergraphs. Informally, a conflict hypergraph is a hypergraph whose vertices are the database facts; hyperedges are formed by grouping facts that together violate some integrity constraint.

Formally, a fact is an expression $R(c_1, \dots, c_n)$ where R is a relation name of arity n, and each c_i is a constant. A database is a finite set of facts. Let $\mathbf{db}$ be a database instance, and $\mathcal{C}$ be a set of integrity constraints. The *(conflict) hypergraph* is defined as an hypergraph $H = (V, E)$ whose vertices are the facts of $\mathbf{db}$; there is an hyperedge $e = \{R_1(\vec{c}_1), \dots, R_k(\vec{c}_k)\}$ if (and only if) the following properties hold:

1. the facts $R_1(\vec{c}_1), \dots, R_k(\vec{c}_k)$ taken together violate one or more integrity constraints of $\mathcal{C}$; and
2. every strict subset of e satisfies $\mathcal{C}$.

In other words, the hyperedges of H are the inclusion-minimal inconsistent subsets of **db**. Recall from graph theory that an *independent set* of a hypergraph $H = (V, E)$ is a set $I \subseteq V$ such that I includes no hyperedge of E. Then, by construction, the following expressions are obviously equivalent for every database instance **db** and set $\mathcal{C}$ of integrity constraints:

- I is an independent set of the (conflict) hypergraph; and
- I is consistent, i.e., I satisfies $\mathcal{C}$.

It is this equivalence between independent sets and database consistency that motivates the hypergraph perspective on database repairing. For most database integrity constraints, minimal (w.r.t. $\subseteq$) inconsistent sets are bounded in size. For example, for functional dependencies or primary keys, this bound is 2. This will be mimicked in the hypergraph perspective by assuming a bound b (some positive integer) on the size of the hyperedges.

Finally, we will consider vertex-weighted hypergraphs, i.e., the vertex set will be a weighted set, as defined by Definition 4.1.

Definition 4.3 (Weighted Hypergraph).
A hypergraph is called *weighted* if its vertex set is a weighted set. Technically, such a hypergraph is a nested pair $((V, w), E)$ with (V, w) a weighted set of vertices, and E a set of hyperedges. However, as announced in Definition 4.1, we often omit the explicit mention of the weight function w. For simplicity, we will assume that no hyperedge is a singleton. For every integer $b \geq 2$, we define $\mathbf{WH}[b]$ as the set of weighted hypergraphs containing no hyperedge of cardinality strictly greater than b.

$\triangleleft$

To conclude this section, we argue that for most common database integrity constraints, the hypergraph perspective is appropriate for our computational complexity studies, even if constraints are given as expressions in relational calculus. The reason is that P (i.e., polynomial time) is the smallest complexity class considered in our complexity analysis, while for most database constraints, conflict hypergraphs can be obtained by a query in relational calculus, which is strictly contained in P. For example, for a functional dependency $R : X \to Y$, the edges of the conflict hypergraph are all pairs of tuples in R that agree on all attributes of X but disagree on some attribute in Y.

4.4. Repair Checking and Related Problems

A repair of an inconsistent databases **db** is often defined as a maximal consistent subinstance of **db**, where maximality can be with respect to set inclusion or cardinality, yielding subset- and cardinality-repairs, respectively. These notions carry over to the hypergraph perspective defined in Section 4.3. For any aggregation function $\mathcal{G}$ and weighted hypergraph, we now define $\mathcal{G}$-repairs as a natural generalization of existing repair notions. Significantly, from a graph-theoretical viewpoint, $\mathcal{G}$-repairs generalize *maximum-weight independent sets*, which are independent sets of vertices whose weights sum to the maximum possible value. In $\mathcal{G}$-repairs, other functions than SUM can be used.

Definition 4.4 ($\mathcal{G}$-repair).
Let $\mathcal{G}$ be an aggregation function, and $H = ((V, w), E)$ a weighted hypergraph. A $\mathcal{G}$-repair of H is a subset $I \subseteq V$ with the following three properties:

Independence: I is an independent set of H;

Maximality: for every other independent set $J \subseteq V$, it holds that $\mathcal{G}_{\triangleright w}(I) \geq \mathcal{G}_{\triangleright w}(J)$; and

Saturation: for every other independent set $J \subseteq V$, if $\mathcal{G}_{\triangleright w}(I) = \mathcal{G}_{\triangleright w}(J)$ and $I \subseteq J$, then $I = J$.

$\lhd$

Informally, among all independent sets that maximize $\mathcal{G}_{\triangleright w}$, a weighted repair is one that is inclusion-maximal. Subset repairs and cardinality-repairs are special cases of $\mathcal{G}$-repairs. Indeed, if we let $\mathcal{G} = \mathsf{SUM}$ and $w(v) = 1$ for every vertex v, then $\mathcal{G}$-repairs coincide with cardinality-repairs. If we let $\mathcal{G} = \mathsf{MIN}$ and $w(v) = 1$ for every vertex v, then $\mathcal{G}$-repairs coincide with subset repairs.

We now relax $\mathcal{G}$-repairs by replacing the *Maximality* requirement in Definition 4.4 by a lower bound on the aggregated value. This corresponds to real-life situations where we may already be happy with a guaranteed lower bound.

Definition 4.5 (q-suitable vertex set).
This definition is relative to some fixed aggregation function $\mathcal{G}$. Let $H =$

$((V,w),E)$ be a weighted hypergraph, and $q \in \mathbb{Q}^+$. A set $I \subseteq V$ is said to be a *q-suitable set* of H if the following three properties hold true:

Independence: I is an independent set of H;

Suitability: $\mathcal{G}_{\triangleright w}(I) \geq q$; and

Saturation: for every other independent set $J \subseteq V$ such that $I \subseteq J$, if $\mathcal{G}_{\triangleright w}(I) \leq \mathcal{G}_{\triangleright w}(J)$, then $I = J$.

$\triangleleft$

Informally, an independent set I is q-suitable if its aggregated value is at least q and every strict extension of I is either not independent or has a strictly smaller aggregated value. The decision problems of our interest generalize repair checking, which is central in consistent query answering [118].

Definition 4.6
The following problems are relative to an aggregation function $\mathcal{G}$ in $\mathbf{AGG}^{\mathsf{poly}}$ and a positive integer b.

PROBLEM $\mathsf{REPAIR\text{-}CHECKING}_{(\mathcal{G},b)}$

Input: A weighted hypergraph H in $\mathbf{WH}[b]$; a set I of vertices.

Question: Is I a $\mathcal{G}$-repair of H?

PROBLEM $\mathsf{REPAIR\text{-}EXISTENCE}_{(\mathcal{G},b)}$

Input: A weighted hypergraph H in $\mathbf{WH}[b]$; a rational number q.

Question: Does H have a $\mathcal{G}$-repair I such that $\mathcal{G}_{\triangleright w}(I) \geq q$?

PROBLEM $\mathsf{SUITABILITY\text{-}CHECKING}_{(\mathcal{G},b)}$

Input: A weighted hypergraph H in $\mathbf{WH}[b]$; a set I of vertices; a rational number q.

Question: Is I a q-suitable set of H (with respect to $\mathcal{G}$)?

$\triangleleft$

These problems obviously have relationships among them. For example, if the answer to $\mathsf{SUITABILITY\text{-}CHECKING}_{(\mathcal{G},b)}$ on input H, I, q is "yes," then the answer to $\mathsf{REPAIR\text{-}EXISTENCE}_{(\mathcal{G},b)}$ on input H, q is also "yes." Also, for a weighted hypergraph H, if $q \coloneqq \max\{\mathcal{G}_{\rhd w}(J) \mid J \text{ is an independent set of } H\}$, then every $\mathcal{G}$-repair is a q-suitable set, and vice versa. We now give some computational complexity results. The proof of the following result is straightforward.

Theorem 4.1 (Complexity upper bounds)**.** *For every $\mathcal{G} \in \mathbf{AGG}^{\mathsf{poly}}$ and $b \geq 2$,* $\mathsf{REPAIR\text{-}CHECKING}_{(\mathcal{G},b)}$*, and* $\mathsf{SUITABILITY\text{-}CHECKING}_{(\mathcal{G},b)}$ *are in* coNP*, and* $\mathsf{REPAIR\text{-}EXISTENCE}_{(\mathcal{G},b)}$ *is in* NP*.*

The following result means that our problems are already intractable under the simplest parametrization.

Theorem 4.2 (Complexity lower bounds)**.** $\mathsf{REPAIR\text{-}CHECKING}_{(\mathsf{COUNT},2)}$ *is* coNP*-hard and* $\mathsf{REPAIR\text{-}EXISTENCE}_{(\mathsf{COUNT},2)}$ *is* NP*-hard.*

On the other hand, $\mathsf{SUITABILITY\text{-}CHECKING}_{(\mathsf{COUNT},2)}$ is tractable (i.e., in P). Indeed, tractability holds for a larger class of aggregation functions that contains COUNT and is defined next.

Definition 4.7 ($\subseteq$-monotone).
An aggregation function is called $\subseteq$*-monotone* if for every weighted set (I, w), for all $J_1, J_2 \subseteq I$ such that $J_1 \subseteq J_2$, it holds that $\mathcal{G}_{\rhd w}(J_1) \leq \mathcal{G}_{\rhd w}(J_2)$.[1]

$\triangleleft$

It is easily verified that COUNT and MAX are $\subseteq$-monotone. SUM is also $\subseteq$-monotone, because we do not consider negative numbers. On the other hand, MIN and AVG are not $\subseteq$-monotone. We give the following claim without proof, because we will shortly prove a stronger result.

Claim 4.3 (Complexity upper bound)**.** *For every $\mathcal{G} \in \mathbf{AGG}^{\mathsf{poly}}$ and $b \geq 2$, if $\mathcal{G}$ is $\subseteq$-monotone, then* $\mathsf{SUITABILITY\text{-}CHECKING}_{(\mathcal{G},b)}$ *is in* P*.*

[1] In the notation $\mathcal{G}_{\rhd w}(J_1)$, the weight function is understood to be the restriction of w to J_1.

4.5. Main Tractability Theorem

Theorem 4.2 shows that $\mathsf{REPAIR\text{-}CHECKING}_{(\mathcal{G},b)}$ becomes already intractable for simple aggregation functions and conflict hypergraphs. The aim of the current section is to better understand the reason for this intractability, and to identify aggregation functions for which $\mathsf{REPAIR\text{-}CHECKING}_{(\mathcal{G},b)}$ is tractable. In Sections 4.5.1 and 4.5.2, we define two properties of aggregation functions that give rise to some first tractability results. Then, in Section 4.5.3, we combine these results in our main tractability theorem for $\mathsf{REPAIR\text{-}CHECKING}_{(\mathcal{G},b)}$.

4.5.1 Monotone Under Priority

The converse of the claim at the end of Section 4.4 does not hold. Indeed, MIN is not $\subseteq$-monotone, but it is easily verified that $\mathsf{SUITABILITY\text{-}CHECKING}_{(\mathsf{MIN},b)}$ is in P. We now aim at larger classes of aggregation functions $\mathcal{G}$ for which $\mathsf{SUITABILITY\text{-}CHECKING}_{(\mathcal{G},b)}$ is in P. The computational complexity of this problem is mainly incurred by the saturation property in Definition 4.5, as there can be exponentially many sets including a given independent set. Therefore, we are looking for conditions that avoid such an exponential search. Such a condition is given in Definition 4.8.

Definition 4.8 (Monotone under priority).
We say that an aggregation function $\mathcal{G}$ is *monotone under priority* if for every weighted set V, for every $I \subseteq V$, it is possible to compute, in polynomial time in $|V|$, a set $S \subseteq V \setminus I$ whose powerset 2^S contains all and only those subsets of $V \setminus I$ that can be unioned with I without incurring a decrease of the aggregated value (i.e., for every $J \subseteq V \setminus I$, the following holds true: $J \subseteq S$ if and only if $\mathcal{G}_{\triangleright w}(I) \leq \mathcal{G}_{\triangleright w}(I \cup J)$).

We write $\mathbf{AGG}^{\mathsf{poly}}_{\mathsf{mon}}$ for the set of aggregation functions in $\mathbf{AGG}^{\mathsf{poly}}$ that are monotone under priority.

$\triangleleft$

To illustrate Definition 4.8, we show that MIN is monotone under priority. To this end, let V be a weighted set and $I \subseteq V$. Clearly, $\mathsf{MIN}_{\triangleright w}(I) \leq \mathsf{MIN}_{\triangleright w}(I \cup J)$ if (and only if) J contains no element with weight strictly smaller than $\mathsf{MIN}_{\triangleright w}(I)$. Therefore, the set $S = \{v \in V \setminus I \mid w(v) \geq \mathsf{MIN}_{\triangleright w}(I)\}$ shows that MIN is monotone under priority. It is even easier to show that every $\subseteq$-monotone aggregation function in $\mathbf{AGG}^{\mathsf{poly}}$ is monotone under priority, by letting $S = V \setminus I$. Therefore, the following lemma is more general than the claim at the end of Section 4.4.

Lemma 4.4. *For every $\mathcal{G} \in \mathbf{AGG}^{\mathsf{poly}}_{\mathsf{mon}}$ and $b \geq 2$, $\mathsf{SUITABILITY\text{-}CHECKING}_{(\mathcal{G},b)}$ is in P.*

Among the six common aggregation functions COUNT, SUM, $\mathsf{PRODUCT}$, MAX, MIN, and AVG, the latter one is the only one that is not in $\mathbf{AGG}^{\mathsf{poly}}_{\mathsf{mon}}$, as illustrated next.

Example 4.1
We show that the aggregation function AVG is not monotone under priority. Let $V = \{a, b, c, d\}$. Let $w : V \to \{1, 2\}$ such that $w(a) = w(b) = 1$ and $w(c) = w(d) = 2$. Let $I = \{a, c\}$. Then, $\mathsf{AVG}_{\triangleright w}(I) = \frac{3}{2}$. The subsets of $V \setminus I = \{b, d\}$ that can be unioned with I without incurring a decrease of AVG are $\{\}$, $\{d\}$, and $\{b, d\}$. However, the set of the latter three sets is not the powerset of some other set.

$\triangleleft$

4.5.2 k-Combinatorial

Lemma 4.4 tells us that $\mathsf{SUITABILITY\text{-}CHECKING}_{(\mathcal{G},b)}$ is in P if $\mathcal{G} = \mathsf{MIN}$ or $\mathcal{G} = \mathsf{MAX}$. However, an easier explanation is that the aggregated values of MIN and MAX over a weighted set are determined by a single element in that set. This observation motivates the following definition.

Definition 4.9 (k-combinatorial).
Let k be a positive integer. We say that an aggregation function $\mathcal{G}$ is k-*combinatorial* if every weighted set I includes a subset J such that $|J| \leq k$ and $\mathcal{G}_{\triangleright w}(J) = \mathcal{G}_{\triangleright w}(I)$. If $\mathcal{G}$ is not k-combinatorial for any k, we say that $\mathcal{G}$ is *full-combinatorial.*

We write $\mathbf{AGG}^{\mathsf{poly}}_{(k)}$ for the set of aggregation functions in $\mathbf{AGG}^{\mathsf{poly}}$ that are k-combinatorial.

$\triangleleft$

Obviously, the aggregation functions MIN and MAX are 1-combinatorial. From this, we can easily obtain an aggregation function that is 2-combinatorial. For example, define SPREAD as the aggregation function such that for every weighted set I, $\mathsf{SPREAD}_{\triangleright w}(I) := \mathsf{MAX}_{\triangleright w}(I) - \mathsf{MIN}_{\triangleright w}(I)$. The notion of k-combinatorial also naturally relates to the well-studied notion of top-k queries. For example, for a fixed k and an aggregation function $\mathcal{G}$, we can define a new aggregation function that, on input of any weighted set (I, w), returns $\max\{\mathcal{G}_{\triangleright w}(J) \mid J \subseteq I, |J| = k\}$, i.e., the highest $\mathcal{G}$-value found in any subset of size exactly k (and returns 0 if $|I| < k$). This new aggregation function is k-combinatorial by construction.

Lemma 4.5. *Let k be a positive integer. For every $\mathcal{G} \in \mathbf{AGG}^{\mathsf{poly}}_{(k)}$ and $b \geq 2$, $\mathsf{REPAIR\text{-}EXISTENCE}_{(\mathcal{G},b)}$ is in P.*

4.5.3 Main Tractability Theorem

By bringing together the results of the two preceding subsections, we obtain our main tractability result.

Theorem 4.6 (Main tractability theorem)**.** *Let k be a positive integer. For every $\mathcal{G} \in \mathbf{AGG}^{\mathsf{poly}}_{(k)} \cap \mathbf{AGG}^{\mathsf{poly}}_{\mathsf{mon}}$, for every $b \geq 2$, $\mathsf{REPAIR\text{-}CHECKING}_{(\mathcal{G},b)}$ is in P.*

It remains an open question whether Theorem 4.6 is often useful in practice, i.e., whether $\mathbf{AGG}^{\mathsf{poly}}_{(k)} \cap \mathbf{AGG}^{\mathsf{poly}}_{\mathsf{mon}}$ captures many aggregation functions of practical interest. To give some insight, we have summarized in Table 4.1 the situation for aggregation functions frequently encountered in practice. We recall that an aggregation function is *anti-monotone* if $\mathcal{G}_{\triangleright w}(I) \geq \mathcal{G}_{\triangleright w}(I \cup J)$ for all pairs of weighted sets I and J.

Table 4.1: Aggregation functions and their properties

aggregation	$\subseteq$-monotone	anti-monotone	$\mathbf{AGG}^{\mathsf{poly}}_{\mathsf{mon}}$	$\mathbf{AGG}^{\mathsf{poly}}_{(k)}$	full-combinatorial
MIN		$\times$	$\times$	$\times$	
MAX	$\times$		$\times$	$\times$	
SUM	$\times$		$\times$		$\times$
AVG					$\times$
COUNT	$\times$		$\times$		$\times$
PRODUCT			$\times$		$\times$

4.6. On Full-Combinatorial Aggregation Functions

Lemma 4.5 stated that the problem $\mathsf{REPAIR\text{-}EXISTENCE}_{(\mathcal{G},b)}$ is tractable if $\mathcal{G}$ is k-combinatorial for some fixed k. We will now show that dropping this condition quickly results in intractability. For a technical reason that will become apparent in the proof of Theorem 4.7, we need the following definition.

Definition 4.10 (witnessable).
Let $\mathcal{G}$ be an aggregation function that is full-combinatorial. We say that $\mathcal{G}$ is *witnessable* if the following task is in polynomial time:

Input: A positive integer k in unary. That is, a string $111\cdots1$ of length k.

Output: Return a shortest sequence $(q_1, q_2, \ldots, q_n)$ of nonnegative rational numbers witnessing that $\mathcal{G}$ is not k-combinatorial ($n > k$). Formally, for the weight function w that maps each i to q_i ($1 \leq i \leq n$), it must hold that for every $N \subseteq \{1, 2, \ldots, n\}$ with $|N| \leq k$, we have $\mathcal{G}_{\rhd w}(N) \neq \mathcal{G}_{\rhd w}(\{1, 2, \ldots, n\})$.

$\lhd$

Clearly, if $\mathcal{G}$ is full-combinatorial, the output requested in Definition 4.10 exists for every k. Therefore, the crux is that the definition asks to return such an output in polynomial time, where it is to be noted that the input is encoded in unary, i.e., has length k (and not $\log k$). Since aggregation functions $\mathcal{G}$ are closed under isomorphism, any permutation of a valid output is still a valid output. An example of a witnessable aggregation function is SUM: on input k, a valid output is the sequence $(1, 1, \ldots, 1)$ of length $k + 1$. For full-combinatorial functions in $\mathbf{AGG}^{\mathsf{poly}}$, the requirement of being witnessable seems very mild, and is expected to be fulfilled by natural aggregation functions.
The following result generalizes a complexity lower bound previously established by Theorem 4.2, because COUNT obviously satisfies the conditions imposed on $\mathcal{G}$ in the following theorem statement.

Theorem 4.7. *Let $\mathcal{G} \in \mathbf{AGG}^{\mathsf{poly}}$ be a full-combinatorial function that is $\subseteq$-monotone and witnessable. Then* $\mathsf{REPAIR\text{-}EXISTENCE}_{(\mathcal{G},2)}$ *is* NP*-complete.*

4.7. Conclusion

Our work combines and generalizes notions from databases and graph theory. From a database-theoretical viewpoint, $\mathcal{G}$-repairs extend subset- and cardinality-repairs by allowing arbitrary aggregation functions. From a graph-theoretical viewpoint, $\mathcal{G}$-repairs extend maximum weighted independent sets by allowing hypergraphs as well as aggregation functions other than SUM. With minor effort, complexity lower bounds for $\mathsf{REPAIR\text{-}CHECKING}_{(\mathcal{G},b)}$ were obtained from known results about maximum (weighted) independent sets. Our main result is the computational tractability result of Theorem 4.6, which shows a polynomial upper time bound on this problem under some restrictions that are not unrealistic (and are actually met by several common aggregation functions).

Throughout this chapter, aggregation functions and their properties were defined and treated in an abstract, semantic way. In the future, we want to study logical languages that allow expressing aggregation functions (e.g., first-order logic with aggregation), and in particular their syntactic restrictions that guarantee tractability.

CHAPTER 5

Rustoner: Computing Ranks Efficiently

5.1. Introduction

Rustoner [88] is a program to compute quality ranks for inconsistent ABoxes in some Description Logic formalism. It is an implementation of the work described in Chapter 3. Rustoner also provides a lightweight reasoning an addition framework for $DL\text{-}Lite_{\mathcal{R}}$ ontologies, focused on exploratory analysis. We are well aware that there exist already several reasoners and tools to work with Description Logics [52,108,114], some with striking efficiency. Therefore, the main contribution of rustoner is not its reasoning capabilities, but rather its back-end tool for computing quality-based ranks of assertions in ABoxes. It should also be mentioned that the ranking framework provided by rustoner is not limited to a Description Logic setting, but applies to any logical framework that allows detecting inconsistencies within a set of affirmations or facts. This includes, for example, inconsistency among tuples of a relational database with respect to a set of integrity constraints.
While rustoner is a single program, it can be conceptually divided into two parts:

- the first part is the quality ranking algorithm, which implements (in an optimized form) the theory in our DL 2020 publication [90];

- the second part is a lightweight *DL-Lite*$_{\mathcal{R}}$ reasoner, initially written to test how the ranking of ABox assertions behaved, and later extended to an exploratory tool to study conflicts.

This chapter is mainly a technical description of the implementation of rustoner. We will assume that the reader is familiar with the underlying theory of Description Logics and basic linear algebra.
The rest of the chapter is organized as follows: Section 5.2 explains the most important parts of the ranking algorithm; Section 5.3 briefly explains the implementation of the *DL-Lite*$_{\mathcal{R}}$ reasoner and shows its tools; finally, Section 5.5 concludes the chapter. The main novelty of this chapter lies in the computation of both the conflict matrix of an ontology and the stabilized rank. A reader desiring to implement or enhance the current version is encouraged to analyze it profoundly. On the other hand, as previously mentioned, the reasoning capabilities of rustoner can also be found in existing reasoners for Description Logics.

5.1.1 Technical Details

Rustoner is written in *rust* [67], a relative young programming language, first appearing between 2010 and 2012. We have chosen *rust* because its speed is comparable to C, its syntax naturally entails memory safety, and it allows programs written in a form close to mathematical language.
The program is publicly accessible at the address `https://github.com/hatellezp/rustoner`, a full walk through of how to use rustoner and its capabilities is there given in the form of a README.
Both the program and the site are under constant development, but a version ready to use, version 0.1.0, is available for download.

5.2. How to Compute Ranks

While the ranking approach can be easily adapted to other logic frameworks, we will explain how it works from a Description Logic point of view. Chapter 3 developed a framework that allows for quality-based ranking of assertions and studied the existence of a stabilized ranking. We briefly recall the main ideas, relative to a fixed ontology $\langle \mathcal{T}, \mathcal{A} \rangle$. We can compute the *conflict matrix* $\mathbb{A}$ of $\mathcal{A}$ with respect to $\mathcal{T}$, which captures how assertions in the ABox interact with each other. From this matrix we define a linear system of equations

$$(a \cdot \mathbb{1} - b \cdot \mathbb{A})\, X = c \cdot \begin{bmatrix} 1 \cdots 1 \end{bmatrix}^{\top},$$

where a, b, c are real positive numbers, and X a real-valued vector. Computing a ranking for $\mathcal{A}$ boils down to solving the system for X and associating each $\alpha_i \in \mathcal{A}$ with its corresponding quality assessment x_i, i.e., the i-th component of X. Furthermore, by the computation of a *stabilized* ranking, we mean the computation of a lower bound for a, say a^*, with respect to a fixed b such that the order on $\mathcal{A}$ induced by the elements in X is the same for every $a \geq a^*$. We can then use this a^* to solve the system and gain a stabilized ranking for our ontology. Thus, two main tasks emerge:

- computing the *conflict matrix* $\mathbb{A}$ relative to $\langle \mathcal{T}, \mathcal{A} \rangle$; and
- finding the bound a^* for a stabilized ranking.

The rest of this section explains how our implementation solves these two tasks.

5.2.1 Computing a Conflict Matrix

Let us fix an ontology $\langle \mathcal{T}, \mathcal{A} \rangle$ for the rest of this subsection. When we mention a conflict matrix $\mathbb{A}$, it is understood to be the conflict matrix relative to $\langle \mathcal{T}, \mathcal{A} \rangle$. We suppose $\langle \mathcal{T}, \mathcal{A} \rangle$ is defined in a Description Logic language that admits *negation*. We also suppose that there are no *self-conflicting* assertions in $\mathcal{A}$, by which we mean that there is no $\alpha \in \mathcal{A}$ such that $\langle \mathcal{T}, \{\alpha\} \rangle \models \bot$. In fact, a self-conflicting assertion must necessarily be false (with respect to $\mathcal{T}$) and would imply every other assertion in $\mathcal{A}$ (*"ex contradictione quodlibet"*), making our subsequent analysis meaningless. Before explaining our algorithms, we recall some notions related to the conflict matrix $\mathbb{A}$. Fix any order on the assertions in $\mathcal{A}$ and let it be equal to $\{\alpha_1, \ldots, \alpha_n\}$ (thus $|\mathcal{A}| = n$).

Supporters and refuters Let $\alpha_i \in \mathcal{A}$ be an assertion and $B \subseteq \mathcal{A}$ a *consistent* subset with respect to $\mathcal{T}$. Suppose that neither α_i nor $\neg\alpha_i$ are present in B. We say that

- B *supports* α_i (or that B is a *supporter* of α_i) if $\langle \mathcal{T}, B \cup \{\neg\alpha_i\}\rangle \models \bot$ and B is $\subseteq$-minimal with this property;
- B *refutes* α_i (or that B is a *refuter* of α_i) if $\langle \mathcal{T}, B \cup \{\alpha_i\}\rangle \models \bot$ and B is $\subseteq$-minimal with this property.

It follows from the definition that the cardinality of supporters and refuters is bounded above by $|\mathcal{A}| - 1$. However, for practical computations, it would be desirable that this cardinality be bounded by a constant that does not depend on $\mathcal{A}$. This gives rise to the notion of *conflict bounded TBox* recalled next.

