FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Querying Rich Ontologies by Exploiting the Structure of Data

DISSERTATION

zur Erlangung des akademischen Grades

Doktor der Technischen Wissenschaften

eingereicht von

Labinot Bajraktari
Matrikelnummer 1429606

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Dr.Techn. Magdalena Ortiz
Zweitbetreuung: Univ.-Prof.Dr. Stefan Szeider

Diese Dissertation haben begutachtet:

Domenico Lembo	Sebastian Rudolph

Wien, 12. Dezember 2019

Labinot Bajraktari

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Querying Rich Ontologies by Exploiting the Structure of Data

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

Labinot Bajraktari
Registration Number 1429606

to the Faculty of Informatics

at the TU Wien

Advisor: Dr.Techn. Magdalena Ortiz
Second advisor: Univ.-Prof.Dr. Stefan Szeider

The dissertation has been reviewed by:

Domenico Lembo

Sebastian Rudolph

Vienna, 12th December, 2019

Labinot Bajraktari

Erklärung zur Verfassung der Arbeit

Labinot Bajraktari
Operngasse 20A/12a

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

Wien, 12. Dezember 2019

Labinot Bajraktari

In dedication
to the Love of my Life, Yllka,
to my mesmerizing boys, Bato *and* Fron,
to my devoted parents, Latife *and* Bislim,
and to my loving siblings, Genzana *and* Granit.

Acknowledgements

Looking back, I realize how lucky I've been to get an advisor like Magdalena Ortiz. For without her guidance and support, this thesis would have never materialized. Particularly I value the positive energy she brought to our countless meetings, and her pinpoint advices on the many challenges I faced in my work. I'm especially grateful for the empathy and help to me and Yllka when we were just two newly made parents. Muchas Gracias Magdalena! My sincere thanks go to Mantas Šimkus, from whom I learned so much and whose company I particularly enjoyed. For all the support, collaboration and conversations, Ačiū Mantas! I also thank Diego Calvanese for hosting me in Free University of Bolzano for the research stay, and members of KRDB group for making me feel welcomed. Notably, I thank Guohui Xiao for his collaboration and fruitful discussions.

My heartfelt appreciation goes to the late father of LogiCS doctoral college (DK) Helmut Veith, for bringing to life a DK that fosters quality research, part of which I had the fortune to belong to. Also, I thank Anna Prianichnikova from the DK, for her extra efforts on making sure that social life at the DK thrives. I'm thankful to all the members of the DK for the nice moments we shared together, and in particular, to my office buddies for making the working environment a pleasure to be in. Especially, I'm fond of Shqiponja Ahmetaj and Martin Diller for the friendship we cultivated during this time.

Big thanks to Nysret Musliu, for his kind support, and for all the warm conversations which helped me keep my sanity and motivation during difficult times. My thanks extend to Driton Statovci for being a friend on whom I could count on. Also, I would like to thank the many friends with whom I share some of the best memories during these years in Vienna, in particular: Enkele Rama, Fjolla Ademaj, Gramoz Goranci, Taulant Berisha, and last but not least, my dear friend Valon Raça.

I am very grateful to my mother for her unconditional love that accompanies me everyday; my father for being a role model and for believing in me; my sister on whom I could always count; and my brother for passing on his stubborn view on chasing one's dreams.

Special thanks go to my sons, Fron and Bato, for stalling the progress of my Ph.D. in the most enjoyable way possible, and for painting my life with some of the best memories a person could wish for. The ultimate gratitude goes to my love Yllka, for showering me with love and support, and for taking the burden of raising our kids during this time, and doing an amazing job at it. Most of all, I am grateful for her genuine smiles which reminded me that the best days are ahead of us. Falemnderit Zemër!

Kurzfassung

Wir leben in aufregenden Zeiten. Unsere Fähigkeit Information zu verarbeiten und zu speichern ist beispiellos in der menschlichen Geschichte und hat unsere Gesellschaft transformiert. Unglücklicherweise ist das Verwalten und Erlangen von Wissen aus heutigen Datenbanken ein kompliziertes Unterfangen für Organisationen, vor allem wegen der eingeschränkten semantischen Bedeutung der gespeicherten Daten.

Eine Lösung für diese Situation versprechen Beschreibungslogiken (Description Logics, DL)—eine Familie ontologischer Sprachen welche reichhaltige Modellierungseigenschaften mit in Logik grundierter Semantik zur Verfügung stellen, welche effektiv genutzt werden können um Wissen zu repräsentieren. Kürzlich ist das Paradigma des ontologiebasierten Datenzugriffs (ontology-based data access, OBDA) aufgekommen als ein Weg zur Verwaltung und Integration traditioneller Datenquellen vermittels der Nutzung von Ontologien. Ein essenzieller Schlussfolgerungsservice in OBDA, und der Fokus dieser Dissertation, ist die ontologievermittelte Anfragebeantwortung (ontology mediated query answering, OMQ-Beantwortung). OMQ-Beantwortung hat im letzten Jahrzehnt viel Aufmerksamkeit erfahren, wobei der Standard-Ansatz die Anfrage-Umschreibung (query rewriting) war. Für ausdrucksstarke Ontologien, d.h., solche, die Disjunktive Operatoren nutzen um Wissen auszudrücken, existieren nur theoretische Algorithmen, die sich nicht für eine Implementierungen eignen. Umschreibungen wie diejenigen für leichtere Ontologien wären wünschenswert, aber diese skalieren nicht, zumindest nicht im üblichen datenunabhängigen Rahmen.