Conflict boundedness A TBox $\mathcal{T}$ is *conflict bounded* if there exists a positive natural b such that for every ABox $\mathcal{A}$, if $\mathcal{A}$ is inconsistent with respect to $\mathcal{T}$, then there exists $B \subseteq \mathcal{A}$ such that $|B| \leq b$ and B is already inconsistent with respect to $\mathcal{T}$. Thus in the case that $\mathcal{T}$ is conflict bounded, say by b, the cardinality of every supporter and refuter is at most $(b-1)$.

Indicator function and aggregate operator The relation between subsets of $\mathcal{A}$ and assertions is summarized in the *indicator* function I. Let α_i, α_j be two assertions in $\mathcal{A}$, let $B \subseteq \mathcal{A}$ be a subset of $\mathcal{A}$, and let T and F be two constants that stand for **true** and **false** respectively. We define the *indicator* function I as:

$$I(B, l, \alpha_i, \alpha_j) = \begin{cases} 1 \text{ if } \alpha_j \in B,\ l = T, \text{ and } B \text{ supports } \alpha_i; \\ 1 \text{ if } \alpha_j \in B,\ l = F, \text{ and } B \text{ refutes } \alpha_i; \\ 0 \text{ otherwise.} \end{cases} \tag{5.1}$$

The indicator function materializes the interaction (positive and negative) between assertions in the ABox, and constitutes the first building block for the conflict matrix. The second building block is the aggregation function, which makes explicit the expert's belief about the content of the ABox. An *aggregation function* (or aggregate operator) is a map $f : 2^{\mathcal{A}} \to \mathbb{R}^{\geq 0}$ from the subsets of $\mathcal{A}$ to the nonnegative real numbers. Let now f be an aggregate operator with respect to $\mathcal{A}$. The conflict matrix $\mathbb{A}$ is defined by its coefficients:

$$\forall i, j \in \{1, \ldots, n\},\ a_{ij} := \sum_{B \subseteq \mathcal{A}} f(B) \left(I(B, T, \alpha_i, \alpha_j) - I(B, F, \alpha_i, \alpha_j) \right).$$

It is easily verified that $a_{ii} = 0$ for all i. Indeed, since α_i is never present in its supporters or refuters, it follows that $I(B, T, \alpha_i, \alpha_i)$ and $I(B, F, \alpha_i, \alpha_i)$ are zero. We will often write $I(B, T - F, \alpha_i, \alpha_j)$ as a syntactic shorthand for $(I(B, T, \alpha_i, \alpha_j) - I(B, F, \alpha_i, \alpha_j))$. Note nevertheless that we could have defined I in this form without ambiguity. Indeed, since supporters and refuters are consistent and since no assertion in $\mathcal{A}$ is self-conflicting, the following statements hold true:

- $(I(B, T, \alpha_i, \alpha_j) - I(B, F, \alpha_i, \alpha_j)) = 0$ implies that both $I(B, T, \alpha_i, \alpha_j)$ and $I(B, F, \alpha_i, \alpha_j)$ are equal to zero;
- $(I(B, T, \alpha_i, \alpha_j) - I(B, F, \alpha_i, \alpha_j)) = 1$ implies that $I(B, T, \alpha_i, \alpha_j) = 1$ and $I(B, F, \alpha_i, \alpha_j) = 0$; and
- $(I(B, T, \alpha_i, \alpha_j) - I(B, F, \alpha_i, \alpha_j)) = -1$ implies that $I(B, T, \alpha_i, \alpha_j) = 0$ and $I(B, F, \alpha_i, \alpha_j) = 1$.

The matrix $\mathbb{A}$ is constructed in two steps. During the first step, called *data gathering*, we compute both the indicator function and the aggregate operator values for the subsets of $\mathcal{A}$. In the second step, each entry in the matrix is computed. The data gathering algorithm is presented next. A *filter* for $\mathcal{A}$ is a function $\mathbf{F}$ that will provide on demand subsets of $\mathcal{A}$ in an order that we will explain later. The *size* of a filter is simply the cardinality of the subset it defines.

We will now explain some of the structures used in Algorithm 1 and argue that it is correct.

Structures in Algorithm 1

Maps I and C Both maps represent functions, which are implemented as hash tables (or hashmaps). Since the data structures for **I** and **C** are only used for storing and retrieving data, hash tables are more efficient than, for example, arrays that would augment the look-up time of a value whose index is not known.

Array S Both supporters and refuters must be $\subseteq$-minimal by definition. That is, if B is a supporter of α, then no strict subset of B can also be a

Algorithm 1: Data gathering.

Input : An ontology $\langle \mathcal{T}, \mathcal{A} \rangle$, a bound b for conflict size, an aggregate operator f, a map $\mathbf{I}$ and a map $\mathbf{C}$.

Result: Computes the indicator function I of $\langle \mathcal{T}, \mathcal{A} \rangle$ and stores the values in $\mathbf{I}$; computes the values of f with respect to subsets of $\mathcal{A}$ and stores them in $\mathbf{C}$.

```
n ← size of 𝒜;
create a filter F with size n;
create an array for inconsistent subsets S;
for i from 1 to n do
    let α_i be the i-th assertion in 𝒜;
    (re)initialize the filter F;
    while the filter size is less than (b − 1) do
        index_B ← the current value of F's index;
        B ← the subset of 𝒜 encoded by F;
        if α_i is not present in B then
            B_T ← B ∪ {¬α_i};
            B_F ← B ∪ {α_i};
            for each subset C in S do
                if C ⊊ B_T or C ⊊ B_F then
                    F ← next iteration of F;
                    go to next while loop iteration;
                else if C = B_F then
                    I(i, index_B) ← (−1, B);
                    F ← next iteration of F;
                    go to next while loop iteration;
            end
            if ⟨𝒯, B_F⟩ ⊨ ⊥ then
                compute f(B);
                C(index_B) ← f(B);
                I(i, index_B) ← (−1, B);
                add B_F to the array S;
            else if ⟨𝒯, B_T⟩ ⊨ ⊥ then
                compute f(B);
                C(index_B) ← f(B);
                I(i, index_B) ← (1, B);
                add B_T to the array S;
        F ← next iteration of F;
    end
end
```

supporter of α. Likewise, if B is a refuter of α, then no strict subset of B can also be a refuter of α. The array $\mathbf{S}$ keeps track of which supporters and refuters have already been found in order to avoid wrongly marking a subset as a supporter or refuter when the $\subseteq$-minimality condition would be violated. A second purpose of the array $\mathbf{S}$ is explained next. Assume that among all subsets of $\mathcal{A}$ that contain a given assertion α, the subset C is one that is minimal (with respect to $\subseteq$) *inconsistent*. That is, every strict subset of C that contains α is consistent. Then we claim that $C \setminus \{\alpha\}$ is a refuter of α. To show this claim, notice first that $\langle \mathcal{T}, (C \setminus \{\alpha\}) \cup \{\alpha\} \rangle \models \bot$ by our assumption that C is inconsistent and contains α. Then, let C' be a strict subset of $C \setminus \{\alpha\}$. By our hypothesis that C is $\subseteq$-minimal *inconsistent*, it follows that $\langle \mathcal{T}, C' \cup \{\alpha\} \rangle \not\models \bot$. It is now correct to conclude that $C \setminus \{\alpha\}$ is a refuter of α. To further illustrate this principle, suppose $B \subsetneq \mathcal{A}$ and $\alpha, \beta \in \mathcal{A}$ are such that

- neither α nor β are in B;
- each set among B, $B \cup \{\alpha\}$, and $B \cup \{\beta\}$ is consistent with respect to $\mathcal{T}$; and
- $\langle \mathcal{T}, B \cup \{\alpha, \beta\} \rangle \models \bot$.

Then, $B \cup \{\alpha, \beta\}$ is a $\subseteq$-minimal inconsistent subset that contains α, and therefore the set $B \cup \{\beta\}$ is a refuter of α and $I(B \cup \{\beta\}, F, \alpha, \beta) = 1$. By symmetry, $B \cup \{\alpha\}$ is a refuter of β and $I(B \cup \{\alpha\}, F, \beta, \alpha) = 1$.
Our algorithm keeps track of $\subseteq$-minimal inconsistent subsets to avoid needless computations. At the end of the next section, we will show that $\mathbf{S}$ contains *only* $\subseteq$-minimal inconsistent sets, which is a crucial aspect of our algorithm.

The filter F The filter function produces subsets of $\mathcal{A}$ such that

- smaller (with respect to cardinality) sets are produced before larger sets; and
- sets of the same cardinality are produced in lexicographic order.

We illustrate this by an example.

Example 5.1
Let $\mathcal{A} = \{\alpha_1, \alpha_2, \alpha_3\}$. We are interested in producing the subsets of $\mathcal{A}$ in the following order:

- $B_0 = \emptyset$;
- $B_1 = \{\alpha_1\}$;
- $B_2 = \{\alpha_2\}$;
- $B_3 = \{\alpha_3\}$;
- $B_4 = \{\alpha_1, \alpha_2\}$;
- $B_5 = \{\alpha_1, \alpha_3\}$;
- $B_6 = \{\alpha_2, \alpha_3\}$;
- $B_7 = \{\alpha_1, \alpha_2, \alpha_3\}$.

$\triangleleft$

The filter itself is implemented as a pair composed of an index i and an array. The array, of length $|\mathcal{A}|$, has entries that belong to $\{0, 1\}$: the j-th entry is 1 if (and only if) α_j is in the subset being produced. For example, in Example 5.1, $\{\alpha_1, \alpha_3\}$ is represented by $[1, 0, 1]$. This array is initialized to all zeros. The index i is initialized to 0 and is incremented whenever a subset is produced. Whenever called, the filter will do two things:

- return the current value of the filter; and
- move to the filter's next value.

In Example 5.1, if the filter is called with value $(5, [1, 0, 1])$, then it will move to $(6, [0, 1, 1])$. It is important to note that if B_1 is generated before B_2 , then we *always* have that $|B_1| \leq |B_2|$.
This sequencing of subsets of a set has already been studied [80]. The complexity of creating the filter is linear and the complexity of producing the next state (the next subset) is also linear in the worst case. Next we explain how Algorithm 1 works.

Logic of Algorithm 1

The following two assumptions have motivated the structure of Algorithm 1. First, we assume that $f(B)$ can always be computed in polynomial time in the size of B. Second, we assume that testing for ABox consistency with respect to $\mathcal{T}$ is *significantly more time-expensive* than any other instruction, and should therefore be avoided as much as possible.

Lines 4–34 The principal outer loop of the algorithm ranges over all assertions $\alpha_1, \alpha_2, \ldots, \alpha_n \in \mathcal{A}$ in turn, and fills $\mathbf{I}$ with the indicator values for each α_i. For every assertion α_i, the filter is reinitialized at line 6 and will subsequently pass over all subsets of $\mathcal{A}$.

Lines 7–33 A priori we need to know the relation between every subset of $\mathcal{A}$ and the current α_i. We will shortly see that there is a way to do early pruning. The filter $\mathbf{F}$ generates subsets of $\mathcal{A}$ on demand up to size $b-1$. As discussed in Section 5.2.1, because of the bound on the size of $\subseteq$-minimal conflicts, we only search for subsets up to cardinality $b-1$. In line 10, we assure that α_i is not present in B.

Lines 13–21 We search for two different conditions in this block. First, in line 14, we test whether a strict subset C of B_T or B_F has already been found to be inconsistent; if this is the case, then B can never be a supporter or a refuter of α_i. We now argue that this test is correct. To this end, let C be in $\mathbf{S}$ such that $C \subsetneq B_F$ (the case that $C \subsetneq B_T$ is symmetrical). Two cases can occur:

- *Case that α_i is in C.* Then $C\backslash\{\alpha_i\}$ is strictly included in B. If $C\backslash\{\alpha_i\}$ is inconsistent, then C is not a $\subseteq$-minimal inconsistent subset and thus not present in $\mathbf{S}$. Otherwise, if $C\backslash\{\alpha_i\}$ is consistent, then B cannot be a refuter because it will not satisfy the $\subseteq$-minimality condition of refuters.
- *Case that α_i is not in C.* Then C is an inconsistent subset of $\mathcal{A}$ that is included in B, and hence B is also inconsistent, and thus not a refuter.

The rationale for the second test at line 17 was previously explained in the discussion of the array $\mathbf{S}$ on page 75, where we showed that if $B_F = B \cup \{\alpha_i\}$ is inconsistent, then B is a candidate for refuting α_i. Because of the invariant property that $\mathbf{S}$ contains only $\subseteq$-minimal inconsistent sets, B is consistent and hence a refuter of α_i.

Lines 22–27 and lines 27–31 If $\langle \mathcal{T}, B_F \rangle \models \bot$, then B is a refuter of α_i. In this case, the indicator map $\mathbf{I}$ is updated, and B_F is added to the array of inconsistent sets $\mathbf{S}$. If $\langle \mathcal{T}, B_T \rangle \models \bot$, then B is a supporter of α_i, and the treatment is analogous to the preceding case. The duplication of code is intentional: if $\langle \mathcal{T}, B_F \rangle \models \bot$ holds true, then we do *not* test $\langle \mathcal{T}, B_T \rangle \models \bot$. Recall that such inconsistency tests are assumed to be time-expensive and hence should be avoided whenever possible.

S contains only $\subseteq$-minimal inconsistent subsets It is easily verified that every set of $\mathbf{S}$ is inconsistent. We now prove the invariant property that every set in $\mathbf{S}$ is $\subseteq$-minimal inconsistent. For the sake of contradiction, assume that at some point in its execution, the algorithm adds to $\mathbf{S}$ a set B having a strict subset that is inconsistent. Then, there is a minimal (with respect to $\subseteq$) strict subset C of B that is inconsistent. Observe that C is not contained in $\mathbf{S}$; indeed, if C was in $\mathbf{S}$, then the test at line 14 would have succeeded and the iteration would have ended without adding B to $\mathbf{S}$. We now prove that C is contained in $\mathbf{S}$, which yields the desired contradiction. There are two possibilities for C ($\uplus$ denotes disjoint union):

- C is equal to $\{\neg\alpha_{i^*}\} \uplus C'$ where $\neg\alpha_{i^*}$ is not in $\mathcal{A}$; or
- C is equal to $\{\alpha_{i^*}\} \uplus C'$ and i^* is minimal among the indices of assertions in C.

In the first case, because of the $\subseteq$-minimality of C, for every strict subset C'' of C', the set $\{\neg\alpha_{i^*}\} \cup C''$ is consistent and not present in $\mathbf{S}$. Therefore, the execution of the block 13–21 will not result in an early break of the **while** loop. Since $\{\alpha_{i^*}\} \cup C'$ and $\{\neg\alpha_{i^*}\} \cup C'$ cannot both be inconsistent, the test of the block 27–31 will succeed, and $\{\neg\alpha_{i^*}\} \cup C' = C$ is added to $\mathbf{S}$. We now consider the second case, that is, $C = \{\alpha_{i^*}\} \cup C'$ where i^* is minimal among the indices of assertions in C. Thus the first time C is generated is as a candidate for refuting the assertion α_{i^*}. Since C is not generated earlier and is $\subseteq$-minimal inconsistent, all tests in the block 13–21 fail. The test at line 22 succeeds and C is added to $\mathbf{S}$. We conclude that in all cases C is present in $\mathbf{S}$, a contradiction. We conclude by contradiction that B is never added to $\mathbf{S}$.

Entries of the Conflict Matrix $\mathbb{A}$

Once the indicator and aggregate functions have been computed and stored in maps **I** and **C** respectively, what remains is to compute the actual entries of the conflict matrix $\mathbb{A}$. This computation is easy; we provide the algorithm for the sake of completeness.

Algorithm 2: Filling the conflict matrix entries.

Input : Maps **I** and **C** for the indicator and aggregation function respectively, both with respect to an ontology $\langle \mathcal{T}, \mathcal{A} \rangle$.

Result: Computes the conflict matrix $\mathbb{A}$ of $\langle \mathcal{T}, \mathcal{A} \rangle$.

$n \leftarrow$ size of $\mathcal{A}$;
create a matrix $\mathbb{A}$ with dimension $n \times n$ and all entries set to zero;
for *each key* $(i, index_B)$ *in* **I** **do**
    $(\mathrm{ind}_{\mathrm{value}}, B) \leftarrow \mathbf{I}(i, \mathrm{index}_B)$;
    $f(B) \leftarrow \mathbf{C}(\mathrm{index}_B)$;
    for *each* α_j *in* B **do**
        $\mathbb{A}[i, j] \leftarrow \mathbb{A}[i, j] + f(B) * \mathrm{ind}_{\mathrm{value}}$;
    end
end
return $\mathbb{A}$;

Complexity of Computing the Conflict Matrix

The theoretical complexity of computing the conflict matrix was studied in Chapter 3. Nevertheless, we will now provide a more fine-grained complexity analysis of our algorithm. We start with Algorithm 1. Under our assumption that $f(B)$ can be computed in polynomial time in the size of B, there remain three points whose time complexity needs a deeper analysis:

- the number of iterations before the filter **F** meets the stop condition at line 7;
- the number of subsets in **S** to test at line 13; and
- checking consistency with respect to $\mathcal{T}$ at lines 22 and 27.

Since it is easily verified that at most one set is added to **S** in each **while** iteration, the size of **S** is bounded by the number of iterations of the **while** loop, which depends on **F**. The stopping condition on the filter **F** depends on whether or not $\mathcal{T}$ is conflict bounded. If $\mathcal{T}$ is conflict bounded with bound b, then there are no more than $b * |\mathcal{A}|^b$ iterations; otherwise the number of iterations can be exponential in $|\mathcal{A}|$. The complexity of checking consistency depends of the Description Logic language used.

Of practical interest is the case where $\mathcal{T}$ is conflict bounded and consistency checking is in polynomial time in the size of $\mathcal{A}$. In this case, Algorithm 1 executes in polynomial time in the size of $\mathcal{A}$. Note here that if $\mathcal{T}$ is conflict bounded, the sizes of both $\mathbf{I}$ and $\mathbf{C}$ are bounded by a polynomial in the size of $\mathcal{A}$.

The second algorithm, Algorithm 2, allows for an easier analysis. Its time complexity depends only on the size of the indicator map $\mathbf{I}$. We conclude that if $\mathcal{T}$ is conflict bounded, then Algorithm 2 executes in polynomial time in the size of $\mathcal{A}$.

5.2.2 Computing a Stabilized Rank

In this subsection, we show and discuss an algorithm for computing the bound a^* for a stabilized quality rank of assertions in an ABox. The underlying theory was already presented in Chapter 3, culminating in Theorem 3.7. In this section, we will briefly recall the global idea behind the computation. We then present our algorithm and discuss its functioning as well as some implementation choices. We again fix an ontology $\langle \mathcal{T}, \mathcal{A} \rangle$ in some Description Logic language, with $\mathcal{A} = \{\alpha_1, \ldots, \alpha_n\}$.

We recall the specifications of the problem. Given a conflict matrix $\mathbb{A} = (a_{ij})_{1 \leq i,j \leq n}$ relative to an ontology $\langle \mathcal{T}, \mathcal{A} \rangle$ (the dimension of $\mathbb{A}$ is $n \times n$) and a positive real parameter b, we can specify the following system for every positive real number a:

$$(a \cdot \mathbb{1} - b \cdot \mathbb{A})\, X = \begin{bmatrix} 1 \cdots 1 \end{bmatrix}^\top . \tag{5.2}$$

The task is to find a lower bound a^* for the parameter a such that

- for every $a \geq a^*$, the system in Eq. (5.2) has a *unique* solution; and
- for every pair $a_1, a_2 \geq a^*$, the solutions relative to a_1 and a_2 are *rank-equivalent*.

Uniqueness of solutions The system in Eq. (5.2) has a unique solution if and only if the matrix $(a \cdot \mathbb{1} - b \cdot \mathbb{A})$ is invertible. Thanks to the structure of our system matrix, invertibility obtains if $|a| > |b| * \|\mathbb{A}\|$, where $\|.\|$ is some norm in the vector space of matrices. Our approach is to find a *provisional* bound for the a parameter. First we compute

$$M(\mathbb{A}) := \max_{1 \le i \le n} \sum_{1 \le j \le n} |a_{ij}|, \tag{5.3}$$

for which it holds that $M(\mathbb{A}) \ge \|\mathbb{A}\|_\infty$. Thus we set the provisional bound on a to be

$$a_{prov} := b(M(\mathbb{A}) + 1) > b\|\mathbb{A}\|.$$

If a is such that $|a| \ge a_{prov}$, then the system in Eq. (5.2) when associated to a has a unique solution. Next we explain how we find the real bound a^* that produces rank-equivalent solutions. From here on, we assume that all values of a discussed have absolute value greater or equal to a_{prov}.

Assuring rank-equivalent solutions We recall what are rank-equivalent solutions. Two vectors x, y of dimension n are rank equivalent if

$$\text{for all } i, j \text{ such that } 1 \le i < j \le n\text{: } x_i < x_j \text{ iff } y_i < y_j.$$

We will use the following equivalent formulation in our analysis:

$$\text{for all } i, j \text{ such that } 1 \le i < j \le n\text{: } x_i - x_j < 0 \text{ iff } y_i - y_j < 0.$$

Let a be such that

$$(a \cdot \mathbb{1} - b \cdot \mathbb{A})\, X = \begin{bmatrix} 1 \cdots 1 \end{bmatrix}^\top$$

has a unique solution, denoted by X_a. Using Cramer's Rule, the solution has an explicit form:

$$\begin{aligned} X_a &= (x_1^a, \ldots, x_n^a) \\ &= \frac{1}{\det(a \cdot \mathbb{1} - b \cdot \mathbb{A})} \left(\det((a \cdot \mathbb{1} - b \cdot \mathbb{A})_{|1}), \ldots, \det((a \cdot \mathbb{1} - b \cdot \mathbb{A})_{|n}) \right), \end{aligned}$$

where

- $\det(a \cdot \mathbb{1} - b \cdot \mathbb{A})$ is the determinant of the matrix $(a \cdot \mathbb{1} - b \cdot \mathbb{A})$, which is nonzero because of the invertibility of $(a \cdot \mathbb{1} - b \cdot \mathbb{A})$; and
- $\det((a \cdot \mathbb{1} - b \cdot \mathbb{A})_{|i})$ is the determinant of the matrix obtained from $(a \cdot \mathbb{1} - b \cdot \mathbb{A})$ by replacing the i-th column with the vector $\begin{bmatrix} 1 \cdots 1 \end{bmatrix}^\top$.

For readability, we introduce the following notations:

- we write $S(a)$ for the system in Eq. (5.2) associated to parameter a; and
- we write $S_{|i}(a)$ for the system obtained from $S(a)$ by replacing the i-th column with the vector $\begin{bmatrix} 1 \cdots 1 \end{bmatrix}^\top$.

We can thus rewrite the solution X_a as

$$X_a = (x_1^a, \ldots, x_n^a) = \frac{1}{\det(S(a))} \left(\det(S_{|1}(a)), \ldots, \det(S_{|n}(a))\right).$$

We argue that for all $a_1, a_2 \geq a_{prov}$, the signs of $\det(S(a_1))$ and $\det(S(a_2))$ are the same. Since the system in Eq. (5.2) is invertible for all values $a \geq a_{prov}$, it follows that $\det(S(a))$ is nonzero for all $a \geq a_{prov}$. Because the function $a \to \det(S(a))$ is continuous with respect to a, a change in the sign of $\det(S(a))$ would imply that there is some $a_0 \geq a_{prov}$ for which $\det(S(a_0))$ is equal to zero, a contradiction. Therefore, it is correct to conclude that for all $a \geq a_{prov}$, the sign of $\det(S(a))$ is the same.
It follows that for $a \geq a_{prov}$, the sign of

$$x_i^a - x_j^a = \frac{1}{\det(S(a))} \left(\det(S_{|i}(a)) - \det(S_{|j}(a))\right)$$

is determined by (and only by) the values $\det(S_{|i}(a))$ and $\det(S_{|j}(a))$. Consequently, to obtain rank-equivalent solutions, the bound a^* must be such that for all $a_1, a_2 \geq a^*$ and for all $1 \leq i < j \leq n$ we have that

$$\det(S_{|i}(a_1)) - \det(S_{|j}(a_1)) < 0 \text{ iff } \det(S_{|i}(a_2)) - \det(S_{|j}(a_2)) < 0, \tag{5.4}$$

which is equivalent to:

$$\det(S(a_1))(x_i^{a_1} - x_j^{a_1}) < 0 \text{ iff } \det(S(a_2))(x_i^{a_2} - x_j^{a_2}) < 0. \tag{5.5}$$

From Linear Algebra, we know that the function that associates, to each $a \geq a_{prov}$, the quantity $\big(\det(S_{|i}(a)) - \det(S_{|j}(a))\big)$ is a polynomial in the parameter a, whose degree is at most n. Let us call this polynomial p_{ij}. Because $1 \leq i < j \leq n$, there are exactly $\frac{n(n-1)}{2}$ such polynomials. Our approach for finding a^* proceeds in three steps:

- choose $n+1$ values for a with each $a \geq a_{prov}$;
- use these values to interpolate each polynomial p_{ij} with $1 \leq i < j \leq n$; and
- find a global upper bound a^* for the roots to all polynomials p_{ij}.

From the definition of each polynomial and the condition stated in Eq. (5.5), all values $a \geq a^*$ will yield rank-equivalent solutions. Next we present the algorithm that produces such a bound. In the following **fft** stands for a *Fast Fourier Solver*, that is, a function that will apply the Fast Fourier Transform to a vector.

The discussion below follows the same pattern that we used for Algorithm 1: we first discuss the structure of Algorithm 3, and then we explain its implementation.
Algorithm 3 uses simple data structures: V and P are simple arrays. The FFT solver **fft** and our choice of the Fast Fourier Transform as interpolation tool will be explained at the end of this section.

Logic of Algorithm 3

Lines 4–11 We begin by finding the value $M(\mathbb{A})$ relative to $\mathbb{A}$, which was defined by Eq. (5.3). This block has a linear time complexity with respect to the size of $\mathbb{A}$, and a quadratic time complexity with respect to $|\mathcal{A}|$.

Line 13 Our use of the Fast Fourier Transform implies that we will use complex roots of the unity as the $n+1$ points for interpolation. We do not need to store all $n+1$ roots of the unity. Indeed, if $\omega \in \mathbb{C}\backslash\{1\}$ is such that $\omega^{n+1} = 1$, then ω is a $(n+1)$-th root of the unity and every other $(n+1)$-th root of the unity is equal to ω^k for some k in $\{1, \ldots, n+1\}$. Our choice for ω is $\cos(\theta) + i\sin(\theta)$ where $\theta = \frac{2\pi}{n+1}$.

Algorithm 3: Compute the bound a^*

Input : A conflict matrix $\mathbb{A}$ relative to an ontology $\langle \mathcal{T}, \mathcal{A} \rangle$, a real positive value b, and FFT solver **fft**

Result: Computes the bound a^* relative to $\mathbb{A}$

create an array V of size $(n+1) \times n$;
create and array P of size $n+1$;
$a^* \leftarrow 0$;
`// compute` $M(\mathbb{A})$
$M(\mathbb{A}) \leftarrow 0$;
for i *from* 1 *to* n **do**
    $m \leftarrow 0$;
    for j *from* 1 *to* n **do**
        $m \leftarrow m + |a_{ij}|$;
    end
    $M(\mathbb{A}) \leftarrow \max(M(\mathbb{A}), m)$;
end
$a_{prov} \leftarrow b(M(\mathbb{A}) + 1)$;
compute ω, a $(n+1)$-th complex root of 1 distinct from 1;
`// populate array` V
for k *from* 1 *to* $(n+1)$ **do**
    $a \leftarrow \omega^k a_{prov}$;
    $X^k \leftarrow$ the solution of $(a \cdot \mathbb{1} + b \cdot A)\, X = \begin{bmatrix} 1 \cdots 1 \end{bmatrix}^\top$;
    $D \leftarrow \det(a \cdot \mathbb{1} + b \cdot A)$;
    for i *from* 1 *to* n **do**
        $V[k, i] \leftarrow D * (X^k)_i$;
    end
end
for i *in from* 1 *to* $(n-1)$ **do**
    for j *from* $(i+1)$ *to* n **do**
        `// populate array` P `with values relative to` i, j
        for k *from* 1 *to* $(n+1)$ **do**
            $P[k] \leftarrow V[k, i] - V[k, j]$;
        end
        compute coefficients of p_{ij} from P with the FFT solver **fft**;
        $P \leftarrow p_{ij}$ `//` P `actually stores` p_{ij}
        $(d+1) \leftarrow$ real degree of p_{ij};
        `// update the bound` a^*
        $a^* \leftarrow \max\left(a^*, 1 + \max_{1 \le k \le d}\left(\frac{-P[k]}{P[d+1]}\right)\right)$;
    end
end
return a^*;

Lines 14–21 To interpolate an n-degree polynomial, we need $n+1$ points, and compute their image under the function to be interpolated. That is, if we want to interpolate a function f using $n+1$ points $x_1, \dots, x_{n+1}$, we compute the image $y_i = f(x_i)$ of each point under f. This procedure generates $n+1$ pairs $(x_1, y_1), \dots, (x_{n+1}, y_{n+1})$ that will be used to interpolate f.
This block is charged with the data gathering part of the procedure. We choose a single set $\{a_1, \dots, a_{n+1}\}$ with $n+1$ points that will be used to interpolate *all* polynomials p_{ij}. We will solve $n+1$ times the system in Eq. (5.2) and store the solution vectors in the array V. In the next paragraph, we explain how the values in V are then used to compute $p_{ij}(a_k)$ for each k in $\{1, \dots, n+1\}$.
Let a_k be one of the $n+1$ points chosen for interpolation. First we compute the solution X^k to the system $(a_k \cdot \mathbb{1} + b \cdot \mathbb{A})\, X = \begin{bmatrix}1 \cdots 1\end{bmatrix}^\top$ and also the determinant of $(a_k \cdot \mathbb{1} + b \cdot \mathbb{A})$, denoted $\det(S(a_k))$. Note that

$$X^k = \frac{1}{\det(S(a_k))}(S(a_k)_{|1}, \dots, S(a_k)_{|n}).$$

Secondly we store the vector

$$\det(S(a_k)) * X^k = (S(a_k)_{|1}, \dots, S(a_k)_{|n})$$

at row k of the array V. Let p_{ij} be one of the polynomials, then

$$\begin{aligned} p_{ij}(a_k) &= \det(S(a_k))(X_i^k - X_j^k) = S(a_k)_{|i} - S(a_k)_{|j} \\ &= V[k,i] - V[k,j]. \end{aligned}$$

Thus, all values needed for interpolation can be found by solving $n+1$ times the system in Eq. (5.2) and computing a determinant.
Now we explain our choice of the points $a_1, \dots, a_{n+1}$. The FFT solver **fft** uses the roots of the unity as points for interpolation, that is, $\{a_1, \dots, a_{n+1}\}$ must be equal to $\{\omega, \omega^2, \dots, \omega^{n+1}\}$ where ω is a $n+1$ complex root of 1 distinct from 1. For each k in $\{1, \dots, n+1\}$, a_k is equal to $\omega^k a_{prov} = \omega^k b(M(\mathbb{A})+1)$. The fact that a^k is distinct from the value ω^k needed by the FFT solver does not change the solution. Indeed, the systems

$$\left(\omega^k b(M(\mathbb{A})+1) \cdot \mathbb{1} - b \cdot \mathbb{A}\right) X = \begin{bmatrix}1 \cdots 1\end{bmatrix}^\top$$

and

$$\left(\omega^k \cdot \mathbb{1} - \frac{b}{b(M(\mathbb{A})+1)} \cdot \mathbb{A}\right) X = \frac{1}{b(M(\mathbb{A})+1)} \cdot \begin{bmatrix}1 \cdots 1\end{bmatrix}^\top$$

have exactly the same solutions. A condition that every a_k has to verify is that $|a_k| \geq a_{prov}$. By our choice of using roots of the unity, this condition is indeed respected:

$$|a_k| = \left|\omega^k b(M(\mathbb{A}) + 1)\right| = |\omega^k| * |b| * |M(\mathbb{A}) + 1| = b(M(\mathbb{A}) + 1).$$

Lines 22–32 Once that all data has been gathered in V (lines 14–21), we actually compute $p_{ij}(a_k)$ for all values i, j, $1 \leq i < j \leq n$ and k, $1 \leq k \leq n+1$. For each i in $\{1, \ldots, n-1\}$ and j in $\{i+1, \ldots, n\}$, $p_{ij}(a_k)$ is equal to $V[k,i] - V[k,j]$. We store the vector of images $(p_{ij}(a_1), \ldots, p_{ij}(a_{n+1}))$ in the array P. We then perform the Fast Fourier Transform on P with the solver **fft**, and store the result in P. At this point, for k in $\{1, \ldots, n+1\}$, $P[k]$ contains the coefficient of the term with power $k-1$ of the polynomial p_{ij}. In particular, $P[1]$ stores the constant coefficient. The degree of p_{ij} is at most n, but can be strictly less. If d is the degree of p_{ij}, then the leading coefficient of p_{ij} is stored in $P[d+1]$, while the entries $P[d+2], P[d+3], \ldots, P[n+1]$ are all zero. We then compute the Cauchy's bound [59] on polynomial roots, which in this case is equal to

$$1 + \max_{1 \leq k \leq d} \left(\frac{-P[k]}{P[d+1]} \right).$$

We update a^*, and let it be the maximum among the previous bound found and the new one. At the end of this block, a^* is an upper bound for the roots of all polynomials p_{ij}, and we can safely return a^* as the desired output.

Complexity of Algorithm 3 The number of iterations of each loop is explicit in the program. The instructions that need special attention are the following:

- solving a linear system at line 16;
- finding the determinant of a matrix at line 17; and
- computing the Fast Fourier Transform at line 27.

The complexity of the computation in the first two items is polynomial in the size of $\mathcal{A}$. In fact, both tasks are in $\mathcal{O}(n^3)$ where $n = |\mathcal{A}|$, while there are several algorithms that are even faster under under some conditions. The Fast Fourier Transform has a complexity in $\mathcal{O}(n \log n)$. It follows that Algorithm 3 has a complexity of $\mathcal{O}(n^4 \log n)$ with $n = |\mathcal{A}|$. Its actual performance in practice strongly depends on what routine is used for solving the linear system and what FFT solver is used. Currently, rustoner uses LAPACK and BLAS routines for linear algebra tasks [10,92] and *FFTW* (the Fastest Fourier Transform in the West) [49] for FFT tasks.

Why FFT

In this section, we motivate in more detail the use of the Fast Fourier Transform as an interpolation tool in Algorithm 3. In particular, the principal motivation is not the speed of FFT, but rather its stability from a numerical analysis point of view. The computation of the bound a^* is in the first place a numerical procedure. We will first argue that our interpolation problem aims at an objective that is slightly different from classical interpolation. Then we will discuss the *condition number* of a numerical procedure, and finally we explain why the Fast Fourier Transform is a reasonably choice for our setting. The main objective of polynomial interpolation is to better understand a process or function. If we can approximate an unknown function f by a polynomial p, then the behavior of f can be studied by means of p. There are several methods for polynomial interpolation [61, 93, 97]. In any case, its objective is rarely to output the coefficients of the polynomial. This is different from our problem where we *know* that the function f to be interpolated is a polynomial, and we are interested in its coefficients. The following is a straightforward solution to compute the coefficients of a polynomial. Let $p(X) = \sum_{0 \leq i \leq n} p_i * X^i$ be a polynomial of degree n, and let $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$ be $n + 1$ couples such that $p(x_i) = y_i$ for $0 \leq i \leq n$. We can write all the $n+1$ equalities as follows:

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{pmatrix} * \begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix} \tag{5.6}$$

The left matrix in Eq. (5.6) is a *Vandermonde* matrix, and is invertible if and only if $x_0, x_1, \ldots, x_n$ are all pairwise distinct. Since $x_0, x_1, \ldots, x_n$ are

$n+1$ distinct points in our case, we can easily find the coefficients of our polynomial p by means of inverting the matrix:

$$\begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_n \end{bmatrix} = \begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^n \end{pmatrix}^{-1} * \begin{bmatrix} y_0 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \tag{5.7}$$

The issue with inverting a matrix, in particular a Vandermonde matrix, is that we have to take into account its *condition number*. The condition number [17] of a matrix A is defined as:

$$\kappa(A) := \|A\| \left\|A^{-1}\right\|. \tag{5.8}$$

This value is a measure of the sensitivity of the solution of a linear system, in our case Eq. (5.6), to perturbations or changes in the corresponding matrix. When $\kappa(A)$ is large, the stability of the linear system is weak and small perturbations of A can move us far away from the actual solution. The number $\kappa(A)$ is bounded below by 1, but has no upper bound. When $\kappa(A) = 1$, we say that A is *perfectly-conditioned*. On the other hand, when $\kappa(A)$ is large, we say that A is *ill-conditioned*. Perturbations or changes in A can be due, for example, to rounding errors from floating point arithmetic. This is a problem when our objective is to precisely determine the coefficients of a polynomial. Even worse, Vandermonde matrices tend to be ill-conditioned [87]. In general, determining the coefficients of an interpolating polynomial is an ill-conditioned problem, for a multitude of algorithms [64, 95].

Nevertheless, not all Vandermonde matrices are ill-conditioned. Let $n \in \mathbb{N}^{>0}$ be a natural, and $\omega \in \mathbb{C}\backslash\{1\}$ be such that $\omega^{n+1} = 1$, an $(n+1)$-th root of the unity. Consider the following Vandermonde matrix:

$$V = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^n \\ 1 & \omega^2 & (\omega^2)^2 & \cdots & (\omega^2)^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^n & (\omega^n)^2 & \cdots & (\omega^n)^n \end{pmatrix} \tag{5.9}$$

The matrix V is perfectly-conditioned, that is, $\kappa(V)$ is equal to 1. The matrix $\frac{1}{\sqrt{n}}V$ is called a *DFT* matrix and is essential to the Discrete Fourier Transform and the Fast Fourier Transform [50].
By using complex roots of unity and the Fast Fourier Transform, we benefit from the following advantages:

- the coefficients of each polynomial can be found with only one matrix multiplication;
- since the condition number is $\kappa = 1$, no theoretical precision is lost during the computation; and
- as a bonus, matrix multiplication of the DFT matrix can be done in $\mathcal{O}(n \log n)$ time complexity instead of the usual $\mathcal{O}(n^2)$ [41].

5.3. Inner *DL-Lite*$_{\mathcal{R}}$ Reasoner

The logical basis and reasoning capabilities of *DL-Lite* are by now well understood [23, 31, 98]. Moreover, several extensions to the original logic exist [44, 56, 86], and systems for solving ontological tasks have been developed, including translations to **SAT** and more logic-based approaches such as tableau algorithms [16, 69, 83, 84, 107, 121].
The reasoning capabilities of `rustoner` are by no means as powerful as such reasoners. Nonetheless, we believe that `rustoner` is a useful tool for studying interactions in simple ontologies and for educational purposes. The rest of this section is organized as follows. We first show, from an abstract point of view, how *DL-Lite*$_{\mathcal{R}}$ ontologies are modeled in `rustoner`. We then exhibit the reasoning functions of `rustoner`, and finally show its capabilities for exploratory analysis.

5.3.1 The *DL-Lite*$_{\mathcal{R}}$ Model

The syntax of *DL-Lite*$_{\mathcal{R}}$ is the following:

- basic roles: $s \to r \mid r^-$;
- complex roles: $q \to s \mid \neg s$;

- basic concepts: $B \to A \mid \exists s$;
- complex concepts: $C \to B \mid \neg B$.

Example 5.2
Let *teaches* be a role symbol (or atomic role) and $\mathsf{Student}$ be a concept symbol (or atomic concept). We could interpret *teaches* as a relation modeling who teaches which course, and $\mathsf{Student}$ as the set of people following some course. Then

- $teaches$;
- $teaches^-$;
- $\neg teaches$; and
- $\neg(teaches^-)$

are all valid roles, where $teaches, teaches^-$ are basic, and $\neg teaches, \neg(teaches^-)$ are complex. In the same way

- $\mathsf{Student}$;
- $\neg\mathsf{Student}$;
- $\exists teaches$;
- $\exists teaches^-$;
- $\neg\exists teaches$; and
- $\neg\exists teaches^-$

are valid concepts, where $\mathsf{Student}$, $\exists teaches$, $\exists teaches^-$ are basic, while the other concepts are complex. Note that complex roles and concepts are characterized by the use of *negation*.

$\triangleleft$

In $DL\text{-}Lite_{\mathcal{R}}$, the following types of inclusion can appear in the TBox:

$$\begin{aligned} &\text{concept inclusions: } && B \sqsubseteq C \\ &\text{role inclusions: } && s \sqsubseteq q. \end{aligned} \tag{5.10}$$

Note that the left-hand in both types of inclusion must be basic. When the right-hand of a TBox inclusion is negated, we say that it is a *negative inclusion*; otherwise it is a *positive inclusion.* ABox assertions can be of two forms:

$$\begin{aligned} &\text{concept assertion: } && a : A \\ &\text{role assertion: } && (a, b) : r \end{aligned} \tag{5.11}$$

where a, b are constants, A is a concept and r is a role. While A and r are atomic in a first approach, for reasons that will become apparent shortly, our technical development will also allow for negation of atomic assertions.
In our model of *DL-Lite*$_{\mathcal{R}}$, we add both $\bot$ and $\top$ as basic constructs. A *DL-Lite*$_{\mathcal{R}}$ ontology in rustoner is not a couple $\langle \mathcal{T}, \mathcal{A} \rangle$, but a triplet $\langle \mathcal{S}, \mathcal{T}, \mathcal{A} \rangle$ where

- $\mathcal{S}$ is a map where each element is of the form (symbol: type), where the three possible types are "concept", "role", and "individual";
- $\mathcal{T}$ is represented by an array of tuples of the form (A, B) which means that the assertion $A \sqsubseteq B$ is present in $\mathcal{T}$; and
- $\mathcal{A}$ is represented by an array of tuples of the form (a, A) or (a, b, r) meaning that $a : A$ is in $\mathcal{A}$ or that $(a, b) : r$ is in $\mathcal{A}$.

Example 5.3
Let $\langle \mathcal{T}, \mathcal{A} \rangle$ be the ontology with $\mathcal{T} = \{A \sqsubseteq \neg B\}$ and $\mathcal{A} = \{a : A, (c, d) : r\}$. Our representation of this ontology is as follows:

- $\mathcal{S} = \{(\bot : \text{concept}),\ (\top : \text{concept}),\ (A : \text{concept}),\ (B : \text{concept}), (r : \text{role}),\ (a : \text{individual}),\ (c : \text{individual}),\ (d : \text{individual})\}$;
- $\mathcal{T} = \{(A, \neg B)\}$; and
- $\mathcal{A} = \{(a, A), (c, d, r)\}$.

We omit the details of the actual technical implementation.

$\triangleleft$

The set $\mathcal{S}$ always stores information about both $\bot$ and $\top$; it never stores $(\neg B : \text{concept})$ since it suffices to keep track of the types of atomic constructs.

The structure $\mathcal{S}$ serves several purposes, such as evaluating more complex constructs and knowing when a deduction rule can be applied.
For reasoning tasks, `rustoner` uses deduction rules [1]. Reasoning will be discussed in the next section, but we present here our conception of deduction rules. A deduction rule is a couple (h, t) of two lists, where h is the hypothesis, and t the consequence. Both lists are composed of elements that can represent a TBox inclusion, an ABox assertion, or the declaration of a symbol's type.

Example 5.4
Let the following be two different rules:

- $$\frac{\vdash x : Y}{\vdash X \sqsubseteq Y \wedge \vdash x : X}$$
- $$\frac{\vdash X \sqsubseteq Z}{\vdash X \sqsubseteq Y \wedge \vdash Y \sqsubseteq Z}$$

Note that throughout this chapter, the consequence is above the horizontal bar, and the hypothesis is below the horizontal bar. Informally, the first rule says that "if x is an X, and X implies Y, then x is a Y." The second says that "if X implies Y, and Y implies Z, then X implies Z." The first rule will be encoded by the following lists:

$$\begin{aligned} h &= \{(X, Y), (x : X), (x : \text{individual}), (X : \text{concept}), (Y : \text{concept})\} \\ t &= \{(x : Y), (x : \text{individual}), (Y : \text{concept})\} \end{aligned} \tag{5.12}$$

The second rule is encoded as follows:

$$\begin{aligned} h &= \{(X, Y), (Y, Z), (X : \text{typ}), (Y : \text{typ}), (Z : \text{typ})\} \\ t &= \{(X : Z), (X : \text{typ}), (Y : \text{typ})\}. \end{aligned} \tag{5.13}$$

Remark that in (5.13), we do not specify a particular type, but require that all constructs involved be of the same type "typ". This is because this rule applies to both concepts and roles, that is, the symbol "typ" is a placeholder for either "concept" or "role."

◁

[1]Deduction rules are at the basis of many DL reasoners - not just [31] but also so-called "consquence-based" reasoners that have been developed for more expressive DLs, e.g. "Consequence-Driven Reasoning for Horn SHIQ Ontologies", work of Yevgeny Kazakov.

5.3.2 *DL-Lite*$_{\mathcal{R}}$ Reasoning in Rustoner

The only reasoning task implemented in rustoner is a check for ABox consistency. It should be clear from Section 5.2 that this is sufficient for computing the conflict matrix of an ontology and its entailed quality ranking. In addition, rustoner implements some tools for exploratory analysis, which heavily rely on ABox consistency checks. In this section, we explain how our implementation of this task works.

In one of the initial works about *DL-Lite*, Calvanese et al. [31] specify how to build an algorithm for checking ABox consistency in a *DL-Lite*$_{\mathcal{R}}$ ontology $\langle \mathcal{T}, \mathcal{A} \rangle$. The procedure described there suffices to build a simple, non-optimized reasoner. Our implementation follows their work, guaranteeing the correctness of our approach. Nonetheless, as we will explain shortly, some extensions are needed because the conventional ABox consistency of [31] is not enough to build a conflict matrix.

The remainder of this section is organized as follows. We first recall the procedure defined in [31]. We then argue that this procedure is insufficient to build the conflict matrix, and finally we propose our additions to the original procedure that allows us to build the conflict matrix.

Procedure to Check ABox Consistency

Let $\langle \mathcal{T}, \mathcal{A} \rangle$ be a *DL-Lite*$_{\mathcal{R}}$ ontology. To check for ABox consistency, there are mainly three steps:

1. build a database $db(\mathcal{A})$ from $\mathcal{A}$, which will be tested for consistency with respect to $\mathcal{T}$. We call this database the *potential model*;

2. from $\mathcal{T}$, build $cln(\mathcal{T})$, the closure of $\mathcal{T}$ with respect to negative inclusions; and

3. build a query q_{unsat} from $cln(\mathcal{T})$, such that q_{unsat} answers **true** on $db(\mathcal{A})$ if and only if the ontology $\langle \mathcal{T}, \mathcal{A} \rangle$ is unsatisfiable.

The potential model $db(\mathcal{A})$ The structure $db(\mathcal{A})$ is simply a grounding of $\mathcal{A}$:

- $\Delta^{db(\mathcal{A})} = \{a \mid a \text{ is a constant occurring in } \mathcal{A}\}$;

- $a^{db(\mathcal{A})} = a$, for each constant a;
- $A^{db(\mathcal{A})} = \{a \mid (a : A) \in \mathcal{A}\}$ for each atomic concept A; and
- $r^{db(\mathcal{A})} = \{(a, b) \mid ((a, b) : r) \in \mathcal{A}\}$ for each atomic role r.

The negative closure $cln(\mathcal{T})$ The negative closure $cln(\mathcal{T})$ can be built from $\mathcal{T}$ as follows. Every TBox negative inclusion in $\mathcal{T}$ is present in $cln(\mathcal{T})$. Then we apply the following rules to $cln(\mathcal{T})$ until no more inclusions are added. Recall that in our notation, the consequence is above the horizontal bar.

- $\mathbf{dr(1)}$:

$$\frac{\vdash X \sqsubseteq \neg Z}{\vdash X \sqsubseteq Y \ \wedge \ (\vdash Y \sqsubseteq \neg Z \vee \vdash Z \sqsubseteq \neg Y)}$$

- $\mathbf{dr(2)}$:

$$\frac{\vdash \exists r \sqsubseteq \neg X}{\vdash r \subseteq s \ \wedge \ (\vdash \exists s \sqsubseteq \neg X \vee \vdash X \sqsubseteq \neg \exists s)}$$

- $\mathbf{dr(3)}$:

$$\frac{\vdash \exists r^- \sqsubseteq \neg X}{\vdash r \subseteq s \ \wedge \ (\vdash \exists s^- \sqsubseteq \neg X \vee \vdash X \sqsubseteq \neg \exists s^-)}$$

- $\mathbf{dr(4)}$:

$$\frac{\vdash r \subseteq \neg q}{\vdash r \subseteq s \ \wedge \ (\vdash s \subseteq \neg q \vee \vdash q \subseteq \neg s)}$$

- $\mathbf{dr(5)}$:

$$\frac{\vdash \exists r \sqsubseteq \neg \exists r \qquad \vdash \exists r^- \sqsubseteq \neg \exists r^- \qquad \vdash r \subseteq \neg r}{\vdash \exists r \sqsubseteq \neg \exists r \vee \vdash \exists r^- \sqsubseteq \neg \exists r^- \vee \vdash r \subseteq \neg r}$$

The notation $\mathbf{dr(i)}$ stands for deduction rule i. The use of disjunction in the hypothesis of a rule is a convenient syntactic shorthand, with its natural meaning. For example, $\mathbf{dr(2)}$ is a shorthand for the following two rules:

$$\frac{\vdash \exists r \sqsubseteq \neg X}{\vdash r \subseteq s \ \wedge \ \exists s \sqsubseteq \neg X}$$

and

$$\frac{\vdash \exists r \sqsubseteq \neg X}{\vdash r \subseteq s \ \wedge \ X \sqsubseteq \neg \exists s}$$

Likewise, the use of multiple consequences, as in **dr(5)**, is a syntactic shorthand with the meaning that each of the consequences can be derived whenever the hypothesis can be derived. The encoding of these shorthand rules is as follows:

- if a disjunction appears in the hypothesis, then more than one list h appears in the body of the rule; and
- if more than one expression appears in the thesis, then more than one list t appears in the head.

For example, **dr(5)** would be represented as

$$((h_1, h_2, h_3), (t_1, t_2, t_3)).$$

Whenever some h_i can be derived, then every t_i will be derived. This procedure terminates. Indeed, since the alphabet of concept names and role names is fixed, there are only finitely many valid concepts and roles that can be constructed, and thus only a finite number of TBox inclusions can be added.

Consistency check Consistency checking is done by means of executing q_{unsat} on $db(\mathcal{A})$. The query q_{unsat} is a disjunction $q_{unsat} = \bigvee_i q_i$ where each q_i is generated from a negative inclusion in $cln(\mathcal{T})$ as follows. To shorten the theoretical development, we will use the shortcut $g(r, a, b)$ for roles, where $g(r, a, b) = r'(a, b)$ if $r = r'$, and $g(r, a, b) = r'(b, a)$ if $r = (r')^{-1}$.

1. if $A \sqsubseteq \neg B \in cln(\mathcal{T})$, then some q_i equals
$$(\exists x,\ A(x) \wedge B(x))\,;$$
2. if $A \sqsubseteq \neg \exists r \in cln(\mathcal{T})$, then some q_i equals
$$(\exists x,\ A(x) \wedge (\exists y,\ g(r, x, y)))\,;$$

3. if $\exists r \sqsubseteq \neg B \in cln(\mathcal{T})$ then some q_i equals

$$(\exists x,\ (\exists y,\ g(r,x,y)) \wedge B(x))\,;$$

4. if $\exists r \sqsubseteq \neg\exists s \in cln(\mathcal{T})$, then some q_i equals

$$(\exists x,\ (\exists y,\ g(r,x,y)) \wedge (\exists z,\ g(s,x,z)))\,;$$

5. if $r \subseteq \neg s \in cln(\mathcal{T})$, then some q_i equals

$$(\exists x, \exists y,\ g(r,x,y) \wedge g(s,x,y))\,.$$

It is proved in [31] that the procedure described above decides ABox consistency for *DL-Lite*$_\mathcal{R}$ ontologies. Moreover, since *DL-Lite*$_\mathcal{R}$ TBoxes are restricted to inclusions, concept assertions, and role assertions, it can be verified that every TBox in *DL-Lite*$_\mathcal{R}$ is conflict bounded with bound 2.
Our aim is to compute supporters and refuters, as defined in Definition 3.1, by checking for ABox consistency. However, the following technical difficulty occurs. For supporters, Definition 3.1 contains a test $\langle \mathcal{T}, B\rangle \models \alpha$. Of course, such a test can be performed by using query entailment and query rewriting [16]. However, for reasons of simplicity and uniformity, we have chosen to replace the previous test by an equivalent test that checks for the inconsistency of $\langle \mathcal{T}, B \cup \{\neg\alpha_i\}\rangle$. This latter test, however, uses a negated atom $\neg\alpha_i$ in the ABox, a feature that is not present as such in the approach of [31]. Indeed, the algorithm in [31] is developed for conventional *DL-Lite* and *DL-Lite* related ontologies where assertions of the form $a : \neg C$ are disallowed. However, as we will explain shortly, this feature can be added with minor effort. We first give a concrete example that illustrates the technical difficulty.

Example 5.5
Let $\langle \mathcal{T}, \mathcal{A}\rangle$ be an ontology with $\mathcal{T} = \{A \sqsubseteq B, B \sqsubseteq C\}$ and $\mathcal{A} = \{a : A, a : C\}$. Clearly, $\{a : A\}$ is a supporter of $\{a : C\}$. Indeed, if we apply the definition of supporter, $\langle \mathcal{T}, \{a : A, a : \neg C)\}\rangle \models \bot$, while $\langle \mathcal{T}, \{a : A\}\rangle \not\models \bot$. Here, it should be noted that $a : \neg C$ is equivalent $\neg(a : C)$.
The reason why $\langle \mathcal{T}, \{a : A, a : \neg C)\}\rangle \models \bot$ is obviously that $\mathcal{T}$ logically implies $A \sqsubseteq C$, and therefore $a : A$ implies $a : C$, contradicting $a : \neg C$. However,

this explanation uses negated assertions and a "positive" closure, two features that are not needed in the ABox consistency checks of [31].

$\triangleleft$

In the following section, we explain that with a simple extension of the procedure in [31], we are able to compute refuters and supporters by using only checks for ABox consistency.

Extended ABox Consistency Procedure

From Example 5.5, two difficulties become evident. The first is that $\{a : A, a : \neg C\}$ is not a valid *DL-Lite*$_{\mathcal{R}}$ ABox. This can be solved by allowing, at the moment of checking consistency, negative ABox assertions to appear in the ABox. The second difficulty is more subtle and comes from the conception of the algorithm devised in [31] to detect ABox inconsistency. In fact, even if negative ABox assertions would be allowed, the original algorithm would answer that $\langle \mathcal{T}, \{a : A, a : \neg C\}\rangle$ is a consistent ontology. The reason for this is that the TBox positive inclusion $A \sqsubseteq C$ does not occur in $cln(\mathcal{T})$, and even if it did, q_{unsat} does not check for violations of positive inclusions, that is, q_{unsat} contains no queries of the form

$$\exists x,\ A(x) \land \neg C(x).$$

Thus, our framework requires to detect the inconsistency of $\langle \mathcal{T}, \{a : A, a : \neg C)\}\rangle$ relative to the inferred positive inclusion $A \sqsubseteq C$. To this end, we extend the procedure of [31] as follows. We will allow negated assertions $a : \neg C$ (or equivalently $\neg(a : C)$) in both ABoxes and queries. We will use the notation $a : \neg^{*} C$ with the meaning that "the assertion $a : \neg C$ *is* present in the ABox." This is different from the usual negation $\neg(a : C)$ which means that "$a : C$ *is not* present in the ABox." Then, $a : \neg^{*} C$ will be treated as a usual assertion in $db(\mathcal{A})$ and q_{unsat}.

We add positive deduction rules to the set of rules $\mathbf{dr(1)}$–$\mathbf{dr(5)}$ to make $cln(\mathcal{T})$ not only contain all negative inclusions entailed by $\mathcal{T}$, but also all entailed positive inclusions. Then we add new subqueries to q_{unsat} to detect violations of positive inclusions. The added deduction rules are the following:

- $\mathbf{dr(6)}$:

$$\frac{\vdash X \sqsubseteq Z}{\vdash X \sqsubseteq Y \land \vdash Y \sqsubseteq Z}$$

- **dr(7)**:

$$\frac{\vdash X \sqsubseteq \exists s}{\vdash r \subseteq s \land \vdash X \sqsubseteq \exists r}$$

- **dr(8)**:

$$\frac{\vdash X \sqsubseteq \exists s^{-1}}{\vdash r \subseteq s \land \vdash X \sqsubseteq \exists r^{-1}}$$

- **dr(9)**:

$$\frac{\vdash r \subseteq q}{\vdash r \subseteq s \land \vdash s \subseteq q}$$

- **dr(10)**:

$$\frac{\vdash r \subseteq s \qquad \vdash r^{-1} \subseteq s^{-1}}{\vdash r \subseteq s \lor \vdash r^{-1} \subseteq s^{-1}}$$

The logical closure with respect to $\mathcal{T}$ will still be denoted $cln(\mathcal{T})$. Its computation starts with $\mathcal{T}$, and then repeatedly applies all inference rules until no new inclusions can be inferred.

If α is an assertion of the form $a : C$ where C is not negated, then the potential model $db(\mathcal{A})$ is allowed to contain $a : \neg^* C$, with the meaning that $\neg\alpha$ is present *as such* in the ABox.

The query q_{unsat} is extended with a number of new subqueries q_i, as follows:

1. if $A \sqsubseteq B \in cln(\mathcal{T})$, then some q_i equals

$$(\exists x,\ A(x) \land \neg^* B(x))\,;$$

2. if $A \sqsubseteq \exists r \in cln(\mathcal{T})$, then some q_i equals

$$(\exists x,\ A(x) \land (\exists y,\ \neg^* g(r, x, y)))\,;$$

3. if $\exists r \sqsubseteq B \in cln(\mathcal{T})$, then some q_i equals

$$(\exists x,\ (\exists y,\ g(r,x,y)) \wedge \neg^* B(x))\,;$$

4. if $\exists r \sqsubseteq \exists s \in cln(\mathcal{T})$, then some q_i equals

$$(\exists x,\ (\exists y,\ g(r,x,y)) \wedge (\exists z, \neg^* g(s,x,z)))\,;$$

5. if $r \subseteq s \in cln(\mathcal{T})$, then some q_i equals

$$(\exists x, \exists y,\ g(r,x,y) \wedge \neg^* g(s,x,y))\,.$$

Significantly, a negated atom in a subquery q_i holds true if the ABox contains a corresponding negated assertion. Recall that we use $\neg^*$ to make explicit the search for negated assertions in the potential model $db(\mathcal{A})$. For example, the subquery $(\exists x, A(x) \wedge \neg^* B(x))$ is true if (and only if) for some a, the ABox contains both $a : A$ and $a : \neg^* B$. This is in contrast with the usual use of negation where $a : \neg B$ would be true if $db(\mathcal{A})$ *does not* contain $a : B$.
We now explain how we find refuters and supporters in the case of *DL-Lite*$_{\mathcal{R}}$. Let α be an assertion in $\mathcal{A}$. Since TBoxes in *DL-Lite*$_{\mathcal{R}}$ are conflict bounded with bound 2, it follows that all refuters and supporters are singletons. For an assertion $\beta \in \mathcal{A}$, the following tests apply:

- $\{\beta\}$ is a refuter of α if and only if $\langle \mathcal{T}, \{\alpha, \beta\}\rangle \models \perp$. In this case, we use the classical ABox consistency test, which creates $db(\{\alpha, \beta\})$, and finds $cln(\mathcal{T})$ with rules $\mathbf{dr(1)}$–$\mathbf{dr(5)}$. The query q_{unsat} can be restricted to subqueries for conflicts with negative inclusions;

- $\{\beta\}$ is a supporter of α if and only of $\langle \mathcal{T}, \{\neg\alpha, \beta\}\rangle \models \perp$. In this case, we use our augmented ABox consistency test, which produces $db(\{\neg\alpha, \beta\})$, and creates $cln(\mathcal{T})$ using rules $\mathbf{dr(6)}$–$\mathbf{dr(10)}$. The subquery q_{unsat} uses only the subqueries assoiated with positive inclusions.

Example 5.6
Consider the ontology $\mathcal{T} = \{A \sqsubseteq B, B \sqsubseteq C, A \sqsubseteq \neg D\}$ and $\mathcal{A} = \{a : A, a : C, a : D\}$. We apply the previously described procedure to find that $\{a : A\}$ is a supporter of $a : C$ and a refuter of $a : D$.

To find that $a : A$ is a refuter of $a : D$, we check for consistency of the ABox $\{a : A, a : D\}$ with respect to $\mathcal{T}$. In this case the rules used are the original ones, and $cln(\mathcal{T}) = \{A \sqsubseteq \neg D\}$. The only subquery of q_{unsat} is $\exists x, A(x) \wedge D(x)$, which is satisfied by $\{a : A, a : D\}$.
To find that $a : A$ is a supporter of $a : C$ we use the second procedure. Using all rules, $cln(\mathcal{T})$ also includes positive inclusions:

$$cln(\mathcal{T}) = \mathcal{T} \cup \{A \sqsubseteq C\}.$$

The query q_{unsat} is larger, as we include the search for violations of positive inclusions:

$$\begin{aligned} q_{unsat} =& (\exists x,\ A(x) \wedge \neg^* B(x)) \vee (\exists x,\ B(x) \wedge \neg^* C(x)) \vee \\ & (\exists x,\ A(x) \wedge \neg^* C(x)). \end{aligned} \tag{5.14}$$

Since the query q_{unsat} is satisfied by $\{a : A, a : \neg C\}$, it is correct to conclude that $\{a : A\}$ is a supporter of $a : C$.

$\triangleleft$

5.3.3 Exploratory Analysis with Rustoner

Rustoner also contains a graphical tool that can be used to better understand ontologies. It provides three different functionalities given a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$:

- show the closure $cln(\mathcal{T})$ as a graph which keeps track of the rules that generated each new TBox inclusion;
- show the consequences of $\mathcal{T}$ acting on $\mathcal{A}$; and
- show the conflict graph relative to the conflict matrix of $\langle \mathcal{T}, \mathcal{A} \rangle$.

We show each functionality by means of an example.

Example 5.7
Consider the small TBox:

$$\mathcal{T}_{\mathrm{ex}_1} = \{A \sqsubseteq B, B \sqsubseteq C, C \sqsubseteq \neg A\}.$$

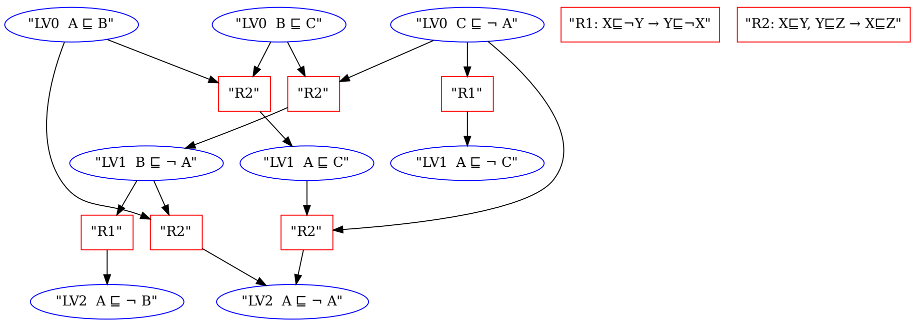

Figure 5.1: Consequences of $\mathcal{T}_{\mathrm{ex}_1}$

For exploratory analysis, it is useful to have a visual help of how the TBox inclusions in $\mathcal{T}$ interact with deduction rules. Figure 5.1 shows how this can be visualized in rustoner.
The graph in Figure 5.1 should be read as follows. The oval blue nodes contain TBox inclusions. The labels LV0, LV1, … specify at which level the inclusions are derived: LV0 inclusions are part of the given TBox, and inclusions at level LVi are derived by applying an inference rule that uses at least one inclusion at level LVj with $j = i - 1$. The red boxes indicate the rules that were applied.

$\triangleleft$

Example 5.8
Let $\langle \mathcal{T}, \mathcal{A} \rangle$ be the following ontology:

$$\mathcal{T} = \{A \sqsubseteq \exists r\},\ \mathcal{A} = \{a : A, (a, b) : r\}. \tag{5.15}$$

Figure 5.2 shows the result of making explicit the consequences of $\mathcal{T}$ acting on $\mathcal{A}$. The green oval nodes are ABox assertions, which were not present in Example 5.7. The levels LV0 and LV1 are as explained in that previous example.

$\triangleleft$

Example 5.9

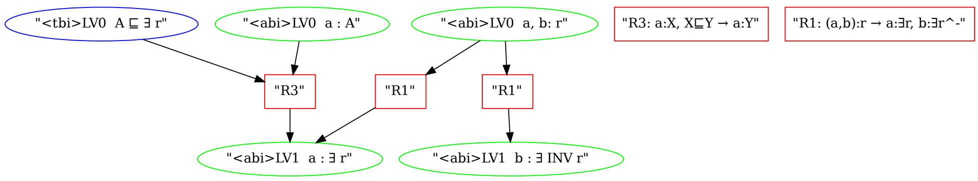

Figure 5.2: Deduction graph of $(\mathcal{T}_{\text{ex}_2}, \mathcal{A}_{\text{ex}_2})$

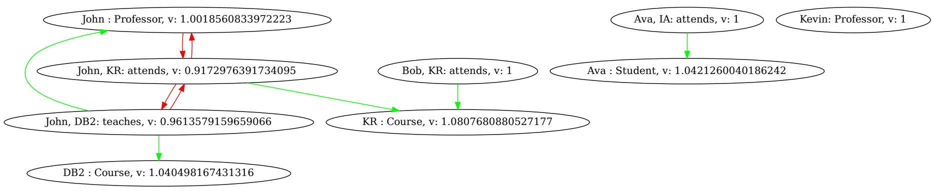

Figure 5.3: Deduction graph of $(\mathcal{T}_{\text{ex}_2}, \mathcal{A}_{\text{ex}_2})$

For the conflict graph, let $\langle \mathcal{T}, \mathcal{A} \rangle$ be the following ontology:

$$\mathcal{T} = \left\{ \begin{array}{rcl} \mathsf{Professor} & \sqsubseteq & \mathsf{Person}, \\ \mathsf{Student} & \sqsubseteq & \mathsf{Person}, \\ \mathsf{Person} & \sqsubseteq & \neg\mathsf{Course}, \\ \mathsf{Student} & \sqsubseteq & \neg\mathsf{Professor}, \\ \exists teaches & \sqsubseteq & \mathsf{Professor}, \\ \exists attends & \sqsubseteq & \mathsf{Student}, \\ \exists teaches^- & \sqsubseteq & \mathsf{Course}, \\ \exists attends^- & \sqsubseteq & \mathsf{Course} \end{array} \right\} \quad \mathcal{A} = \left\{ \begin{array}{rcl} \mathsf{John} & : & \mathsf{Professor} \\ \mathsf{Ava} & : & \mathsf{Student} \\ \mathsf{DB2} & : & \mathsf{Course} \\ \mathsf{KR} & : & \mathsf{Course} \\ (\mathsf{John}, \mathsf{DB2}) & : & teaches \\ (\mathsf{John}, \mathsf{KR}) & : & attends \\ (\mathsf{Ava}, \mathsf{IA}) & : & attends \\ (\mathsf{Bob}, \mathsf{KR}) & : & attends \\ \mathsf{Kevin} & : & \mathsf{Professor} \end{array} \right\}$$

$\triangleleft$

In the case of *DL-Lite*$_{\mathcal{R}}$, supporters and refuters are singletons. Thus the conflict matrix can be seen as a weighted adjacency matrix of a graph. This graph is depicted in Figure 5.3. Each node has two different pieces of information: the ABox assertion, and the quality rank computed using a stabilized bound a^* (given b, in this case $b = 1$). A red arrow from node n_1 to n_2 means that the assertion in node n_1 refutes the assertion in node n_2. A green arrow from node n_1 to n_2 means that the assertion in node n_1 supports the assertion in node n_2.

5.4. Experimental Results

We have experimentally validated the rustoner software. Rustoner consists of two parts: an interface for the ranking algorithm, and a lightweight reasoner for the *DL-Lite*$_\mathcal{R}$ logic. We first tested the ranking algorithm, which takes as input a square matrix and produces a stabilized ranking. We then tested the reasoner in rustoner on the following tasks:

- the ABox consistency check for a given ABox $\mathcal{A}$ with respect to a TBox $\mathcal{T}$;
- the construction of the conflict matrix $\mathbb{A}^f$ with respect to an ontology $\langle\mathcal{T}, \mathcal{A}\rangle$; and
- the entire procedure of producing a stabilized ranking given an ontology $\langle\mathcal{T}, \mathcal{A}\rangle$.

The remainder of this section describes our experimental setup and results. The experiments were executed on the following laptop:

```
2017 Dell XPS 15 9000 9560 Laptop: 15.6in
Intel Quad-Core i7-7700HQ
1TB SSD
16GB DDR4
NVIDIA GTX 1050
```

5.4.1 Ranking of General Matrices

We evaluated rustoner's ranking algorithm, which takes as input a zero-diagonal square matrix and computes its stabilized ranking. The input matrix is an $n \times n$ matrix $\mathbb{A}^f$ of the form given in Eq. (3.5). The matrix construction itself is not considered in this first experiment.

The algorithm will first apply the optimization induced by Proposition 3.3, which consists in removing independent assertions, as defined in Definition 3.3. Recall that if α_i is independent, then the ith row and the ith column of the input matrix are all zero. We define the *density* d as the fraction of assertions that are not independent. Thus, a density equal to 1 means that there are no independent assertions.

Example 5.10
Let A be the following matrix:

$$\begin{bmatrix} 0 & 2 & 0 & 4 \\ 1 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 \\ 3 & 5 & 0 & 0 \end{bmatrix}.$$

Rustoner will suppose that this matrix comes from an ABox

$$\mathcal{A} = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\},$$

in which the assertion α_3 is independent. The densitiy of this matrix is $\frac{3}{4}$. In a preprocessing step, rustoner will reduce this matrix to

$$\begin{bmatrix} 0 & 2 & 4 \\ 1 & 0 & 4 \\ 3 & 5 & 0 \end{bmatrix}.$$

After the ranking, the assessment for the assertion α_3 is obtained by the expression $\nu(\alpha_3) = \frac{c}{a}$ in Proposition 3.3.

$\triangleleft$

For values of n between 10 and 1000, random matrices were generated. Densities d varying from 0.1 to 1 were then obtained by making some assertions independent. Making α_i independent is tantamount to setting the ith row and the ith column equal to all zero.
To gain in robustness during the test, we performed two types of iterations:

- for each matrix, the ranking was performed an adaptive number of times to reduce CPU noise [4, 34, 58]; and
- each combination of (n, d) was tested a minimum of 50 times, and tests were repeated until the standard deviation of the execution times dropped below a given threshold.

The results of these experiments are shown in Figure 5.4. A zoom of this image for $n \leq 450$ is given in Figure 5.5.

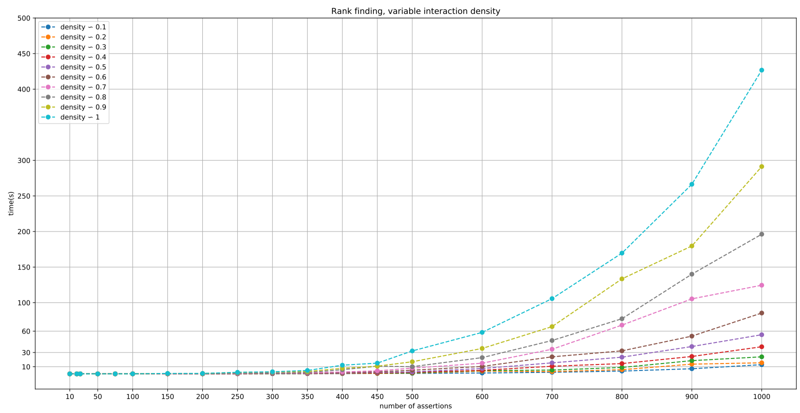

Figure 5.4: Execution time for finding a stabilized assessment in function of the number of ABox assertions.

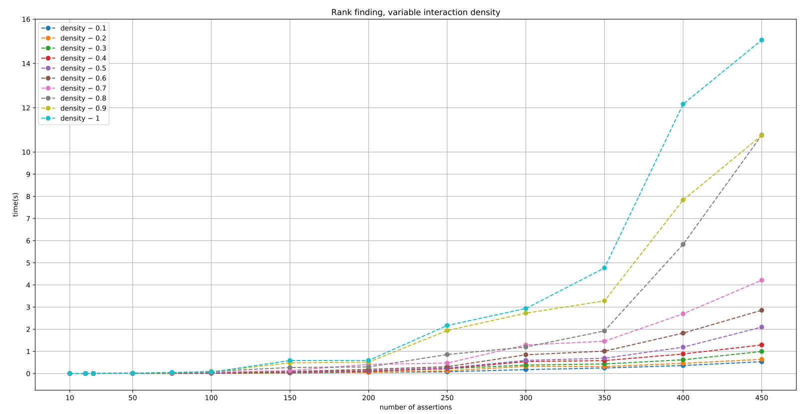

Figure 5.5: Execution time for finding a stabilized assessment in function of the number of ABox assertions (up to 450 assertions).

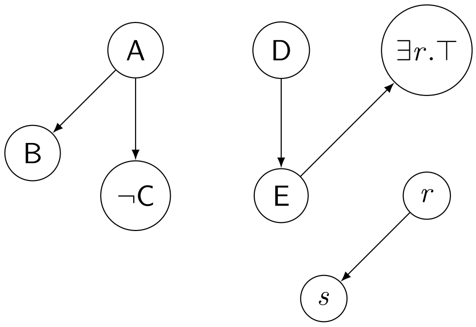

Figure 5.6: A directed graph representing a TBox.

5.4.2 Reasoner in **Rustoner**

To test the capabilities of the reasoner in rustoner, we produced synthetic ontologies, as described next. To create these synthetic ontologies, three different vocabularies where used:

- concept names where generated using as source the 1500 most used nouns in english: [5];
- role names where generated using as source the 1000 most common verbs in english: [1];
- individual names where randomly generated on demand.

Synthetic TBoxes

To generate a TBox, we followed the vision of a TBox as a directed graph [16]. For example, the directed graph of Figure 5.6 represents the TBox $\mathcal{T} = \{\mathsf{A} \sqsubseteq \mathsf{B}, \mathsf{A} \sqsubseteq \neg\mathsf{C}, \mathsf{D} \sqsubseteq \mathsf{E}, \mathsf{E} \sqsubseteq \exists r.\top, r \sqsubseteq s\}$.
To generate these graphs we used several hyperparameters:

- n_r: number of axioms involving roles present in the TBox;
- n_c: number of axioms involving concepts present in the TBox;
- d_r: the maximal length of a chain of role inclusions;

- d_c: the maximal length of a chain of concept inclusions;
- e_r: for an ordered pair of role nodes, the probability to insert a directed edge from the first node to either the second one or its negation;
- e_c: for an ordered pair of concept nodes, the probability to insert a directed edge from the first node to either the second node or its negation;
- c_r: whenever a directed edge from role r to either s or $\neg s$ is to be inserted, we pick the endpoint $\neg s$ with probability c_r, creating a negative role inclusion;
- c_c: whenever a directed edge from concept C to either D or $\neg\mathsf{D}$ is to be inserted, we pick the endpoint $\neg\mathsf{D}$ with probability c_c;
- i_r: the probability of inverting a role in a role inclusion;
- ex_c: the probability of creating a concept of the form $\exists r.\top$ instead of using a concept name in a concept inclusion.

We created TBoxes with the number of axioms in three different intervals: $[10, 20]$, $[50, 60]$, or $[100, 110]$. We changed the value of the hyperparameters to gain a variety of TBoxes for each of the three intervals.

Synthetic ABoxes

ABoxes were created relative to a TBox. For each TBox constructed with the procedure described in the preceding subsection, a family of ABoxes was created. In the following, we will use the term *interaction* for an ordered pair of assertions (α, β) such that β is present in a supporter or a refuter of α, that is, for some $B \subseteq \mathcal{A}$ we have that $I(T, B, \alpha, \beta)$ is different from zero or $I(F, B, \alpha, \beta)$ is different from zero. An interaction is *positive* if β supports α, and *negative* if β refutes α. We specify our method for creating synthetic ABoxes.
We used three hyperparameters in the case of ABoxes:

- n: number of assertions;
- i: proportion of interactions (both positive and negative); and

- c: proportion of conflicts (only negative interactions).

Note that necessarily $c \leq i$. Recall that all *DL-Lite*$_{\mathcal{R}}$ TBoxes are conflict-bounded with bound 2. Thus all refuters and supporters have cardinality equal to 1, that is, if β is involved in a positive (resp. negative) interaction with α, then $\{\beta\}$ is a supporter (resp. refuter) of α. To build an ABox, a TBox is taken as input. We first generate a random number of individual names $\mathbf{I}$ such that $\frac{n}{4} \leq |\mathbf{I}| \leq n$.
To generate the necessary number of conflicts, we iterate the next procedure: we choose randomly an ordered pair of concepts or roles such that there exists a directed path from the first assertion to the negation of the second assertion; we then choose randomly a pair of individuals from $\mathbf{I}$ if a pair of roles was selected or a single individual from $\mathbf{I}$ if a pair of concepts was selected. We build the following assertions:

- if a role pair $(r, \neg s)$ and a pair of individuals (a, b) were selected, we insert the two assertions $(\mathsf{a}, \mathsf{b}) : r$ and $(\mathsf{a}, \mathsf{b}) : s$. Note that a path from r to $\neg s$ means that $r \sqsubseteq \neg s$ is entailed by the TBox.

- if a concept pair $(\mathsf{C}, \neg\mathsf{D})$ and an individual a were selected, we insert two assertions $\mathsf{a} : \mathsf{C}$ and $\mathsf{a} : \mathsf{D}$.

This creation of conflicts terminates when the proportion c is attained. To generate the remainder of interactions, we follow the same schema, with the difference that we only consider paths from assertions (concepts or roles) to non-negated assertions.
After creating the number of assertions needed to meet the required proportion of interaction, we produce the remaining number of assertions needed to arrive at n assertions. Therefore, it remains to generate approximately $(n - \lfloor n * i \rfloor)$ assertions, which are also randomly generated. We check that no additional interactions are produced during this step by comparing with the already created assertions. For each TBox, we generated ABoxes with a density of interaction taking values in $\{0.1, 0.2, 0.5, 1.0\}$.
We comment on the results. Figure 5.7 shows that execution times for checking ABox consistency behave chaotically. Other figures are available in Appendix F. We think that this behavior is due to our implementation of the

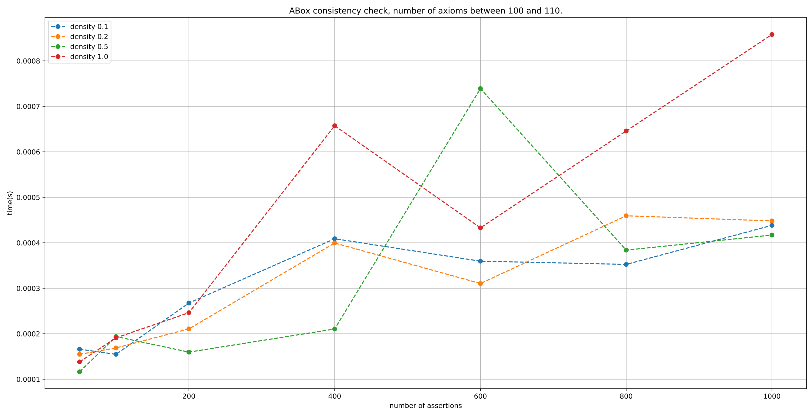

Figure 5.7: Execution time for ABox consistency check, for TBoxes with size vary between 100 and 110.

ABox consistency check, which searches for a pair of conflicting assertions until such a conflict is found. Our generation of ABoxes makes that the duration of such a search is unpredictable.
The second part of the test involves the construction of the conflict matrix $\mathbb{A}^f$ and the computation of a stabilized ranking. The results are closer to our theoretical results and our intuition: larger densities of interaction and larger ABoxes result in larger execution times. Figures 5.8 and 5.9 show that the computation of a stabilized ranking takes more time than the construction of the conflict matrix. However, this may no longer be true for description logics where supporters and refuters can be of size ≥ 1. Recall that in *DL-Lite*$_{\mathcal{R}}$, which is used in our experiments, supporters and refuters are singletons.

5.5. Conclusion

Rustoner was developed as a tool for the fast computation of quality ranks for ABox assertions. Its reasoner has been extended to a graphical tool for the exploratory analysis of *DL-Lite*$_{\mathcal{R}}$ ontologies.

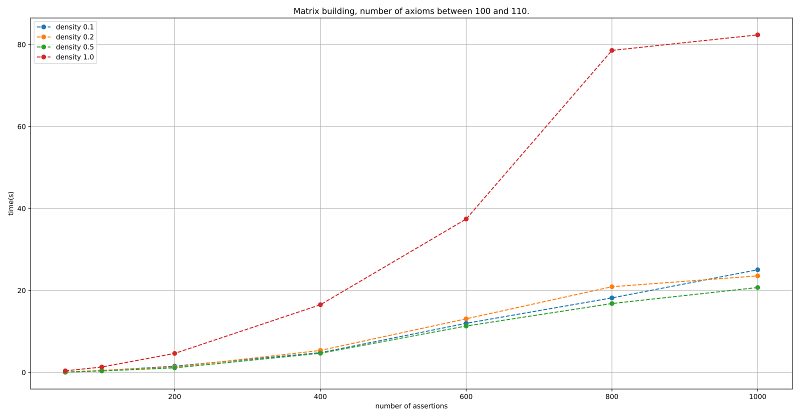

Figure 5.8: Execution time for building the conflict matrix, for TBoxes with size varying between 100 and 110.

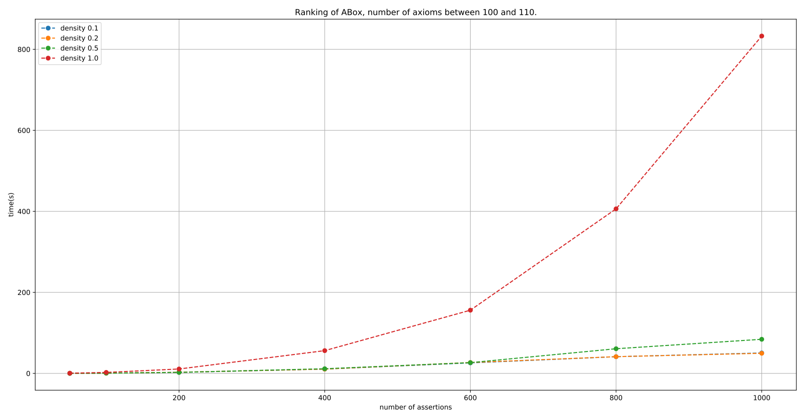

Figure 5.9: Execution time for finding a stabilized assessment, for TBoxes with size varying between 100 and 110.

In the future, we want to explore two different paths to further enhance rustoner. First, we want to speed up the computation of the stabilized bound a^* by detecting that some polynomials p_{ij} do not contribute to the final bound and hence can be omitted. Second, we want to augment rustoner with more reasoning capabilities and possibly to more powerful Description Logics languages, for example, $\mathcal{SHIQ}$.

CHAPTER 6

Conclusion

This thesis developed and investigated approaches to deal with the inconsistency problem in knowledge bases. A first contribution is a new OBDA mapping language for linking databases and Description Logics. This mapping language allows for the decidability of some desirable properties related to data quality, as stated by Theorem 2.4. Moreover, Theorem 2.5 states that the mapping language can also be used to deduce database constraints from an ontology.

The second part of this thesis developed a framework to not only quantify the quality of data in knowledge bases, but also to use this quantification in aggregate-based repairs, called $\mathcal{G}$-repairs. The ranking entailed by this quantification takes into account the user's knowledge and can be computed for ontologies defined in any DL. Theorem 3.7 states that, once a conflict matrix has been obtained, finding a *stabilized* ranking is a tractable procedure. While building the conflict matrix is a procedure that always terminates, its computational complexity depends of the Description Logic used. Significantly, Theorem 3.8 settles a tractable case of practical interest.

We defined a new notion of aggregate-based repairs, called $\mathcal{G}$-repairs, which generalizes some existing repair notions. It is assumed that database facts (or ABox assertions) are associated with weights, which can result from the quantified approach in Chapter 3. $\mathcal{G}$-repairs are then defined in terms of aggregation operators over such weighted databases. We studied the computational complexity of repair-checking and some related problems. Theorem 4.6 and Theorem 4.7 state how the complexity of these problems depend on some desirable properties of the aggregation operator used.

Finally, we presented rustoner, a program that implements the ranking procedure of Chapter 3. Rustoner also contains an exploratory tool for $DL\text{-}Lite_{\mathcal{R}}$ ontologies.

We end this thesis by listing some interesting problems for future research.

- Chapter 2 introduced an OBDA setting that allows generating an ABox from a relational database. It is an open problem to extend and combine this setting with the quantified approach developed in later chapters of this thesis.

- The ranking procedure of Chapter 3 is static, in the sense that the ABox is supposed to be fixed. It is worthwhile to study how such a ranking can be maintained while additions and deletions take place in the ABox.

- The practically important notion of *conflict-bounded TBox* in Chapter 3 is semantically defined. It would be interesting to study syntactically restricted fragments of logics that guarantee conflict-boundedness.

- Theorem 4.7 in Chapter 4 establishes that reasoning about $\mathcal{G}$-repairs quickly becomes intractable when aggregation functions are *full-combinatorial*. It is an interesting open question to develop polynomial-time approximation algorithms for such reasoning tasks.

- It would be interesting to study logical languages that allow expressing aggregation functions as a query, for example, first-order logic with aggregation [76]. It is an open question to syntactically characterize families of queries that meet the desirable semantic properties defined in Chapter 4.

- It is an open task to extend rustoner with the OBDA framework developed in Chapter 2, and to allow for more expressive description logics, for example, $\mathcal{SHIQ}$.

- In this thesis, we focused on database repairing and left open the problem of Consistent Query Answering (CQA) [118] when multiple repairs are possible. It would be interesting to combine CQA with the quantified approaches developed in this thesis.

To conclude, we believe that a quantified approach is the right answer to the *inconsistency problem* in database and knowledge base systems. Paraphrasing Lord Kelvin, *"Understanding goes through quantification."*

Appendices

APPENDIX **A**

Semantics of Relational Algebra Operators

A database $\mathbf{db}$ associates to each relation name R a finite relation over $sort(R)$. We write $R^{\mathbf{db}}$ to denote the relation associated to R by $\mathbf{db}$. For every algebra expression E, we recursively define $eval(E, \mathbf{db})$, the result of E on $\mathbf{db}$. If X is a set of attributes, then we write $\mathbf{dom}^X$ for the set of all tuples over X.

- for every relation name R, $eval(R, \mathbf{db}) = R^{\mathbf{db}}$;
- $eval(\sigma_{A=c}E, \mathbf{db}) = \{t \in eval(E, \mathbf{db}) \mid t(A) = c\}$;
- $eval(\sigma_{A=B}E, \mathbf{db}) = \{t \in eval(E, \mathbf{db}) \mid t(A) = t(B)\}$;
- $eval(\pi_X E, \mathbf{db}) = \{t[X] \mid t \in eval(E, \mathbf{db})\}$;
- if $sort(E_1) = X_1$ and $sort(E_2) = X_2$, then $eval(E_1 \bowtie E_2, \mathbf{db}) = \{t \in \mathbf{dom}^{X_1 \cup X_2} \mid t[X_1] \in eval(E_1, \mathbf{db})$ and $t[X_2] \in eval(E_2, \mathbf{db})\}$;
- if $sort(E_1) = X_1$ and $sort(E_2) = X_2$, then $eval(E_1 \ltimes E_2, \mathbf{db}) = \{t[X_1] \mid t \in \mathbf{dom}^{X_1 \cup X_2}, t[X_1] \in eval(E_1, \mathbf{db})$, and $t[X_2] \in eval(E_2, \mathbf{db})\}$;
- $eval(\delta_f E, \mathbf{db}) = \{f(t) \mid t \in eval(E, \mathbf{db})\}$;
- $eval(E_1 \cup E_2, \mathbf{db}) = eval(E_1, \mathbf{db}) \cup eval(E_2, \mathbf{db})$;
- $eval(E_1 - E_2, \mathbf{db}) = eval(E_1, \mathbf{db}) - eval(E_2, \mathbf{db})$.

APPENDIX B

Proofs for Chapter 2

B.1. Proofs of Theorem 2.1 and Corollaries 2.2 and 2.3

We use the following helping lemma. We write $\mathsf{free}(\varphi)$ for the set of free variables of a first-order formula φ.

Lemma B.1. *Let ψ and $\exists\vec{v}\varphi$ be formulas such that $\mathsf{free}(\psi) \subseteq \mathsf{free}(\exists\vec{v}\varphi)$. Then, $\psi \wedge \exists\vec{v}\varphi \equiv \exists\vec{v}\,(\varphi \wedge \psi)$.*

Proof. Since $\mathsf{free}(\psi) \subseteq \mathsf{free}(\exists\vec{v}\varphi)$, the sequence $\vec{v}$ contains no variables of $\mathsf{free}(\psi)$. Consequently, we can place ψ within the scope of $\exists\vec{v}$.

□

Proof of Theorem 2.1. We associate to each attribute A a fresh variable z_A. In our construction, if E is an Entity-expression with $sort(E) = \{A_1, \ldots, A_n\}$, then $[\![E]\!]$ will be a formula with free variables $z_{A_1}, \ldots, z_{A_n}$. The mapping $[\![\cdot]\!]$ is inductively defined as follows:

- Let $sort(R) = \{A_1, \ldots, A_n\}$, in that order. Then, $[\![R]\!] = R(z_{A_1}, \ldots, z_{A_n})$;
- $[\![\sigma_{A=c}E]\!] = [\![E]\!] \wedge (z_A = c)$;
- $[\![\sigma_{A=B}E]\!] = [\![E]\!] \wedge (z_A = z_B)$;

- $[\![\pi_X E]\!] = \exists z_{B_1} \cdots \exists z_{B_\ell} [\![E]\!]$ where $\{B_1, \dots, B_\ell\} = \mathit{sort}(E) \setminus X$;

- $[\![E_1 \ltimes E_2]\!] = [\![E_1]\!] \wedge [\![\pi_X E_2]\!]$ where $X = \mathit{sort}(E_1) \cap \mathit{sort}(E_2)$;

- $[\![\delta_{A_1,A_2,\dots,A_n \to B_1,B_2,\dots,B_n} E]\!]$ is the formula obtained from $[\![E]\!]$ by (1) renaming bound variables z_{B_i} in $[\![E]\!]$, and then (2) replacing each free occurrence of z_{A_i} with z_{B_i} (for $1 \leq i \leq n$);

- $[\![E_1 \cup E_2]\!] = [\![E_1]\!] \vee [\![E_2]\!]$; and

- $[\![E_1 - E_2]\!] = [\![E_1]\!] \wedge \neg [\![E_2]\!]$.

We show that $[\![E]\!]$ is a finite formula for every Entity-expression E. Define the weight of an Entity-expression E, denoted $\mathsf{w}(E)$, as the weighted number of algebraic operators in it, where the weight of $\ltimes$ is 2, and the weight of all other operators is 1. Since $\mathsf{w}(E_1 \ltimes E_2) = \mathsf{w}(E_1) + \mathsf{w}(E_2) + 2$ and $\mathsf{w}(\pi_X E_2) = \mathsf{w}(E_2) + 1$, it follows $\mathsf{w}(\pi_X E_2) < \mathsf{w}(E_1 \ltimes E_2)$. Then, since the function $[\![\cdot]\!]$ is applied recursively to arguments of strictly smaller weights, its computation terminates.

It is straightforward to show that for every Entity-expression E with $\mathit{sort}(E) = \{A_1, \dots, A_n\}$, $[\![E]\!]$ is a domain-independent formula $\varphi(z_{A_1}, \dots, z_{A_n})$ in relational calculus such that for every database $\mathbf{db}$, for all $a_1, \dots, a_n \in \mathbf{dom}$, $\{A_1 : a_1, \dots, A_n : a_n\} \in \mathit{eval}(E, \mathbf{db})$ if and only if $\mathbf{db} \models \varphi(a_1, \dots, a_n)$. In the remainder, we show that $[\![E]\!]$ is equivalent to a guarded formula. We note here that $[\![E]\!]$ as defined above is not automatically guarded. For example, for $\mathit{sort}(R) = \mathit{sort}(S) = \{A, B\}$ and $E = \pi_A(R \cup S)$, we obtain $[\![E]\!] = \exists z_B \, (R(z_A, z_B) \vee S(z_A, z_B))$, a formula that is not guarded.

We can write every Entity-expression in union normal form, by exhaustively applying the following rules to subexpressions until no more rules apply:

$$\begin{aligned}
\sigma_{A=c}(E \cup F) &\equiv \sigma_{A=c}E \cup \sigma_{A=c}F \\
\sigma_{A=B}(E \cup F) &\equiv \sigma_{A=B}E \cup \sigma_{A=B}F \\
\pi_X(E \cup F) &\equiv \pi_X E \cup \pi_X F \\
E \ltimes (F \cup G) &\equiv (E \ltimes F) \cup (E \ltimes G) \\
(E \cup F) \ltimes G &\equiv (E \ltimes G) \cup (F \ltimes G) \\
\delta_f(E \cup F) &\equiv \delta_f E \cup \delta_f F \\
E - (F \cup G) &\equiv (E - F) \ltimes (E - G) \\
(E \cup F) - G &\equiv (E - G) \cup (F - G)
\end{aligned}$$

In what follows, if $\vec{x}$ is a sequence of variables, then the set of the variables that occur in $\vec{x}$ is also denoted $\vec{x}$. We next show that if F is a union-free Entity-expression, then for some $k \geq 0$, $[\![F]\!]$ is equivalent to a guarded formula of the form

$$\exists v_1 \cdots \exists v_k \, (R(\vec{x}) \wedge \psi) \,, \tag{B.1}$$

with the same free variables as $[\![F]\!]$, where $R(\vec{x})$ is a relation atom, ψ is a guarded formula, and $\vec{x} \supseteq \mathsf{free}(\psi)$. The proof is by structural induction on F.

- If $F = R$, then $R(z_{A_1}, \ldots, z_{A_n}) \wedge R(z_{A_1}, \ldots, z_{A_n})$ is equivalent to $[\![F]\!]$ and has the desired form.

- Assume $F = \sigma_{A=c}E$. Thus, $[\![F]\!] = [\![E]\!] \wedge (z_A = c)$. By the induction hypothesis, we can assume

  $$[\![E]\!] \equiv \exists \vec{w} \, (S(\vec{y}) \wedge \phi) \,,$$

  a guarded formula in which ϕ is also guarded and $\vec{y} \supseteq \mathsf{free}(\phi)$. From $z_A \in \mathsf{free}([\![E]\!]) = \mathsf{free}([\![F]\!])$, it follows that z_A occurs in $\vec{y}$. Then,

  $$[\![F]\!] \equiv \exists \vec{w} \, (S(\vec{y}) \wedge (\phi \wedge z_A = c)) \,,$$

  a guarded formula of the desired form (B.1).

- Assume $F = \sigma_{A=B}E$. The reasoning is similar to the previous item.

- Assume $F = \pi_X E$. Let $X = \{A_1, \ldots, A_n\}$ and let $(sort(E) \setminus X) = \{B_1, \ldots, B_\ell\}$. Thus,

$$[\![F]\!] = \exists z_{B_1} \cdots \exists z_{B_\ell} [\![E]\!],$$

where

$$\mathsf{free}([\![E]\!]) = \{z_{A_1}, \ldots, z_{A_n}, z_{B_1}, \ldots, z_{B_\ell}\}$$

and

$$\mathsf{free}([\![F]\!]) = \{z_{A_1}, \ldots, z_{A_n}\}.$$

By the induction hypothesis, we can assume

$$[\![E]\!] \equiv \exists \vec{w} \, (S(\vec{y}) \wedge \phi) \, ,$$

a guarded formula in which ϕ is also guarded and $\vec{y} \supseteq \mathsf{free}(\phi)$. Since $z_{B_1}, \ldots, z_{B_\ell}$ are free variables of $[\![E]\!]$, they will occur in $\vec{y}$. Then,

$$[\![F]\!] \equiv \exists z_{B_1} \cdots \exists z_{B_\ell} \exists \vec{w} \, (S(\vec{y}) \wedge \phi) \, ,$$

a guarded formula of the desired form.

- Assume $F = E_1 \ltimes E_2$. Let $X = sort(E_1) \cap sort(E_2)$. Thus, $[\![E_1 \ltimes E_2]\!] = [\![E_1]\!] \wedge [\![\pi_X E_2]\!]$. By the induction hypothesis, we can assume that $[\![\pi_X E_2]\!]$ is guarded. Let $X = \{A_1, \ldots, A_n\}$. We have

$$\mathsf{free}([\![\pi_X E_2]\!]) = \{z_{A_1}, \ldots, z_{A_n}\} \subseteq \mathsf{free}([\![E_1]\!]).$$

By the induction hypothesis, we can assume

$$[\![E_1]\!] \equiv \exists \vec{w} \, (S(\vec{y}) \wedge \phi) \, ,$$

a guarded formula in which ϕ is also guarded and $\vec{y} \supseteq \mathsf{free}(\phi)$. Then,

$$[\![F]\!] \equiv \exists \vec{w} \, (S(\vec{y}) \wedge \phi) \wedge [\![\pi_X E_2]\!].$$

By Lemma B.1,

$$[\![F]\!] \equiv \exists \vec{w} \, (S(\vec{y}) \wedge (\phi \wedge [\![\pi_X E_2]\!])) \, ,$$

a guarded formula of the desired form, because $\vec{y} \supseteq \{z_{A_1}, \ldots z_{A_n}\}$.

- Assume $F = \delta_f E$. This case is obvious.
- Assume $F = E_1 - E_2$. Thus, $[\![F]\!] = [\![E_1]\!] \wedge \neg[\![E_2]\!]$ with $\mathsf{free}([\![E_1]\!]) = \mathsf{free}([\![E_2]\!])$. By the induction hypothesis, we can assume that $[\![E_2]\!]$ is guarded, hence $\neg[\![E_2]\!]$ is guarded. By the induction hypothesis, we can assume

  $$[\![E_1]\!] \equiv \exists \vec{w}\,(S(\vec{y}) \wedge \phi)\,,$$

  a guarded formula in which ϕ is also guarded and $\vec{y} \supseteq \mathsf{free}(\phi)$. Then,

  $$[\![F]\!] \equiv \exists \vec{w}\,(S(\vec{y}) \wedge \phi) \wedge \neg[\![E_2]\!].$$

  By Lemma B.1,

  $$[\![F]\!] \equiv \exists \vec{w}\,(S(\vec{y}) \wedge (\phi \wedge \neg[\![E_2]\!]))\,,$$

  a guarded formula of the desired form.

To conclude the proof, since E is in union normal form, the rule $[\![E_1 \cup E_2]\!] = [\![E_1]\!] \vee [\![E_2]\!]$ will now lead to a disjunction of guarded formulas, all of the form (B.1). Since a disjunction of guarded formulas is guarded, it follows that for every Entity-expression E, we can construct a guarded formula that, on all database instances, returns the same answers as E. This concludes the proof. $\square$

Proof of Corollary 2.2. Let E be an Entity-expression. Then, $\pi_{\{\}}E$ is an Entity-expression which returns either $\emptyset$ or $\{\{\}\}$ (a singleton containing the empty tuple), interpreted as **false** and **true**, respectively. Note that

$$eval(E, \mathbf{db}) = \emptyset$$

if and only if $eval(\big(\pi_{\{\}}E\big), \mathbf{db}) = \emptyset$. By Theorem 2.1, $\pi_{\{\}}E$ can be translated in a Boolean guarded formula φ, with constants, such that E and φ agree on all database instances. Let C be the set of constant symbols that occur in E, and let

$$\psi := \bigwedge_{\substack{a, b \in C \\ a \neq b}} \neg(a = b).$$

Then, $\varphi \wedge \psi$ is a guarded formula, and $eval(E, \mathbf{db}) \neq \emptyset$ for some database **db** if and only if $\varphi \wedge \psi$ is satisfiable. The desired results follows, because *(i)* satisfiability of guarded formulas with constant symbols is decidable [111], and *(ii)* the guarded fragment has the finite model property. The latter property is important, because a database is a finite model. The role of ψ is to enforce a Herbrand interpretation for constant symbols, as is common in database theory.

To illustrate the role of ψ, consider $E = \pi_{\{\}}(\sigma_{A=0}(\sigma_{A=1}R))$ with $sort(R) = \{A\}$, which agrees with $\varphi = \exists x\,(R(x) \wedge x = 0 \wedge x = 1)$ on all database instances. In fact, φ is **false** on all database instances. Note, however, that φ is satisfied by the structure $\mathfrak{A}$ with $R^{\mathfrak{A}} = \{\langle a \rangle\}$ and $0^{\mathfrak{A}} = 1^{\mathfrak{A}} = a$. The latter structure, however, is not a model of $\neg(0 = 1) \wedge \varphi$.

□

Proof of Corollary 2.3. Every Relationship-expression combines Entity-expressions by means of the operators $\sigma_{A=c}$, $\sigma_{A=B}$, $\bowtie$, δ_f, $\cup$, and $-$. These operators are projection-free, and hence their translation in first-order logic introduces no quantifiers. By Theorem 2.1, for every Entity-expression E with $sort(E) = \{A_1, \ldots, A_n\}$, we can compute a formula $\varphi(z_{A_1}, \ldots, z_{A_n})$ in *GF* that is logically equivalent to E. If we combine such formulas without introducing quantifiers, the result will belong to *GF*.

Let E be a Relationship-expression. By what precedes, we can construct a formula φ in *GF* that is logically equivalent to E. As in the proof of Corollary 2.2, the closed formula

$$\varphi \wedge \bigwedge_{\substack{a, b \in C \\ a \neq b}} \neg(a = b),$$

where C is the set of constant symbols in E, is guarded and ensures that different constant symbols are interpreted by different values, as is common in database theory. The desired result follows, because *(i)* satisfiability of guarded formulas is decidable, also in the presence of constant symbols [111], and *(ii)* the guarded fragment has the finite model property. The latter property is important, because a database is a finite model. This concludes the proof.

∎

B.2. Proof of Theorem 2.4

We will use two helping lemmas. The following helping lemma shows that the set of role assertions generated by an RDAD can be obtained by a single algebra expression.

Lemma B.2. *Let $\rho \coloneqq [E_1/f_1, E_2/f_2, E] : r$ be a (not necessarily join-free) RDAD. Let $X_1 = sort(E_1)$ and $X_2 = sort(E_2)$. Define $F \coloneqq (E \ltimes \delta_{f_1} E_1) \ltimes \delta_{f_2} E_2$. Then, for every database* $\mathbf{db}$*, the set of role assertions generated by ρ from* $\mathbf{db}$ *is*

$$\{ \ \left(\iota \circ {f_1}^{-1} \left(t[f_1 \left(X_1\right)]\right), \ \ \iota \circ {f_2}^{-1} \left(t[f_2 \left(X_2\right)]\right)\right) : r \ \ \mid t \in eval(F, \mathbf{db}) \ \},$$

where $\circ$ is function composition.

Proof. We show equality with the set defined by Definition 2.5.
$\boxed{\subseteq}$ Assume $t_1 \in eval(E_1, \mathbf{db})$, $t_2 \in eval(E_2, \mathbf{db})$, and $f_1(t_1) \cup f_2(t_2) \subseteq t$ for some $t \in eval(E, \mathbf{db})$. Then, $t \in eval(F, \mathbf{db})$. We have $f_1\left(t_1\right) = t[f_1\left(X_1\right)]$, thus $t_1 = {f_1}^{-1}\left(t[f_1\left(X_1\right)]\right)$. Likewise for the second position.
$\boxed{\supseteq}$ Assume $t \in eval(F, \mathbf{db})$, hence $t \in eval(E, \mathbf{db})$. Then,

$${f_1}^{-1}\left(t[f_1\left(X_1\right)]\right) \in eval(E_1, \mathbf{db}).$$

Thus, there exists $t_1 \in eval(E_1, \mathbf{db})$ such that

$$t_1 = {f_1}^{-1}\left(t[f_1\left(X_1\right)]\right).$$

∎

The following helping lemma states a useful syntactic simplification: we can assume that all Entity-expressions and Relationship-expressions that occur in CDADs and RDADs are union-free.

Lemma B.3. *For every set $\mathcal{M}$ of CDADs and (not necessarily join-free) RDADs, there exists a set $\mathcal{M}'$ such that*

1. *for every CDAD $E : C$ in $\mathcal{M}'$, we have that E is union-free;*

2. *for every RDAD* $[E_1/f_1, E_2/f_2, E] : r$ *in* $\mathcal{M}'$*, we have that* E_1*,* E_2*, and* E *are union-free; and*

3. *for every database* $\mathbf{db}$*,* $\mathcal{M}(\mathbf{db}) = \mathcal{M}'(\mathbf{db})$*.*

Proof. Obviously,

$$\begin{aligned} E \bowtie (F \cup G) &\equiv (E \bowtie F) \cup (E \bowtie G) \\ (E \cup F) \bowtie G &\equiv (E \bowtie G) \cup (F \bowtie G) \end{aligned}$$

Let $[E/f, F/g, H] : r$ be an RDAD in $\mathcal{M}$. Using the two previous equivalences together with those in the proof of Theorem 2.1, we can rewrite $E \equiv E_1 \cup E_2 \cup \cdots \cup E_e$ where each E_i is union-free. Likewise, $F \equiv F_1 \cup F_2 \cup \cdots \cup F_f$ and $H \equiv H_1 \cup H_2 \cup \cdots \cup H_h$. Now replace, in $\mathcal{M}$, the RDAD $[E/f, F/g, H] : r$ with all RDADs

$$[E_i/f, F_j/g, H_k] : r \qquad \text{for all } 1 \leq i \leq e,\ \ 1 \leq j \leq f,\ \ 1 \leq k \leq h$$

It is obvious that on every $\mathbf{db}$, this replacement does not change the set of role assertions generated. It is even simpler to make CDADs union-free.

□

Proof of Theorem 2.4. In Section 2.4, we assumed a total order $\leq_{\mathbf{att}}$ on the set $\mathbf{att}$ of attributes. This order is commonly used in database theory to switch between different representations for tuples (see, e.g., [6, p. 32]): When we list a tuple $\{A_1 : a_1, \ldots, A_k : a_k\}$, it is assumed that the attributes are written according to $\leq_{\mathbf{att}}$, i.e., $A_1 \leq_{\mathbf{att}} A_2 \leq_{\mathbf{att}} \cdots$. Then, this tuple can also be represented as the *ordered tuple* $s = (a_1, \ldots, a_k)$ *over* $\{A_1, \ldots, A_k\}$, and we denote each a_i as $s(A_i)$. For a technical reason that will become apparent shortly, for all sets S, U such that $S \subseteq U \subseteq \mathbf{att}$, we assume a function $\mathsf{cast}_{S \mapsto U}$ that maps ordered tuples over S to ordered tuples over U, as follows: for every ordered tuple s over S, $\mathsf{cast}_{S \mapsto U}(s)$ is the ordered tuple u over U such that s and u agree on all attributes of S, and $u(A) = \varepsilon$ for every $A \in U \setminus S$, where ε is a fixed fresh constant. For example, if s is the ordered tuple (a, c) over $\{A, C\}$ and $U = \{A, B, C, D\}$, then $\mathsf{cast}_{\{A,C\} \mapsto U}(s) = (a, \varepsilon, c, \varepsilon)$, an ordered tuple over U. Clearly, such a function $\mathsf{cast}_{S \mapsto U}$ is injective.

Let

$$U := \left(\bigcup_{E:C \in \mathcal{M}} sort(E) \right) \cup \left(\bigcup_{[E_1/f_1, E_2/f_2, E]:r \in \mathcal{M}} sort(E_1) \cup sort(E_2) \right).$$

The set U contains all attributes that are used to create individual names. For example, let $U = \{\mathit{Lastname}, \mathit{Firstname}, \mathit{Hotel}, \mathit{City}\}$. Let $s = (\mathtt{Hilton}, \mathtt{Paris})$ be an ordered tuple. Then,

$$\mathsf{cast}_{\{\mathit{Lastname}, \mathit{Firstname}\} \mapsto U}(s) = (\mathtt{Hilton}, \mathtt{Paris}, \varepsilon, \varepsilon)$$

has the same arity, but is distinct from

$$\mathsf{cast}_{\{\mathit{Hotel}, \mathit{City}\} \mapsto U}(s) = (\varepsilon, \varepsilon, \mathtt{Hilton}, \mathtt{Paris}).$$

Since ordered tuples over U now one-to-one correspond to individual names, we can assume that

$$\iota = \bigcup_{S \subseteq U} \mathsf{cast}_{S \mapsto U},$$

which defines an injective mapping.
From here on, we use m for $|U|$. By our hypothesis, there exists a guarded first-order formula $\varphi_{\mathcal{T}}$ that is equivalent to $\mathcal{T}$. Let $\varphi_{\mathcal{T},m}$ be the formula obtained from $\varphi_{\mathcal{T}}$ by replacing all occurrences of every variable x with $x_1, \ldots, x_m$. It can be seen that $\varphi_{\mathcal{T},m}$ will be guarded. For example, if

$$\varphi_{\mathcal{T}} = \forall x \forall y \left(r(x,y) \to \left(\left(\forall z \left(r(x,z) \to A(z) \right) \right) \to B(x) \right) \right),$$

then

$$\varphi_{\mathcal{T},2} = \forall x_1, x_2 \forall y_1, y_2 \left(\begin{array}{l} r(x_1, x_2, y_1, y_2) \to \\ \left(\left(\forall z_1, z_2 \left(r(x_1, x_2, z_1, z_2) \to A(z_1, z_2) \right) \right) \to B(x_1, x_2) \right) \end{array} \right).$$

Note that unary and binary predicates in $\varphi_{\mathcal{T}}$ become, respectively, m-ary and $2m$-ary in $\varphi_{\mathcal{T},m}$. Furthermore, it is understood that $x_1, \ldots, x_m = y_1, \ldots, y_m$ is a shorthand for $\bigwedge_{i=1}^{m} x_i = y_1$.
We now show how a join-free RDAD $\sigma = [E_1/f_1, E_2/f_2, E] : r$ is expressed in *GF*. By Lemma B.3, we can assume without loss of generality that E_1, E_2, and E are union-free. Let $F = (E \ltimes \delta_{f_1} E_1) \ltimes \delta_{f_2} E_2$. Since E is a join-free

Relationship-expression, it follows that F is an Entity-expression, which will be union-free. Following the proof of Theorem 2.1 and using Lemma B.2, F can be translated into an equivalent guarded formula

$$\phi_\sigma(\vec{x}_1, \vec{x}_2, \vec{x}_3) = \exists \vec{y}\,(S(\vec{x}_1, \vec{x}_2, \vec{x}_3, \vec{y}) \wedge \psi(\vec{x}_1, \vec{x}_2, \vec{x}_3, \vec{y}))\,,$$

where $\vec{x}_1$ is an ordered tuple over $sort(\delta_{f_1} E_1)$, $\vec{x}_2$ is an ordered tuple over $sort(\delta_{f_2} E_2)$, and $\vec{x}_3$ is an ordered tuple over $(sort(E) \setminus sort(\delta_{f_1} E_1)) \setminus sort(\delta_{f_2} E_2)$. That is, if we let $X_1 = sort(E_1)$ and $X_2 = sort(E_2)$, then the following are equivalent for all $\vec{c}_1 \in \mathbf{dom}^{|\vec{x}_1|}$, $\vec{c}_2 \in \mathbf{dom}^{|\vec{x}_2|}$:

- $\mathbf{db} \models \phi_\sigma(\vec{c}_1, \vec{c}_2, \vec{c}_3)$ for some $\vec{c}_3 \in \mathbf{dom}^{|\vec{x}_3|}$;
- the role assertion $(\mathsf{cast}_{X_1 \mapsto U}(\vec{c}_1), \mathsf{cast}_{X_2 \mapsto U}(\vec{c}_2)) : r$ is generated by σ from $\mathbf{db}$. Note incidentally that in this cast operator, $\vec{c}_1$ is considered as an ordered tuple over X_1, and $\vec{c}_2$ as an ordered tuple over X_2.

The following closed formula φ_σ in *GF* captures the semantics of the RDAD σ:

$$\varphi_\sigma = \forall \vec{x}_1 \forall \vec{x}_2 \forall \vec{x}_3 \forall \vec{y} \left(\begin{array}{l} S(\vec{x}_1, \vec{x}_2, \vec{x}_3, \vec{y}) \rightarrow \\ (\psi(\vec{x}_1, \vec{x}_2, \vec{x}_3, \vec{y}) \rightarrow r(\mathsf{cast}_{X_1 \mapsto U}(\vec{x}_1), \mathsf{cast}_{X_2 \mapsto U}(\vec{x}_2))) \end{array} \right).$$

Likewise, for every CDAD $C : E$ with $sort(E) = X$ and E union-free, we can construct a guarded formula

$$\phi_{C:E}(\vec{x}) = \exists \vec{y}\,(S(\vec{x}, \vec{y}) \wedge \psi(\vec{x}, \vec{y}))\,,$$

where $\vec{x}$ is an ordered tuple over X, such that the following are equivalent for every $\vec{c} \in \mathbf{dom}^{|\vec{x}|}$:

- $\mathbf{db} \models \phi_{C:E}(\vec{c})$;
- $\mathsf{cast}_{X \mapsto U}(\vec{c}) : C$ is generated by $C : E$ from $\mathbf{db}$.

The following closed formula φ_σ in *GF* captures the semantics of the CDAD $C : E$:

$$\varphi_{C:E} = \forall \vec{x} \forall \vec{y}\,(S(\vec{x}, \vec{y}) \rightarrow (\psi(\vec{x}, \vec{y}) \rightarrow C(\mathsf{cast}_{X \mapsto U}(\vec{x})))\,.$$

Now define

$$
\begin{aligned}
\Gamma_1 &\coloneqq \varphi_{\mathcal{T},m} \wedge \left(\bigwedge_{\sigma\in\mathcal{M}} \varphi_\sigma\right) \wedge \left(\bigwedge_{\sigma\in\Sigma} \sigma\right)\\
\Gamma_2 &\coloneqq \varphi_{\mathcal{T},m} \wedge \left(\bigwedge_{\sigma\in\mathcal{M}} \varphi_\sigma\right) \wedge \neg\left(\bigwedge_{\sigma\in\Sigma} \sigma\right)\\
\Gamma_3 &\coloneqq \neg\varphi_{\mathcal{T},m} \wedge \left(\bigwedge_{\sigma\in\mathcal{M}} \varphi_\sigma\right) \wedge \left(\bigwedge_{\sigma\in\Sigma} \sigma\right)
\end{aligned}
$$

which are three closed formulas in GF. Note that σ ranges over all CDADs and RDADs in $\mathcal{M}$. By construction, we obtain the following equivalences:

- $(\Sigma, \mathcal{M}, \mathcal{T})$ is a "yes"-instance to SATISFIABILITY if and only if Γ_1 is satisfiable.
- $(\Sigma, \mathcal{M}, \mathcal{T})$ is a "yes"-instance to NON-FATHFULNESS if and only if Γ_2 is satisfiable.
- $(\Sigma, \mathcal{M}, \mathcal{T})$ is a "yes"-instance to NON-PROTECTION if and only if Γ_3 is satisfiable.

Since satisfiability of guarded formulas is decidable, the desired result obtains. For GLOBAL-CONSISTENCY, for every $E : C$ in $\mathcal{M}$, let $\psi_{E:C}$ be the closed first-order formula that is logically equivalent to $\pi_{\{\}}E$. For every RDAD $\sigma = [E_1/f_1, E_2/f_2, E] : r$ in $\mathcal{M}$, let ψ_σ be be the closed first-order formula that is logically equivalent to $\pi_{\{\}}((E \ltimes \delta_{f_1}E_1) \ltimes \delta_{f_2}E_2)$. By Theorem 2.1 and since all RDADs in $\mathcal{M}$ are join-free, if σ is a CDAD or an RDAD in $\mathcal{M}$, then ψ_σ is expressible in GF. Then, GLOBAL-CONSISTENCY is equivalent to satisfiability of the following closed formula in GF:

$$
\Gamma_4 \coloneqq \varphi_{\mathcal{T},m} \wedge \left(\bigwedge_{\sigma\in\mathcal{M}} \varphi_\sigma \wedge \psi_\sigma\right) \wedge \left(\bigwedge_{\sigma\in\Sigma} \sigma\right)
$$

This concludes the proof. □

B.3. Proof of Theorem 2.5

Proof of Theorem 2.5 (Sketch). We show the construction of Σ'. From [31], it follows that unsatisfiability in $DL\text{-}Lite_{core}$ can only arise due to some negative inclusion $C \sqsubseteq \neg D$ implied by the TBox that is violated in the ABox. The negative inclusion $C \sqsubseteq \neg D$ can be of four different forms, where A, B denote concept names, and r, s role names:

- $A \sqsubseteq \neg B$. Then, for all CDADs $E : A$ and $F : B$ in $\mathcal{M}$ such that $sort(E) = sort(F)$, Σ' contains a formula stating emptiness of $\pi_{\{\}}(E \ltimes F)$.
- $A \sqsubseteq \neg\exists r$. Then, for every CDAD $E : A$ and RDAD $[F_1/g, F_2/g', F] : r$ in $\mathcal{M}$ such that $sort(E) = sort(F_1)$, Σ' contains a formula stating emptiness of $\pi_{\{\}}\big(E \ltimes \delta_{g^{-1}}\big(\delta_g(F_1) \ltimes \big(F \ltimes \delta_{g'}F_2\big)\big)\big)$.
- $\exists r \sqsubseteq \neg A$. This is equivalent to $A \sqsubseteq \neg\exists r$, which is of the form in the previous item.
- $\exists r \sqsubseteq \neg\exists s$. Then, for all RDADs $[E_1/f, E_2/f', E] : r$ and $[F_1/g, F_2/g', F] : s$ in $\mathcal{M}$ such that $sort(E_1) = sort(F_1)$, Σ' contains a formula stating emptiness of

$$\pi_{\{\}}\big(\delta_{f^{-1}}\big(\delta_f(E_1) \ltimes \big(E \ltimes \delta_{f'}E_2\big)\big) \ltimes \delta_{g^{-1}}\big(\delta_g(F_1) \ltimes \big(F \ltimes \delta_{g'}F_2\big)\big)\big).$$

From the construction, it follows that for every database $\mathbf{db}$, $\mathbf{db} \models \Sigma'$ if and only if $(\mathcal{T}, \mathcal{M}(\mathbf{db}))$ is a consistent knowledge base.
Finally, if all RDADs are join-free, then all previous algebra expressions are actually Entity-expressions, and thus, by Theorem 2.1, can be expressed in GF.

$\square$

APPENDIX C

Background from Algebra

We recall some notions of linear algebra [14, 72] that are used in this thesis, particularly in Chapters 3 and 5.

Vector Spaces

Vector spaces over $\mathbb{R}$ are a building block for linear algebra. The vector space $\mathbb{R}^n$, with *dimension* n, is used in our study of inconsistencies. Operators include coordinate-wise sum and multiplication by a scalar. Although *finite* vector spaces over the real numbers can take several forms, it can be shown that, modulo isomorphism, they are all equal to $\mathbb{R}^n$ for some natural number n.

Norms Norms can be seen as functions that measure the size of the elements in a vector space, or as functions that output the distance between an element and the null vector. A function

$$\|.\| : \mathbb{R}^n \to \mathbb{R}^{\geq 0}$$

is a *norm* over $\mathbb{R}^n$ if it satisfies the following three properties:

- $\|v\| = 0$ if and only if $v = 0$, the null vector of $\mathbb{R}^n$;
- for all v in $\mathbb{R}^n$ and all $t \in \mathbb{R}$, we have that $\|t * v\| = |t| * \|v\|$; and

- *Triangular inequality:* for all v, u in $\mathbb{R}^n$ we have that $\|v+u\| \leq \|v\|+\|u\|$.

Let $x = (x_1, \ldots, x_n)$ be a vector in $\mathbb{R}^n$. Common norms are the following:

- 1-norm $\|x\|_1 := \sum_{i=1}^{n} |x_i|$;
- 2-norm or euclidean distance $\|x\|_2 := \left(\sum_{i=1}^{n} |x_i|^2\right)^{\frac{1}{2}}$; and
- infinity norm $\|x\|_\infty := \max_{1 \leq i \leq n} |x_i|$.

Linear Operators For vector spaces $\mathbb{R}^n$ and $\mathbb{R}^m$, a *linear operator* is any function $f : \mathbb{R}^n \to \mathbb{R}^m$ such that for all u, v in $\mathbb{R}^n$ and all t in $\mathbb{R}$, we have $f(t * u + v) = t * f(u) + f(v)$. A function $g : \mathbb{R}^n \to \mathbb{R}^m$ is *affine* if there exists a linear operator $f : \mathbb{R}^n \to \mathbb{R}^m$ and $u_0 \in \mathbb{R}^m$ such that for all v in $\mathbb{R}^n$ we have that $g(v) = u_0 + f(v)$.

Matrices

Let n, m be two natural numbers. The space of matrices of dimension $n \times m$ with coefficients in $\mathbb{R}$ is not only a vector space, but also a representation of linear operators from $\mathbb{R}^n$ to $\mathbb{R}^m$. We illustrate this by two examples.

Matrix Multiplication If A is a matrix of dimension $n \times m$, and B is a matrix of dimension $m \times p$, then their multiplication AB is a matrix of dimension $n \times p$. This multiplication coincides with the composition of two linear operators: A is a linear operator from $\mathbb{R}^n$ to $\mathbb{R}^m$, B is a linear operator from $\mathbb{R}^m$ to $\mathbb{R}^p$, and AB is the linear operator from $\mathbb{R}^n$ to $\mathbb{R}^p$ that is the composition of A and B.
In the case that $n = m$, the matrix A is called *square* and is a function from $\mathbb{R}^n$ to itself. We write $\mathbb{1}$ for the square matrix A that is the identity function over $\mathbb{R}^n$.

Linear Systems A linear system of equations with m unknowns $x_1, \ldots, x_m$ and n equations, contains, for every $i \in \{1, 2, \ldots, n\}$, an equality

$$\sum_{1 \leq j \leq m} a_{ij} x_j = b_j.$$

This system can be represented as a matrix equation $AX = b$ where A is a matrix of dimension $n \times m$ whose (i,j)-th coordinate is a_{ij}, and b is a vector of dimension n whose j-th coordinate is b_j.

A linear system has a *unique* solution if and only if the matrix A that defines the equation is *invertible*. A matrix is invertible if there exists another matrix, denoted A^{-1}, such that $AA^{-1} = A^{-1}A = \mathbb{1}$. This is only possible for square matrices, in which case the solution X is given by

$$X = A^{-1}b.$$

Determinant The determinant of a square matrix, denoted $\det(A)$, is a real number. In this thesis, we use the following properties of determinants:

- the determinant $\det(A)$ of a matrix A is non-zero if and only A is invertible;

- *Cramer's Rule:* if $AX = b$ is a matrix equation, and A is invertible, then the solution vector is given by the following expression:

  $$X = (x_1, \ldots, x_n) = \frac{1}{\det(A)}(\det(A_1), \ldots, \det(A_n)),$$

  where A_i is the matrix obtained from A by replacing the i-th column with the vector b;

- if f is an *affine* function from the real numbers $\mathbb{R}$ to the space of matrices of dimension $n \times n$, then the expression $\det(f(a))$ is a polynomial in a for every a in $\mathbb{R}$.

It should be noted that vector spaces of matrices can also be equipped with norms. We will use the following result that relates such norms to linear systems.

Theorem C.1 (Banach's Lemma)**.** *Let M be a square matrix with the property that $\|M\| < 1$ for some operator norm $\|\cdot\|$. Then the matrix $\mathbb{1} - M$ is invertible and*

$$(\mathbb{1} - M)^{-1} = \sum_{n \geq 0} M^n.$$

This result is also known as *Neumann's Lemma*.

APPENDIX D

Proofs for Chapter 3

Proof of Proposition 3.1. The proof of the first two items is straightforward. We next prove the last item. Assume B is a refuter of α, and B' is a supporter of α. Consequently,

(a) $\langle \mathcal{T}, B \rangle$ is consistent;

(b) $\langle \mathcal{T}, B \cup \{\alpha\} \rangle$ is inconsistent;

(c) $\langle \mathcal{T}, B' \rangle$ is consistent; and

(d) $\langle \mathcal{T}, B' \rangle \models \alpha$.

From (c) and (d), it follows $\langle \mathcal{T}, B' \cup \{\alpha\} \rangle$ is consistent.
We first show $B \not\subseteq B'$. Assume for a contradiction that $B \subseteq B'$, hence $B \cup \{\alpha\} \subseteq B' \cup \{\alpha\}$. From (b) and our assumption that the underlying Description Logic is monotonic, it follows that $\langle \mathcal{T}, B' \cup \{\alpha\} \rangle$ is inconsistent, a contradiction.
Finally, we show $B' \not\subseteq B$. Assume for a contradiction $B' \subseteq B$. From (d) and our assumption that the underlying Description Logic is monotonic, it follows $\langle \mathcal{T}, B \rangle \models \alpha$. By (a), it follows $\langle \mathcal{T}, B \cup \{\alpha\} \rangle$ is consistent, which contradicts (b).

$\square$

Proof of Proposition 3.2. First note that both $\mathcal{A}$ and $\mathcal{A}'$ denote the same set, thus the behavior of f over $\mathcal{A}'$ is identical to the behavior of f over $\mathcal{A}$. Let $1 \leq i, j \leq n$, we have that

$$\begin{aligned}
\mathbb{A}^f_{\rho(i),\rho(j)} &= \sum_{B \subseteq \mathcal{A}} f(B) * \left(I(F, B, \alpha_{\rho(i)}, \alpha_{\rho(j)}) - I(F, B, \alpha_{\rho(i)}, \alpha_{\rho(j)})\right) \\
&= \left(I(F, B, \beta_i, \beta_j) - I(F, B, \beta_i, \beta_j)\right) \\
&= \mathbb{A}'^f_{i,j}.
\end{aligned}$$

The first assertion of the proposition is thus proved.
Now let P_ρ be the function from the space of square matrices $n \times n$ to itself such that for any matrix A we have that $P_\rho(A)_{i,j} = A_{\rho(i),\rho(j)}$ for all $1 \leq i, j \leq n$. It is known that ρ being a permutation implies that P_ρ is a linear bijection that fixes the identity matrix and that is also a multiplicative morphism, that is, for all square matrices A, B we have that $P_\rho(AB) = P_\rho(A)P_\rho(B)$. Moreover, $P_\rho(\mathbb{A}^f) = \mathbb{A}'^f$.
Now let (a, b, c) be any triple of reals and let $\left(a \cdot \mathbb{1} - b \cdot \mathbb{A}^f\right)$ be the system matrix. We obtain that

$$P_\rho\left(a \cdot \mathbb{1} - b \cdot \mathbb{A}^f\right) = a \cdot P_\rho(\mathbb{1}) - b \cdot P_\rho(\mathbb{A}^f) = a \cdot \mathbb{1} - b \cdot \mathbb{A}'^f.$$

From the properties of P_ρ, it follows that $\left(a \cdot \mathbb{1} - b \cdot \mathbb{A}^f\right)$ is invertible if and only if $\left(a \cdot \mathbb{1} - b \cdot \mathbb{A}'^f\right)$ is invertible. Indeed, the existence of a multiplicative inverse for $\left(a \cdot \mathbb{1} - b \cdot \mathbb{A}^f\right)$ is equivalent to the existence of a multiplicative inverse for $\left(a \cdot \mathbb{1} - b \cdot \mathbb{A}'^f\right)$.
Suppose now that $\left(a \cdot \mathbb{1} - b \cdot \mathbb{A}^f\right)$ is invertible. Then the assessment ν is uniquely defined by the vector

$$x = \left(a \cdot \mathbb{1} - b \cdot \mathbb{A}^f\right)^{-1} \cdot \begin{bmatrix} c & c & \cdots & c \end{bmatrix}^\intercal$$

and the assessment ν' is uniquely defined by the vector

$$x' = \left(a \cdot \mathbb{1} - b \cdot \mathbb{A}'^f\right)^{-1} \cdot \begin{bmatrix} c & c & \cdots & c \end{bmatrix}^\intercal$$

We get that

$$
\begin{aligned}
x'_i &= \sum_{j=1}^{n} \left(a\cdot\mathbb{1} - b\cdot \mathbb{A}'^{f}\right)^{-1}_{i,j} * c = \sum_{j=1}^{n} \left(P_\rho\left(a\cdot\mathbb{1} - b\cdot\mathbb{A}^{f}\right)\right)^{-1}_{i,j} * c \\
&= \sum_{j=1}^{n} P_\rho\left(\left(a\cdot\mathbb{1} - b\cdot\mathbb{A}^{f}\right)^{-1}\right)_{i,j} * c = \sum_{j=1}^{n}\left(a\cdot\mathbb{1} - b\cdot\mathbb{A}^{f}\right)^{-1}_{\rho(i),\rho(j)} * c \\
&= x_{\rho(i)}
\end{aligned}
$$

and the last assertion of the proposition holds.

□

Proof of Proposition 3.3. Let α be an independent assertion in $\mathcal{A}$. By Proposition 3.2, we can suppose that $\alpha = \alpha_n$ without lost of generality. By definition of ν and α_n being independent, we get that

$$
\begin{aligned}
\nu(\alpha_n) &= \frac{c}{a} + \frac{c}{b}\sum_{B\subseteq\mathcal{A}}\left(f(B) * \sum_{\beta\in\mathcal{A}}\nu(\beta) * \left(I(T,B,\alpha_n,\beta) - I(F,B,\alpha_n,\beta)\right)\right) \\
&= \frac{c}{a}\sum_{B\subseteq\mathcal{A}}\left(f(B) * \sum_{\beta\in\mathcal{A}}\nu(\beta) * 0\right) \\
&= \frac{c}{a}.
\end{aligned}
$$

Let $\mathbb{A}^f$ be the conflict matrix relative to $\langle\mathcal{T},\mathcal{A}\rangle$ and f. Let (a,b,c) be a triple of real numbers, and let $A = \left(a\cdot\mathbb{1} - b\cdot\mathbb{A}^f\right)$. Since α_n is independent, the following equations hold for all i distinct from n:

$$
A_{i,n} = -b * \sum_{B\subseteq\mathcal{A}} f(B) * \left(I(T,B,\alpha_i,\alpha_n) - I(F,B,\alpha_i,\alpha_n)\right) = 0
$$

and

$$
A_{n,i} = -b * \sum_{B\subseteq\mathcal{A}} f(B) * \left(I(T,B,\alpha_n,\alpha_i) - I(F,B,\alpha_n,\alpha_i)\right) = 0
$$

In the same way, let $A' = \left(a\cdot\mathbb{1} - b\cdot\mathbb{A}'^{f}\right)$ where $\mathbb{A}'^{f}$ is the conflict matrix relative to $\langle\mathcal{T},\mathcal{A}\backslash\{\alpha\}\rangle$. Let Γ be the function that to every square matrix A

of dimension $(n-1)\times(n-1)$ associates the following matrix:

$$\Gamma(A)_{i,j} = \begin{cases} A_{i,j} \text{ if } i \leq n-1 \text{ and } j \leq n-1; \\ A_{n,n} = a; \\ 0 \text{ otherwise.} \end{cases}$$

That is, Γ is an injection from the space of square matrices $(n-1)\times(n-1)$ to the space of square matrices $n\times n$. It is known that Γ is a linear bijection that is also a multiplicative morphism, and that its image is contained in the following subspace of matrices:

$$S = \{A \text{ matrix of dimension } n \times n \mid \forall i \neq n, A_{i,n} = A_{n,i} = 0\}.$$

Since $\Gamma(A') = A$, it holds that A is invertible if and only if A' is invertible. Consequently, the triple (a,b,c) produces a unique assessment for $\langle \mathcal{T}, \mathcal{A}\rangle$ if and only if it produces a unique assessment for $\langle \mathcal{T}, \mathcal{A}\backslash\{\alpha_n\}\rangle$. It is now correct to conclude that the first affirmation of the proposition holds.
Let $\alpha_i \in \mathcal{A}\backslash\{\alpha_n\}$ be an assertion. Then,

$$\begin{aligned}
\nu(\alpha_i) &= \left(\left(a\cdot\mathbb{1} - b\cdot\mathbb{A}^f\right)^{-1}\begin{bmatrix} c & c & \cdots & c\end{bmatrix}^\intercal\right)_i \\
&= \sum_{j=1}^{n} A_{i,j} * c = \sum_{j=1}^{n-1} A_{i,j} * c = \sum_{j=1}^{n-1} A'_{i,j} * c \\
&= \left(\left(a\cdot\mathbb{1} - b\cdot\mathbb{A}'^f\right)^{-1}\begin{bmatrix} c & c & \cdots & c\end{bmatrix}^\intercal\right)_i \\
&= \nu'(\alpha_i)
\end{aligned}$$

and the second affirmation of the proposition also holds.

□

Proof of Proposition 3.4. Both inequalities are immediate from the definition of unrefuted and unsupported assertions respectively. If α is unrefuted:

$$\nu(\alpha) = \frac{c}{a} + \frac{c}{b}\sum_{B\subseteq\mathcal{A}}\left(f(B) * \sum_{\beta\in\mathcal{A}} \nu(\beta) I(T,B,\alpha,\beta)\right) \geq \frac{c}{a};$$

and that α in unsupported:

$$\nu(\alpha) = \frac{c}{a} - \frac{c}{b} \sum_{B \subseteq \mathcal{A}} \left(f(B) * \sum_{\beta \in \mathcal{A}} \nu(\beta) I(F, B, \alpha, \beta) \right) \leq \frac{c}{a}.$$

$\square$

Proof of Theorem 3.7. Clearly, if x_c is the solution to

$$(a \cdot \mathbb{1} - b \cdot M) \cdot X = \begin{bmatrix} c & c & \cdots & c \end{bmatrix}^\intercal$$

with $c > 0$, and x_1 is the solution to

$$(a \cdot \mathbb{1} - b \cdot M) \cdot X = \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix}^\intercal,$$

then $x_c = c \cdot x_1$. Since x_c and x_1 are rank-equivalent, we can fix $c = 1$ without loss of generality. Define a_{prov} as follows:

$$a_{prov} := 1 + |b| * (n-1) \left(\max_{1 \leq i,j \leq n} |M_{ij}| \right).$$

By the Levy-Desplanques theorem, for every $a \geq a_{prov}$, the matrix $(a \cdot \mathbb{1} - b \cdot M)$ is invertible. For every $a \geq a_{prov}$, we write x^a for the unique solution of the equation

$$(a \cdot \mathbb{1} - b \cdot M) \cdot X = \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix}^\intercal.$$

For $a \geq a_{prov}$, define $\delta_{ij}(a) := x_i^a - x_j^a$, where x_i^a is the ith coordinate of x^a. We will show that there is a value $a^* \geq a_{prov}$ such that for all $a_1, a_2 \geq a^*$, for all $1 \leq i < j \leq n$, $\delta_{ij}(a_1)$ and $\delta_{ij}(a_2)$ have the same sign, which implies that x^{a_1} and x^{a_2} are rank-equivalent.
Define the following matrix $S_{|i}$ in function of a, for $1 \leq i \leq n$:

$$(S_{|i}(a))_{\ell k} = \begin{cases} (a \cdot \mathbb{1} - b \cdot M)_{\ell k} & \text{if } k \neq i \\ 1 & \text{if } k = i \end{cases} \tag{D.1}$$

That is, $S_{|i}$ is obtained from $(a \cdot \mathbb{1} - b \cdot M)$ by replacing the ith column with $\begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix}^\intercal$. By Cramer's Rule, we have

$$x_i^a = \frac{\det\left(S_{|i}(a)\right)}{\det\left(a * \mathbb{1} - b * M\right)}, \tag{D.2}$$

and consequently

$$\delta_{ij}(a) = \frac{\det\big(S_{|i}(a)\big) - \det\big(S_{|j}(a)\big)}{\det\left(a \cdot \mathbb{1} - b \cdot M\right)}. \tag{D.3}$$

Since the determinants $\det(\cdot)$ are polynomial expressions, the following are all polynomials of degree at most n:

- $p_i(a) = \det\big(S_{|i}(a)\big)$;
- $p_j(a) = \det\big(S_{|j}(a)\big)$; and
- $p(a) = \det\left(a \cdot \mathbb{1} - b \cdot M\right)$.

If $a \geq a_{prov}$, then since the polynomial $p(a)$ does not vanish, the sign of $\delta_{ij}(a)$ is fully determined by the expression

$$\det\big(S_{|i}(a)\big) - \det\big(S_{|j}(a)\big). \tag{D.4}$$

Let $p_{ij} = p_i - p_j$ for $1 \leq i < j \leq n$. We determine a^* such that for all $a \geq a^*$ and for all $1 \leq i < j \leq n$, the sign of $p_{ij}(a)$ does not change (i.e., is either always positive or always negative). Note that the sign of p_{ij} not changing is equivalent to the sign of p_{ji} not changing, so we can assume $i < j$ without loss of generality.

In the remainder of the proof, we show that the desired a^* exists and can be computed in polynomial time in n. Pick $n+1$ real numbers $V = \{a_1, \ldots, a_{n+1}\}$, all greater than a_{prov}. Compute

$$p_{ij}(a_k) = \det\big(S_{|i}(a_k)\big) - \det\big(S_{|j}(a_k)\big) \tag{D.5}$$

for all $a_k \in V$ and $1 \leq i < j \leq n$. The set

$$\{p_{ij}(a_k) \mid 1 \leq i < j \leq n, 1 \leq k \leq n+1\}$$

can be computed in polynomial time in n, because it involves $n(n+1)$ determinants (i.e., $\det\big(S_{|i}(a_k)\big)$ for $1 \leq i \leq n$ and $1 \leq k \leq n+1$), each of which can be computed in polynomial time in n. For all $1 \leq i < j \leq n$, the polynomial p_{ij} can be computed from $\{(a_1, p_{ij}(a_1)), \ldots, (a_{n+1}, p_{ij}(a_{n+1}))\}$ in polynomial time using, for example, Lagrange interpolation. The number of polynomials to compute is $\frac{n(n-1)}{2}$ (i.e., polynomially many), and each of them has at most n coefficients. We will represent the polynomial p_{ij} by its coefficients, i.e., by $\langle (p_{ij})_0, \ldots, (p_{ij})_{n_{ij}} \rangle$ where each $(p_{ij})_\ell$ is the coefficient of degree ℓ, and $n_{ij} \leq n$ is the polynomial's degree. We now define a^*_{ij} as

$$a^*_{ij} := \max\left(a_{prov}, 2 + \max_{0 \leq \ell \leq n_{ij}-1} \frac{-(p_{ij})_\ell}{|(p_{ij})_{n_{ij}}|}\right).$$

By Cauchy's bound [59] on positive real roots of polynomials, if x_0 is a root of p_{ij}, then $x_0 < a^*_{ij}$. This implies that if $a_1, a_2 > a^*_{ij}$, then $p_{ij}(a_1)$ and $p_{ij}(a_2)$ have the same sign (i.e., either both positive or both negative), and therefore, by definition of δ_{ij}, we have $x_i^{a_1} < x_j^{a_1}$ if and only if $x_i^{a_2} < x_j^{a_2}$. Finally, let

$$a^* = \max_{1 \leq i < j \leq n} a^*_{ij},$$

which can be computed in polynomial time. By our construction, if follows that for all $a_1, a_2 > a^*$, the solutions x^{a_1}, x^{a_2} exist and are rank-equivalent. Finally, we incidentally note that a slightly better bound is obtained by letting

$$a^* = \max\left(a_{prov}, 2 + \max_{1 \leq i < j \leq n, 0 \leq \ell \leq n_{ij}-1} \frac{-(p_{ij})_\ell}{|(p_{ij})_{n_{ij}}|}\right). \tag{D.6}$$

This concludes the proof.

□

APPENDIX **E**

Proofs for Chapter 4

Proof of Theorem 4.2. The following is a well-known NP-complete problem [51]:

> **PROBLEM:** INDEPENDENT SET
>
> **Input:** A simple graph $G = (V, E)$; a positive integer $k \leq |V|$.
>
> **Question:** Does G have an independent set I with cardinality $|I| \geq k$?

This problem is also referenced as MAX INDEPENDENT SET in the literature. There is a straightforward polynomial-time many-one reduction from the problem INDEPENDENT SET to REPAIR-EXISTENCE$_{(\mathsf{COUNT},2)}$. We show next a polynomial-time many-one reduction from INDEPENDENT SET to the complement of REPAIR-CHECKING$_{(\mathsf{COUNT},2)}$. Let $G = (V, E)$, k be an input to INDEPENDENT SET. Let I be a set of fresh vertices such that $|I| = k - 1$. Let F be the set of all edges $\{u, v\}$ such that $u \in I$ and $v \in V$. Clearly, I is an inclusion-maximal independent set of the graph $H := (V \cup I, E \cup F)$, and the pair H, I is a legal input to REPAIR-CHECKING$_{(\mathsf{COUNT},2)}$. It is now easily verified that G has an independent set of cardinality $\geq k$ if and only if I is not a COUNT-repair of H. This concludes the proof.

$\square$

Proof of Lemma 4.4. Let $\mathcal{G} \in \mathbf{AGG}^{\mathsf{poly}}$ be a function that is monotone under priority. Let H, I, q be an input to SUITABILITY-CHECKING$_{(\mathcal{G},b)}$. If $\mathcal{G}_{\rhd w}(I) <$

q or I is not an independent set, return "no"; otherwise the *saturation* condition in the definition of q-suitable sets remains to be verified. To this end, compute in polynomial time the set S mentioned in Definition 4.8. Then compute in polynomial time its subset $S' := \{v \in S \mid I \cup \{v\} \text{ is an independent set}\}$. By Definition 4.5, I is saturated (and hence q-suitable) if and only if there is no nonempty set $J \subseteq V \setminus I$ such that $I \cup J$ is independent and $\mathcal{G}_{\triangleright w}(I) \leq \mathcal{G}_{\triangleright w}(I \cup J)$. Consequently, by Definition 4.8, I is saturated if and only if $S' = \emptyset$, which can be tested in polynomial time.

□

Proof of Lemma 4.5. Let H, q be an input to $\mathsf{REPAIR\text{-}EXISTENCE}_{(\mathcal{G},b)}$. We can compute in polynomial time the value m defined as follows:

$$m := \max\{\mathcal{G}_{\triangleright w}(J) \mid J \text{ is an independent set of } H \text{ with } |J| \leq k\}. \tag{E.1}$$

Since $\mathcal{G}$ is k-combinatorial, every repair I of H satisfies $\mathcal{G}_{\triangleright w}(I) = m$. Thus, the answer to $\mathsf{REPAIR\text{-}EXISTENCE}_{(\mathcal{G},b)}$ is "yes" if $q = m$, and "no" otherwise. □

Proof of Theorem 4.6. Let $\mathcal{G} \in \mathbf{AGG}^{\mathsf{poly}}_{(k)} \cap \mathbf{AGG}^{\mathsf{poly}}_{\mathsf{mon}}$. Let H, I be an input to the problem $\mathsf{REPAIR\text{-}CHECKING}_{(\mathcal{G},b)}$. We can compute, in polynomial time, the value m defined by (E.1) in the proof of Lemma 4.5. If $\mathcal{G}_{\triangleright w}(I) < m$, return "no"; otherwise we solve $\mathsf{SUITABILITY\text{-}CHECKING}_{(\mathcal{G},b)}$ with input H, I, m, which is in P by Lemma 4.4. In particular, if H, I, m is a "no"-instance of the problem $\mathsf{SUITABILITY\text{-}CHECKING}_{(\mathcal{G},b)}$, return "no". If we have not answered "no" so far, then $\mathcal{G}_{\triangleright w}(I) = m$ and I is an m-suitable set of H; in this case, return "yes". It is clear that this decision procedure is correct and runs in polynomial time.

□

Proof of Theorem 4.7. Membership in NP follows from Theorem 4.1. The NP-hardness proof is a polynomial-time many-one reduction from $\mathsf{3SAT}$. To this end, let φ be an instance of $\mathsf{3SAT}$ with k clauses. Let $(q_1, q_2, \ldots, q_n)$ with $n > k$ be the output of the task defined in Definition 4.10. Let w be the weight function that maps each i to q_i $(1 \leq i \leq n)$, and let $Q := \mathcal{G}_{\triangleright w}(\{1, \ldots, n\})$. Assume for the sake of contradiction that for some strict subset N of $\{1,\ldots,n\}$, we have $\mathcal{G}_{\triangleright w}(N)=Q$. Since $\mathcal{G}$ is not k-combinatorial, $|N|\geq k+1$. Then the sequence $(q_i)_{i\in N}$ of length $<n$ witnesses that $\mathcal{G}$ is not k-combinatorial, contradicting that Definition 4.10 requires a shortest witness. We conclude by contradiction that $N \subsetneq \{1,2,\ldots,n\}$ implies $\mathcal{G}_{\triangleright w}(N)\neq Q$. Since $\mathcal{G}$ is $\subseteq$-monotone, it follows that $N \subsetneq \{1,2,\ldots,n\}$ implies $\mathcal{G}_{\triangleright w}(N)<Q$. The reduction constructs, in polynomial time in the length of φ, a weighted graph $H=((V,w'),E)$ as follows. If the ith clause of φ is $\ell_1\vee\ell_2\vee\ell_3$, where ℓ_1,ℓ_2,ℓ_3 are positive or negative literals, then (i,ℓ_1), (i,ℓ_2), (i,ℓ_3) are vertices of V that form a triangle in E, and these three vertices are mapped to q_i by w'. For every propositional variable p, if (i,p) and $(j,\neg p)$ are vertices, then they are connected by an edge. Finally, we add isolated fresh vertices v_{k+1}, v_{k+2}, $\ldots$, v_n, and let $w'(v_j)=q_j$ for $k+1\leq j\leq n$. We claim that the following are equivalent:

1. φ has a satisfying truth assignment; and
2. H has a $\mathcal{G}$-repair I such that $\mathcal{G}_{\triangleright w'}(I)\geq Q$.

For the direction $1\Rightarrow 2$, let τ be a satisfying truth assignment for φ. Construct I as follows. First, I includes $\{v_{k+1},v_{k+2},\ldots,v_n\}$. Then, for i ranging from 1 to the number k of clauses, if the ith clause of φ is $\ell_1\vee\ell_2\vee\ell_3$, we pick $g\in\{1,2,3\}$ such that ℓ_g evaluates to true under τ, and add (i,ℓ_g) to I. In this way, I contains exactly one vertex from each triangle. Moreover, since τ is a truth assignment, we will never insert into I both (i,p) and $(j,\neg p)$ for a same propositional variable p. By construction, I is an independent set of H containing n elements, and $\mathcal{G}_{\triangleright w'}(I)=\mathcal{G}_{\triangleright w}(\{1,\ldots,n\})=Q$.
For the direction $2\Rightarrow 1$, let I be a $\mathcal{G}$-repair such that $\mathcal{G}_{\triangleright w'}(I)\geq Q$. Then, from our construction of H and our previous result that Q can only be attained if all q_is are aggregated, it follows that for every $i\in\{1,\ldots,k\}$, there is a literal ℓ in the ith clause such that I contains the vertex (i,ℓ). Moreover, since I is an independent set, it cannot contain both (i,p) and $(j,\neg p)$ for a same propositional variable p. Then I obviously defines a satisfying truth assignment for φ. This concludes the proof.

$\square$

APPENDIX F

Experiments with **Rustoner**

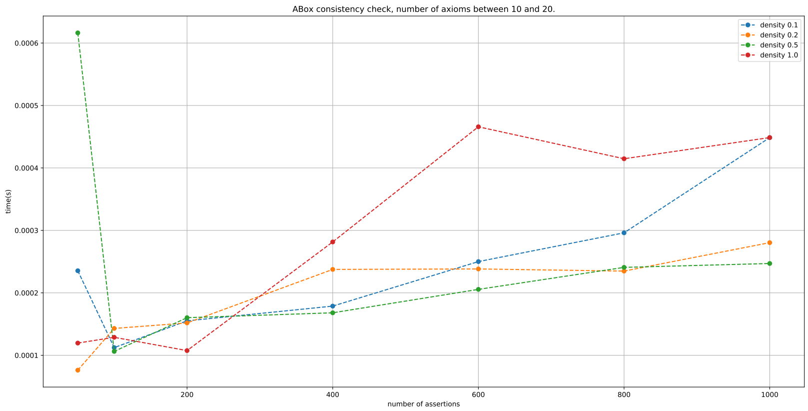

Figure F.1: Execution time for ABox consistency check, for TBox size varying between 10 and 20.

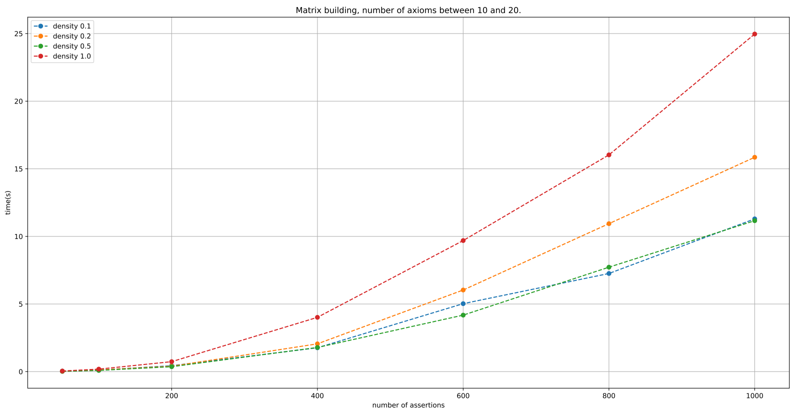

Figure F.2: Execution time for building the conflict matrix, for TBox size varying between 10 and 20.

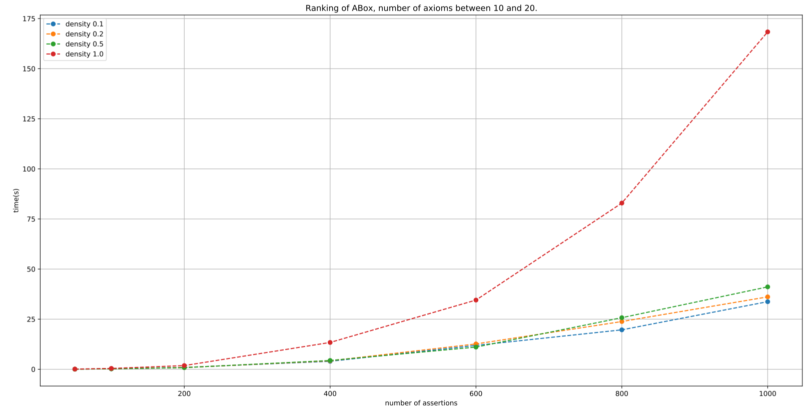

Figure F.3: Execution time for finding a stabilized assessment, for TBox size varying between 10 and 20.

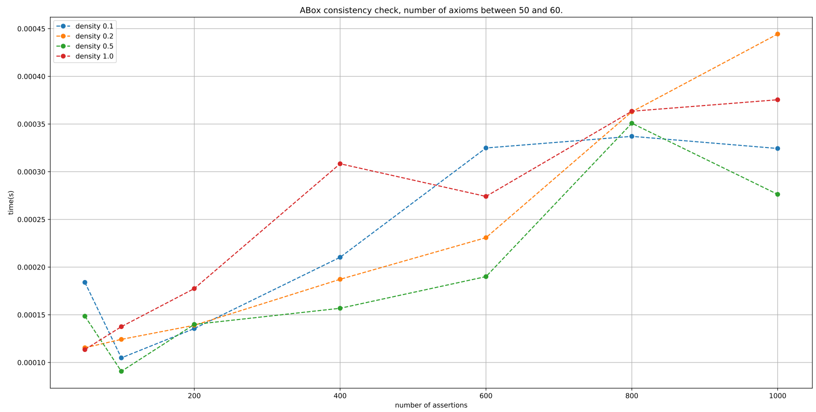

Figure F.4: Execution time for ABox consistency check, for TBox size varying between 50 and 60.

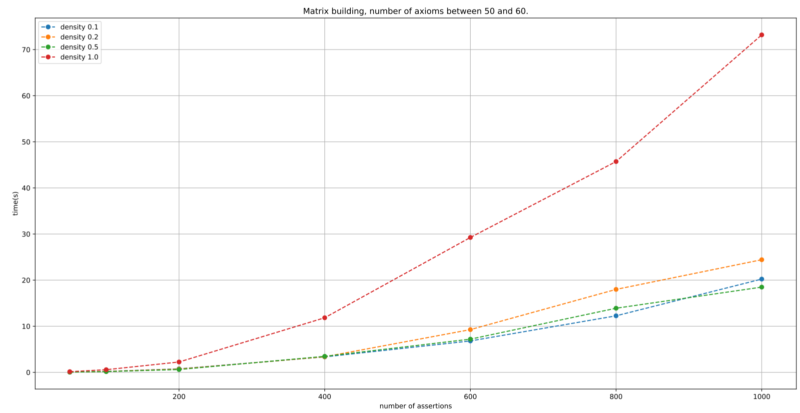

Figure F.5: Execution time for building the conflict matrix, for TBox size varying between 50 and 60.

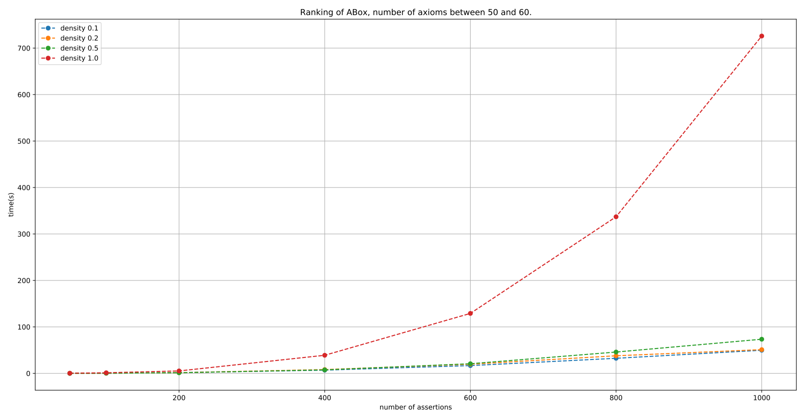

Figure F.6: Execution time for finding a stabilized assessment, for TBox size varying between 100 and 110

Bibliography

[1] 1000 common verbs in english. https://7esl.com/english-verbs/. Last Accessed: 2021-08-10.

[2] Amount of data created, consumed and stored 2010-2015. https://www.statista.com/statistics/871513/worldwide-data-created/. Latest Access: 12-07-2021.

[3] Big data growth statistics to blow your mind (or, what is a yottabyte anyway?). https://www.aparavi.com/resources-blog/data-growth-statistics-blow-your-mind. Latest Access: 12-07-2021.

[4] Top 1500 used english nouns. https://blog.kevmod.com/2016/06/benchmarking-minimum-vs-average/. Last Accessed: 2021-08-10.

[5] Top 1500 used english nouns. https://www.talkenglish.com/vocabulary/top-1500-nouns.aspx. Last Accessed: 2021-08-10.

[6] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995. URL: http://webdam.inria.fr/Alice/.

[7] Foto N. Afrati and Phokion G. Kolaitis. Repair checking in inconsistent databases: algorithms and complexity. In Ronald Fagin, editor, *Database Theory - ICDT 2009, 12th International Conference, St. Petersburg, Russia, March 23-25, 2009, Proceedings*, volume 361 of *ACM*

International Conference Proceeding Series, pages 31–41. ACM, 2009. doi:10.1145/1514894.1514899.

[8] Geir Agnarsson, Magnús M. Halldórsson, and Elena Losievskaja. Sdp-based algorithms for maximum independent set problems on hypergraphs. *Theor. Comput. Sci.*, 470:1–9, 2013. doi:10.1016/j.tcs.2012.11.025.

[9] Leila Amgoud and Jonathan Ben-Naim. Ranking-based semantics for argumentation frameworks. In Weiru Liu, V. S. Subrahmanian, and Jef Wijsen, editors, *Scalable Uncertainty Management - 7th International Conference, SUM 2013, Washington, DC, USA, September 16-18, 2013. Proceedings*, volume 8078 of *Lecture Notes in Computer Science*, pages 134–147. Springer, 2013. doi:10.1007/978-3-642-40381-1_11.

[10] Ed Anderson, Zhaojun Bai, Jack J. Dongarra, Anne Greenbaum, A. McKenney, Jeremy Du Croz, Sven Hammarling, James Demmel, Christian H. Bischof, and Danny C. Sorensen. LAPACK: a portable linear algebra library for high-performance computers. In Joanne L. Martin, Daniel V. Pryor, and Gary Montry, editors, *Proceedings Supercomputing '90, New York, NY, USA, November 12-16, 1990*, pages 2–11. IEEE Computer Society, 1990. doi:10.1109/SUPERC.1990.129995.

[11] Hajnal Andréka, István Németi, and Johan Benthem. Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic*, 27, 06 1998. doi:10.1023/A:1004275029985.

[12] Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In Victor Vianu and Christos H. Papadimitriou, editors, *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 31 - June 2, 1999, Philadelphia, Pennsylvania, USA*, pages 68–79. ACM Press, 1999. doi:10.1145/303976.303983.

[13] Marcelo Arenas, Óscar Corcho, Elena Simperl, Markus Strohmaier, Mathieu d'Aquin, Kavitha Srinivas, Paul Groth, Michel Dumontier, Jeff Heflin, Krishnaprasad Thirunarayan, and Steffen Staab, editors. *The*

Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part I, volume 9366 of *Lecture Notes in Computer Science*. Springer, 2015. doi:10.1007/978-3-319-25007-6.

[14] S. Axler. *Linear Algebra Done Right*. Undergraduate Texts in Mathematics. Springer International Publishing, 2014. URL: https://books.google.be/books?id=5qYxBQAAQBAJ.

[15] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[16] Franz Baader, Ian Horrocks, Carsten Lutz, and Ulrike Sattler. *An Introduction to Description Logic*. Cambridge University Press, 2017. URL: http://www.cambridge.org/de/academic/subjects/computer-science/knowledge-management-databases-and-data-mining/introduction-description-logic?format=PB#17zVGeWD2TZUeu6s.97.

[17] D.A. Belsley, E. Kuh, and R.E. Welsch. *Regression Diagnostics: Identifying Influential Data and Sources of Collinearity*. Wiley Series in Probability and Statistics. Wiley, 2004. URL: https://books.google.be/books?id=OwImTJMwNgwC.

[18] Leopoldo E. Bertossi. Database repairs and consistent query answering: Origins and further developments. In Dan Suciu, Sebastian Skritek, and Christoph Koch, editors, *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 48–58. ACM, 2019. doi:10.1145/3294052.3322190.

[19] Leopoldo E. Bertossi, Loreto Bravo, Enrico Franconi, and Andrei Lopatenko. The complexity and approximation of fixing numerical attributes in databases under integrity constraints. *Inf. Syst.*, 33(4-5):407–434, 2008. doi:10.1016/j.is.2008.01.005.

[20] Meghyn Bienvenu. Inconsistency-tolerant ontology-based data access revisited: Taking mappings into account. In Magdalena Ortiz and Thomas Schneider, editors, *Proceedings of the 31st International Workshop on Description Logics co-located with 16th International Conference on Principles of Knowledge Representation and Reasoning (KR 2018), Tempe, Arizona, US, October 27th - to - 29th, 2018.*, volume 2211 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2018. URL: `http://ceur-ws.org/Vol-2211/paper-41.pdf`.

[21] Meghyn Bienvenu. Inconsistency-tolerant ontology-based data access revisited: Taking mappings into account. In Lang [71], pages 1721–1729. `doi:10.24963/ijcai.2018/238`.

[22] Meghyn Bienvenu, Camille Bourgaux, and François Goasdoué. Querying inconsistent description logic knowledge bases under preferred repair semantics. In Carla E. Brodley and Peter Stone, editors, *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada*, pages 996–1002. AAAI Press, 2014. URL: `http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8231`.

[23] Meghyn Bienvenu, Camille Bourgaux, and François Goasdoué. Computing and explaining query answers over inconsistent dl-lite knowledge bases. *J. Artif. Intell. Res.*, 64:563–644, 2019. `doi:10.1613/jair.1.11395`.

[24] Meghyn Bienvenu, Pierre Bourhis, Marie-Laure Mugnier, Sophie Tison, and Federico Ulliana. Ontology-mediated query answering for key-value stores. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 844–851. ijcai.org, 2017. `doi:10.24963/ijcai.2017/117`.

[25] Meghyn Bienvenu and Riccardo Rosati. Tractable approximations of consistent query answering for robust ontology-based data access. In Francesca Rossi, editor, *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China,*

August 3-9, 2013, pages 775–781. IJCAI/AAAI, 2013. URL: http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6904.

[26] Philip Bohannon, Michael Flaster, Wenfei Fan, and Rajeev Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In Fatma Özcan, editor, *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*, pages 143–154. ACM, 2005. doi:10.1145/1066157.1066175.

[27] Elise Bonzon, Jérôme Delobelle, Sébastien Konieczny, and Nicolas Maudet. A comparative study of ranking-based semantics for abstract argumentation. In Dale Schuurmans and Michael P. Wellman, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 914–920. AAAI Press, 2016. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12465.

[28] Richard Booth, Martin Caminada, Mikolaj Podlaszewski, and Iyad Rahwan. Quantifying disagreement in argument-based reasoning. In Wiebe van der Hoek, Lin Padgham, Vincent Conitzer, and Michael Winikoff, editors, *International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2012, Valencia, Spain, June 4-8, 2012 (3 Volumes)*, pages 493–500. IFAAMAS, 2012. URL: http://dl.acm.org/citation.cfm?id=2343647.

[29] Elena Botoeva, Diego Calvanese, Benjamin Cogrel, Julien Corman, and Guohui Xiao. A generalized framework for ontology-based data access. In Chiara Ghidini, Bernardo Magnini, Andrea Passerini, and Paolo Traverso, editors, *AI*IA 2018 - Advances in Artificial Intelligence - XVIIth International Conference of the Italian Association for Artificial Intelligence, Trento, Italy, November 20-23, 2018, Proceedings*, volume 11298 of *Lecture Notes in Computer Science*, pages 166–180. Springer, 2018. doi:10.1007/978-3-030-03840-3_13.

[30] Marco Calautti, Leonid Libkin, and Andreas Pieris. An operational approach to consistent query answering. In Jan Van den Bussche

and Marcelo Arenas, editors, *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Houston, TX, USA, June 10-15, 2018*, pages 239–251. ACM, 2018. doi:10.1145/3196959.3196966.

[31] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *J. Autom. Reason.*, 39(3):385–429, 2007. doi:10.1007/s10817-007-9078-x.

[32] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Query processing under GLAV mappings for relational and graph databases. *Proc. VLDB Endow.*, 6(2):61–72, 2012. URL: http://www.vldb.org/pvldb/vol6/p61-calvanese.pdf, doi:10.14778/2535568.2448940.

[33] Diego Calvanese, Martin Giese, Dag Hovland, and Martin Rezk. Ontology-based integration of cross-linked datasets. In Arenas et al. [13], pages 199–216. doi:10.1007/978-3-319-25007-6_12.

[34] Jiahao Chen and Jarrett Revels. Robust benchmarking in noisy environments. *CoRR*, abs/1608.04295, 2016. URL: http://arxiv.org/abs/1608.04295, arXiv:1608.04295.

[35] Peter P. Chen. The entity-relationship model - toward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36, 1976. doi:10.1145/320434.320440.

[36] Jan Chomicki. Consistent query answering: Five easy pieces. In Thomas Schwentick and Dan Suciu, editors, *Database Theory - ICDT 2007, 11th International Conference, Barcelona, Spain, January 10-12, 2007, Proceedings*, volume 4353 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2007. doi:10.1007/11965893_1.

[37] Jan Chomicki and Jerzy Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Inf. Comput.*, 197(1-2):90–121, 2005. doi:10.1016/j.ic.2004.04.007.

[38] Jan Chomicki, Jerzy Marcinkowski, and Slawomir Staworko. Computing consistent query answers using conflict hypergraphs. In David A. Grossman, Luis Gravano, ChengXiang Zhai, Otthein Herzog, and David A. Evans, editors, *Proceedings of the 2004 ACM CIKM International Conference on Information and Knowledge Management, Washington, DC, USA, November 8-13, 2004*, pages 417–426. ACM, 2004. doi:10.1145/1031171.1031254.

[39] Sik Chun, Joey Lam, Jeff Z. Pan, D. Sleeman, and W. Vasconcelos. Ontology inconsistency handling : Ranking and rewriting axioms. Technical report, 2006.

[40] Marco Console and Maurizio Lenzerini. Data quality in ontology-based data access: The case of consistency. In Carla E. Brodley and Peter Stone, editors, *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 1020–1026. AAAI Press, 2014. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8552.

[41] James Cooley and John Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.

[42] Vilhelm Dahllöf and Peter Jonsson. An algorithm for counting maximum weighted independent sets and its applications. In David Eppstein, editor, *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA*, pages 292–298. ACM/SIAM, 2002. URL: http://dl.acm.org/citation.cfm?id=545381.545420.

[43] Cristobald de Kerchove and Paul Van Dooren. The pagetrust algorithm: How to rank web pages when negative links are allowed? In *Proceedings of the SIAM International Conference on Data Mining, SDM 2008, April 24-26, 2008, Atlanta, Georgia, USA*, pages 346–352. SIAM, 2008. doi:10.1137/1.9781611972788.31.

[44] Ala Djeddai, Hassina Seridi, and Tarek Khadir. A new dl-lite N bool probabilistic extension using belief. In Thomas Lukasiewicz, Rafael

Peñaloza, and Anni-Yasmin Turhan, editors, *Proceedings of the First Workshop on Logics for Reasoning about Preferences, Uncertainty, and Vagueness, PRUV 2014, co-located with 7th International Joint Conference on Automated Reasoning (IJCAR 2014), Vienna, Austria, July 23-24, 2014*, volume 1205 of *CEUR Workshop Proceedings*, pages 88–100. CEUR-WS.org, 2014. URL: `http://ceur-ws.org/Vol-1205/00010088.pdf`.

[45] Jianfeng Du and Guilin Qi. Tractable computation of representative abox repairs in description logic ontologies. In Songmao Zhang, Martin Wirsing, and Zili Zhang, editors, *Knowledge Science, Engineering and Management - 8th International Conference, KSEM 2015, Chongqing, China, October 28-30, 2015, Proceedings*, volume 9403 of *Lecture Notes in Computer Science*, pages 28–39. Springer, 2015. `doi:10.1007/978-3-319-25159-2\_3`.

[46] Jianfeng Du, Guilin Qi, and Yi-Dong Shen. Weight-based consistent query answering over inconsistent $\mathcal{SHIQ}$ knowledge bases. *Knowl. Inf. Syst.*, 34(2):335–371, 2013. `doi:10.1007/s10115-012-0478-9`.

[47] Ronald Fagin, Benny Kimelfeld, and Phokion G. Kolaitis. Dichotomies in the complexity of preferred repairs. In Tova Milo and Diego Calvanese, editors, *Proceedings of the 34th ACM Symposium on Principles of Database Systems, PODS 2015, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 3–15. ACM, 2015. `doi:10.1145/2745754.2745762`.

[48] Sergio Flesca, Filippo Furfaro, and Francesco Parisi. Querying and repairing inconsistent numerical databases. *ACM Trans. Database Syst.*, 35(2):14:1–14:50, 2010. `doi:10.1145/1735886.1735893`.

[49] Matteo Frigo and Steven G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. Special issue on "Program Generation, Optimization, and Platform Adaptation".

[50] Woon Siong Gan. *Fast Fourier Transform*, pages 17–20. Springer Singapore, Singapore, 2020. `doi:10.1007/978-981-10-5550-8_5`.

[51] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[52] Birte Glimm, Ian Horrocks, Boris Motik, Giorgos Stoilos, and Zhe Wang. Hermit: An OWL 2 reasoner. *J. Autom. Reason.*, 53(3):245–269, 2014. doi:10.1007/s10817-014-9305-1.

[53] Erich Grädel. On the restraining power of guards. *J. Symb. Log.*, 64(4):1719–1742, 1999. doi:10.2307/2586808.

[54] Sergio Greco and Cristian Molinaro. Probabilistic query answering over inconsistent databases. *Ann. Math. Artif. Intell.*, 64(2-3):185–207, 2012. doi:10.1007/s10472-012-9287-9.

[55] Eric Gribkoff, Guy Van den Broeck, and Dan Suciu. The most probable database problem. In *Proceedings of the First International Workshop on Big Uncertain Data (BUDA)*, June 2014. URL: http://starai.cs.ucla.edu/papers/GribkoffBUDA14.pdf.

[56] Maxim Haddad and Diego Calvanese. Extending dl-lite_a with (singleton) nominals. In Thomas Eiter, Birte Glimm, Yevgeny Kazakov, and Markus Krötzsch, editors, *Informal Proceedings of the 26th International Workshop on Description Logics, Ulm, Germany, July 23 - 26, 2013*, volume 1014 of *CEUR Workshop Proceedings*, pages 704–723. CEUR-WS.org, 2013. URL: http://ceur-ws.org/Vol-1014/paper_46.pdf.

[57] Magnús M. Halldórsson and Elena Losievskaja. Independent sets in bounded-degree hypergraphs. *Discret. Appl. Math.*, 157(8):1773–1786, 2009. doi:10.1016/j.dam.2008.11.013.

[58] John L. Hennessy and David A. Patterson. *Computer Architecture - A Quantitative Approach, 5th Edition*. Morgan Kaufmann, 2012.

[59] Holly P. Hirst and Wade T. Macey. Bounding the roots of polynomials. *The College Mathematics Journal*, 28(4):292–295, 1997. URL: http://www.jstor.org/stable/2687152.

[60] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Reasoning with individuals for the description logic SHIQ. *CoRR*, cs.LO/0005017, 2000. URL: `https://arxiv.org/abs/cs/0005017`.

[61] J. Humpherys and T.J. Jarvis. *Foundations of Applied Mathematics, Volume 2: Algorithms, Approximation, Optimization*. Other Titles in Applied Mathematics. SIAM, 2020. URL: `https://books.google.be/books?id=yjbWDwAAQBAJ`.

[62] Anthony Hunter and Matthias Thimm. Probabilistic reasoning with abstract argumentation frameworks. *J. Artif. Intell. Res.*, 59:565–611, 2017. `doi:10.1613/jair.5393`.

[63] Ihab F. Ilyas and Xu Chu. *Data Cleaning*. ACM, 2019. `doi:10.1145/3310205`.

[64] E. Isaacson and H.B. Keller. *Analysis of Numerical Methods*. Dover Books on Mathematics. Dover Publications, 1994. URL: `https://books.google.be/books?id=y77n2ySMJHUC`.

[65] Akihisa Kako, Takao Ono, Tomio Hirata, and Magnús M. Halldórsson. Approximation algorithms for the weighted independent set problem in sparse graphs. *Discret. Appl. Math.*, 157(4):617–626, 2009. `doi:10.1016/j.dam.2008.08.027`.

[66] Benny Kimelfeld, Ester Livshits, and Liat Peterfreund. Detecting ambiguity in prioritized database repairing. In Michael Benedikt and Giorgio Orsi, editors, *20th International Conference on Database Theory, ICDT 2017, March 21-24, 2017, Venice, Italy*, volume 68 of *LIPIcs*, pages 17:1–17:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. `doi:10.4230/LIPIcs.ICDT.2017.17`.

[67] Steve Klabnik and Carol Nichols. *The Rust Programming Language*. No Starch Press, USA, 2018.

[68] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5):604–632, 1999. URL: `http://doi.acm.org/10.1145/324133.324140`, `doi:10.1145/324133.324140`.

[69] Vladimir Kolovski, Bijan Parsia, and Evren Sirin. Extending the *SHOIQ(D)* tableaux with dl-safe rules: First results. In Bijan Parsia, Ulrike Sattler, and David Toman, editors, *Proceedings of the 2006 International Workshop on Description Logics (DL2006), Windermere, Lake District, UK, May 30 - June 1, 2006*, volume 189 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006. URL: `http://ceur-ws.org/Vol-189/submission_26.pdf`.

[70] Sik Chun Lam, Jeff Z. Pan, Derek H. Sleeman, and Wamberto Weber Vasconcelos. A fine-grained approach to resolving unsatisfiable ontologies. In *2006 IEEE / WIC / ACM International Conference on Web Intelligence (WI 2006), 18-22 December 2006, Hong Kong, China*, pages 428–434. IEEE Computer Society, 2006. `doi:10.1109/WI.2006.11`.

[71] Jérôme Lang, editor. *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*. ijcai.org, 2018. URL: `http://www.ijcai.org/proceedings/2018/`.

[72] S. Lang. *Linear Algebra*. Springer, 2014. URL: `https://books.google.be/books?id=k_QlswEACAAJ`.

[73] Dirk Leinders, Maarten Marx, Jerzy Tyszkiewicz, and Jan Van den Bussche. The semijoin algebra and the guarded fragment. *Journal of Logic, Language and Information*, 14(3):331–343, 2005. `doi:10.1007/s10849-005-5789-8`.

[74] Domenico Lembo, Maurizio Lenzerini, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. Inconsistency-tolerant semantics for description logics. In Pascal Hitzler and Thomas Lukasiewicz, editors, *Web Reasoning and Rule Systems - Fourth International Conference, RR 2010, Bressanone/Brixen, Italy, September 22-24, 2010. Proceedings*, volume 6333 of *Lecture Notes in Computer Science*, pages 103–117. Springer, 2010. `doi:10.1007/978-3-642-15918-3\_9`.

[75] Domenico Lembo, José Mora, Riccardo Rosati, Domenico Fabio Savo, and Evgenij Thorstensen. Mapping analysis in ontology-based data ac-

cess: Algorithms and complexity. In Arenas et al. [13], pages 217–234. doi:10.1007/978-3-319-25007-6_13.

[76] Leonid Libkin. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004. URL: http://www.cs.toronto.edu/%7Elibkin/fmt, doi:10.1007/978-3-662-07003-1.

[77] Ester Livshits and Benny Kimelfeld. Counting and enumerating (preferred) database repairs. In Emanuel Sallinger, Jan Van den Bussche, and Floris Geerts, editors, *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017*, pages 289–301. ACM, 2017. doi:10.1145/3034786.3056107.

[78] Ester Livshits, Benny Kimelfeld, and Sudeepa Roy. Computing optimal repairs for functional dependencies. *ACM Trans. Database Syst.*, 45(1):4:1–4:46, 2020. doi:10.1145/3360904.

[79] Andrei Lopatenko and Leopoldo E. Bertossi. Complexity of consistent query answering in databases under cardinality-based and incremental repair semantics (extended version). *CoRR*, abs/1605.07159, 2016. URL: http://arxiv.org/abs/1605.07159, arXiv:1605.07159.

[80] J. Loughry, J.I. van Hemert, and L. Schoofs. Efficiently enumerating the subsets of a set.

[81] Dany Maslowski and Jef Wijsen. Uncertainty that counts. In Henning Christiansen, Guy De Tré, Adnan Yazici, Slawomir Zadrozny, Troels Andreasen, and Henrik Legind Larsen, editors, *Flexible Query Answering Systems - 9th International Conference, FQAS 2011, Ghent, Belgium, October 26-28, 2011 Proceedings*, volume 7022 of *Lecture Notes in Computer Science*, pages 25–36. Springer, 2011. doi:10.1007/978-3-642-24764-4_3.

[82] Marie-Laure Mugnier, Marie-Christine Rousset, and Federico Ulliana. Ontology-mediated queries for NOSQL databases. In Dale Schuurmans and Michael P. Wellman, editors, *Proceedings of the Thirtieth AAAI*

Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA, pages 1051–1057. AAAI Press, 2016. URL: `http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12395`.

[83] Linh Anh Nguyen. Designing a tableau reasoner for description logics. In Hoai An Le Thi, Ngoc Thanh Nguyen, and Tien Van Do, editors, *Advanced Computational Methods for Knowledge Engineering - Proceedings of 3rd International Conference on Computer Science, Applied Mathematics and Applications - ICCSAMA 2015, Metz, France, 11-13 May, 2015*, volume 358 of *Advances in Intelligent Systems and Computing*, pages 321–333. Springer, 2015. `doi:10.1007/978-3-319-17996-4\_29`.

[84] Linh Anh Nguyen and Joanna Golinska-Pilarek. An ExpTime tableau method for dealing with nominals and qualified number restrictions in deciding the description logic SHOQ. *Fundam. Informaticae*, 135(4):433–449, 2014. `doi:10.3233/FI-2014-1133`.

[85] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120. URL: `http://ilpubs.stanford.edu:8090/422/`.

[86] Francesco Pagliarecci, Luca Spalazzi, and Gilberto Taccari. Reasoning with temporal aboxes: Combining dl-lite_core with CTL. In Thomas Eiter, Birte Glimm, Yevgeny Kazakov, and Markus Krötzsch, editors, *Informal Proceedings of the 26th International Workshop on Description Logics, Ulm, Germany, July 23 - 26, 2013*, volume 1014 of *CEUR Workshop Proceedings*, pages 885–897. CEUR-WS.org, 2013. URL: `http://ceur-ws.org/Vol-1014/paper_71.pdf`.

[87] Victor Y. Pan. How bad are vandermonde matrices? *SIAM J. Matrix Anal. Appl.*, 37(2):676–694, 2016. `doi:10.1137/15M1030170`.

[88] Horacio Tellez Perez. `rustoner` source code. `https://github.com/hatellezp/rustoner`. Accessed: 2021-05-26.

[89] Horacio Tellez Perez and Jef Wijsen. Connecting databases and ontologies: A data quality perspective. In Mantas Simkus and Grant E.

Weddell, editors, *Proceedings of the 32nd International Workshop on Description Logics, Oslo, Norway, June 18-21, 2019*, volume 2373 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2019. URL: `http://ceur-ws.org/Vol-2373/paper-26.pdf`.

[90] Horacio Tellez Perez and Jef Wijsen. Logic-based ranking of assertions in inconsistent ABoxes. In Stefan Borgwardt and Thomas Meyer, editors, *Proceedings of the 33rd International Workshop on Description Logics (DL 2020) co-located with the 17th International Conference on Principles of Knowledge Representation and Reasoning (KR 2020), Online Event [Rhodes, Greece], September 12th to 14th, 2020*, volume 2663 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2020. URL: `http://ceur-ws.org/Vol-2663/paper-20.pdf`.

[91] Horacio Tellez Perez and Jef Wijsen. Generalized weighted repairs. In Troels Andreasen, Guy De Tré, Janusz Kacprzyk, Henrik Legind Larsen, Gloria Bordogna, and Sławomir Zadrożny, editors, *Flexible Query Answering Systems - 14th International Conference, FQAS 2021, Bratislava, Slovakia, September 19-21, 2021, Proceedings*, volume 12871 of *Lecture Notes in Computer Science*, pages 67–81. Springer, 2021. `doi:https://doi.org/10.1007/978-3-030-86967-0_6`.

[92] Chris Phillips. The performance of the BLAS and LAPACK on a shared memory scalar multiprocessor. *Parallel Comput.*, 17(6-7):751–761, 1991. `doi:10.1016/S0167-8191(05)80064-X`.

[93] G.M. Phillips. *Interpolation and Approximation by Polynomials*. CMS Books in Mathematics. Springer, 2003. URL: `https://books.google.be/books?id=87vciTxMcF8C`.

[94] Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Linking data to ontologies. *J. Data Semantics*, 10:133–173, 2008. `doi:10.1007/978-3-540-77688-8\_5`.

[95] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing*. Numerical

Recipes in C book set. Cambridge University Press, 1992. URL: `https://books.google.be/books?id=2WFJyAEACAAJ`.

[96] Badran Raddaoui. On the measure of conflicts: an argumentation-based framework. *J. Appl. Non Class. Logics*, 28(2-3):240–259, 2018. `doi:10.1080/11663081.2018.1457255`.

[97] Michael Revers. A survey on lagrange interpolation based on equally spaced nodes. In Martin D. Buhmann and Detlef H. Mache, editors, *Advanced Problems in Constructive Approximation*, pages 153–164, Basel, 2003. Birkhäuser Basel.

[98] Riccardo Rosati. Finite model reasoning in dl-lite. In Sean Bechhofer, Manfred Hauswirth, Jörg Hoffmann, and Manolis Koubarakis, editors, *The Semantic Web: Research and Applications, 5th European Semantic Web Conference, ESWC 2008, Tenerife, Canary Islands, Spain, June 1-5, 2008, Proceedings*, volume 5021 of *Lecture Notes in Computer Science*, pages 215–229. Springer, 2008. `doi:10.1007/978-3-540-68234-9\_18`.

[99] Riccardo Rosati. On the complexity of dealing with inconsistency in description logic ontologies. In Toby Walsh, editor, *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 1057–1062. IJCAI/AAAI, 2011. `doi:10.5591/978-1-57735-516-8/IJCAI11-181`.

[100] Juan F. Sequeda. Integrating relational databases with the semantic web: A reflection. In Giovambattista Ianni, Domenico Lembo, Leopoldo E. Bertossi, Wolfgang Faber, Birte Glimm, Georg Gottlob, and Steffen Staab, editors, *Reasoning Web. Semantic Interoperability on the Web - 13th International Summer School 2017, London, UK, July 7-11, 2017, Tutorial Lectures*, volume 10370 of *Lecture Notes in Computer Science*, pages 68–120. Springer, 2017. `doi:10.1007/978-3-319-61033-7\_4`.

[101] Juan F. Sequeda, Syed Hamid Tirmizi, Óscar Corcho, and Daniel P. Miranker. Survey of directly mapping SQL databases to the semantic web. *Knowledge Eng. Review*, 26(4):445–486, 2011. `doi:10.1017/S0269888911000208`.

[102] Nigel Shadbolt, Tim Berners-Lee, and Wendy Hall. The semantic web revisited. *IEEE Intell. Syst.*, 21(3):96–101, 2006. doi:10.1109/MIS.2006.62.

[103] Gerardo I. Simari, Paulo Shakarian, and Marcelo A. Falappa. A quantitative approach to belief revision in structured probabilistic argumentation. *Ann. Math. Artif. Intell.*, 76(3-4):375–408, 2016. doi:10.1007/s10472-015-9476-4.

[104] Dimitrios-Emmanuel Spanos, Periklis Stavrou, and Nikolas Mitrou. Bringing relational databases into the semantic web: A survey. *Semantic Web*, 3(2):169–209, 2012. doi:10.3233/SW-2011-0055.

[105] Slawek Staworko, Jan Chomicki, and Jerzy Marcinkowski. Prioritized repairing and consistent query answering in relational databases. *Ann. Math. Artif. Intell.*, 64(2-3):209–246, 2012. doi:10.1007/s10472-012-9288-8.

[106] Slawomir Staworko and Jan Chomicki. Consistent query answers in the presence of universal constraints. *Inf. Syst.*, 35(1):1–22, 2010. doi:10.1016/j.is.2009.03.004.

[107] Andreas Steigmiller. *Optimisation of tableau-based reasoning systems for expressive description logics.* PhD thesis, University of Ulm, Germany, 2016. URL: https://nbn-resolving.org/urn:nbn:de:bsz:289-oparu-4042-6.

[108] Andreas Steigmiller, Thorsten Liebig, and Birte Glimm. Konclude: System description. *J. Web Semant.*, 27-28:78–85, 2014. doi:10.1016/j.websem.2014.06.003.

[109] Kai Sun. An algorithm framework for the exact solution and improved approximation of the maximum weighted independent set problem. *CoRR*, abs/2008.01961, 2020. URL: https://arxiv.org/abs/2008.01961, arXiv:2008.01961.

[110] Abdelmoutia Telli, Salem Benferhat, Mustapha Bourahla, Zied Bouraoui, and Karim Tabia. Polynomial algorithms for computing a

single preferred assertional-based repair. *Künstliche Intell.*, 31(1):15–30, 2017. doi:10.1007/s13218-016-0466-4.

[111] Balder ten Cate and Massimo Franceschet. Guarded fragments with constants. *Journal of Logic, Language and Information*, 14(3):281–288, 2005. doi:10.1007/s10849-005-5787-x.

[112] Camilo Thorne, Raffaella Bernardi, and Diego Calvanese. Designing efficient controlled languages for ontologies. In *Computing Meaning: Volume 4*, pages 149–173. Springer Netherlands, Dordrecht, 2014. doi:10.1007/978-94-007-7284-7_9.

[113] Vincent A. Traag, Yurii E. Nesterov, and Paul Van Dooren. Exponential ranking: Taking into account negative links. In Leonard Bolc, Marek Makowski, and Adam Wierzbicki, editors, *Social Informatics - Second International Conference, SocInfo 2010, Laxenburg, Austria, October 27-29, 2010. Proceedings*, volume 6430 of *Lecture Notes in Computer Science*, pages 192–202. Springer, 2010. doi:10.1007/978-3-642-16567-2_14.

[114] Dmitry Tsarkov and Ian Horrocks. Fact++ description logic reasoner: System description. In Ulrich Furbach and Natarajan Shankar, editors, *Automated Reasoning, Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4130 of *Lecture Notes in Computer Science*, pages 292–297. Springer, 2006. doi:10.1007/11814771_26.

[115] Richard S. Varga. On recurring theorems on diagonal dominance. *Linear Algebra and its Applications*, 13(1):1 – 9, 1976. URL: http://www.sciencedirect.com/science/article/pii/0024379576900379, doi:https://doi.org/10.1016/0024-3795(76)90037-9.

[116] Yair Wand and Richard Y. Wang. Anchoring data quality dimensions in ontological foundations. *Commun. ACM*, 39(11):86–95, 1996. doi:10.1145/240455.240479.

[117] Xi Wang and Chen Wang. Time series data cleaning: A survey. *IEEE Access*, 8:1866–1881, 2020. doi:10.1109/ACCESS.2019.2962152.

[118] Jef Wijsen. Foundations of query answering on inconsistent databases. *SIGMOD Rec.*, 48(3):6–16, 2019. doi:10.1145/3377391.3377393.

[119] Guohui Xiao, Diego Calvanese, Roman Kontchakov, Domenico Lembo, Antonella Poggi, Riccardo Rosati, and Michael Zakharyaschev. Ontology-based data access: A survey. In Lang [71], pages 5511–5519. doi:10.24963/ijcai.2018/777.

[120] Guohui Xiao, Dag Hovland, Dimitris Bilidas, Martin Rezk, Martin Giese, and Diego Calvanese. Efficient ontology-based data integration with canonical iris. In Aldo Gangemi, Roberto Navigli, Maria-Esther Vidal, Pascal Hitzler, Raphaël Troncy, Laura Hollink, Anna Tordai, and Mehwish Alam, editors, *The Semantic Web - 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3-7, 2018, Proceedings*, volume 10843 of *Lecture Notes in Computer Science*, pages 697–713. Springer, 2018. doi:10.1007/978-3-319-93417-4_45.

[121] Riccardo Zese, Elena Bellodi, Fabrizio Riguzzi, Giuseppe Cota, and Evelina Lamma. Tableau reasoning for description logics and its extension to probabilities. *Ann. Math. Artif. Intell.*, 82(1-3):101–130, 2018. doi:10.1007/s10472-016-9529-3.

[122] Weiguo Zheng, Jiewei Gu, Peng Peng, and Jeffrey Xu Yu. Efficient weighted independent set computation over large graphs. In *36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020*, pages 1970–1973. IEEE, 2020. doi:10.1109/ICDE48307.2020.00216.

Dealing with Inconsistencies in Knowledge Bases

This thesis develops and studies theoretical frameworks for dealing with inconsistencies in database and knowledge-base systems. A first framework defines a mapping language for expressing rules that take a relational database instance as input, and produce an ABox in some description logic (DL). Given a family of mapping rules, it is desirable that every database instance that is consistent with respect to some given integrity constraints maps to an ABox that is consistent with respect to a given TBox. While it is generally undecidable whether this and other desirable properties obtain, it is shown that decidability can be achieved under some moderate syntactic restrictions.
A second framework addresses the problem of repairing ABoxes that are inconsistent with respect to a given TBox. It introduces a novel approach for computing a numeric credibility score for each ABox assertion, by combining a user-defined initial scoring with logical arguments and counterarguments derived from the TBox. Once a credibility score has been established for each ABox assertion (or, in general, for each fact of a knowledge base), it is natural to define repairs as consistent subsets of the ABox with maximum aggregate credibility score, according to some aggregation function. It is studied how the computational complexity of recognizing such repairs depends on certain characteristics of the aggregation function.
In addition to these theoretical developments, a software system has been built that implements the computational approach underlying the second framework.

UMONS
Université de Mons