In dieser Dissertation suchen wir nach einem durchführbaren Ansatz für OMQ-Beantwortung in ausdrucksstarken Ontologien, indem die Struktur der Daten genutzt wird zur Lenkung des Schließens der Anfragebeantwortungsalgorithmen. Wir stellen eine generische Beschreibung von ABoxen (der Datenkomponente in DL Wissensbasen) vor, welche wir Profile nennen und benutzen diese in drei verschiedenen Situationen um Schließen zu ermöglichen. In der Ersten stellen wir einen Algorithmus vor, der die Repräsentation der Menge der Modelle für $\mathcal{ALCHI}$ Ontologien kompiliert, was, wie wir zeigen, ausreichend ist zur Beantwortung jeder Anfrage die unter Homomorphismen erhalten bleibt. Wir entwickeln Algorithmen zur Beantwortung verschiedener Anfragesprachen welche vorberechnete Repräsentationen benutzen und schreiben diese um in ASP Programme. In der Zweiten erweitern wir unseren Ansatz auf den ausdrucksstärkeren Rahmen von hybriden Sprachen, welche Ontologien mit Regeln kombinieren. Wir definieren eine neue

Hybridsprache, die wir Clopen nennen und wir stellen, für ein Fragment, das bestimmte Einschränkungen erfüllt, eine praktische Übersetzung in einfache ASP Programme vor, die Profile benutzt. In der Dritten setzten wir Profile zur Optierung eines wolhbekannten Algorithmus zur Anfragebeantwortung für Horn-$\mathcal{SHIQ}$, einer ausrucksstarken Horn DL, ein und zeigen signifikante Verbesserung im Vergleich zum aktuellen Stand der Technik. Am Ende zeigen wir, dass unsere Profile leicht von OBDA Spezifikationen mit in R2RML ausgedrückten Zuordnungen bezogen werden können, was unseren Ansatz in praktischen Umgebungen einsetzbar macht.

Abstract

We live in exciting times; our ability for processing and storing information is unprecedented in human history and has transformed our society. Unfortunately, managing and acquiring knowledge from today's databases is a complicated endeavour for organizations, in large part due to the limited semantic meaning of the stored data.

To remedy the situation one can use *Description Logics* (DL)—a family of ontological languages which provide rich modeling features with semantics grounded in logic that can be used effectively for representing knowledge. Recently, the Ontology-based Data Access (OBDA) paradigm has emerged as a way of managing and integrating traditional data sources through the use of ontologies. An essential reasoning service in OBDA, and the focus of this thesis is *Ontology Mediated Query* (OMQ) answering. OMQ answering has received much attention in the last decade, with the standard approach being query rewriting. For *expressive* ontologies, i.e., those that use the disjunctive operator to express knowledge, only theoretical algorithms not amenable to implementation exist. Rewritings like the ones for lighter ontologies would be desirable, but they don't scale, at least not in the usual data independent setting.

In this thesis, we seek to find a feasible approach for OMQ in expressive ontologies, by utilizing the structure of data for 'guiding' the reasoning of query answering algorithms. We propose a generic description of ABoxes (the data component in DL knowledge bases) by what we call *profiles*, and use them in three different settings to facilitate reasoning. In the first, we propose an algorithm to compile a representation of sets of models for $\mathcal{ALCHI}$ ontologies, which we show is sufficient for answering any query preserved under homomorphisms. We develop algorithms for answering different query languages that make use of the computed representation, and rewrite them into ASP programs. In the second, we extend our approach into the more expressive setting of hybrid languages which combine rules with ontologies. We define a new hybrid language that we call *Clopen*, and provide a practicable translation into plain ASP programs which utilizes profiles for a fragment that satisfies certain restrictions. In the third, we employ profiles for optimizing a well-known query answering algorithm for Horn-$\mathcal{SHIQ}$, an expressive Horn DL, and show significant gains compared to the state of the art algorithm. In the end, we show that our *profiles* can be easily obtained from OBDA specifications with mappings expressed in R2RML, which makes our approach deployable in practical settings.

Contents

CHAPTER 1

Introduction

The advent of the technology for storing and retrieving information was of paramount significance for the shift of our society into the information age. The introduction of relational database theory and relational database management systems propelled the wide adoption of databases, and since then new databases models and query languages have been adopted. Query languages are powerful tools for extracting the 'answers' one seeks from the database. However, writing queries requires a deep understanding of how the data that is stored resembles the application domain. Unfortunately database schemas contain very little semantic meaning of the domain they model, hence transferring and acquiring the knowledge about the domain is a time consuming and challenging task. Moreover, databases see the world as a set of facts. They make the so-called closed world assumption (CWA), i.e., only the information recorded in the database is considered to be true. The lack of tools for expressing knowledge make the addition of inference services on top of databases very cumbersome, typically forcing the developers to encode the knowledge directly into the queries, or in low level application code, resulting in complicated solutions with very limited inferential capabilities that are hard to maintain and interpret.

Around the time of the mass adoption of databases, in the area of *knowledge representation* (a sub area of AI) significant work on approaches for representing knowledge was taking place. The most prominent early approaches were Semantic Networks [Qui67] and Frame-based systems [Min85]. They used simple intuitive graphic representations for describing knowledge. However, they were problematic to interpret since they lacked formal semantics. The work of [Hay81] showed that frames could be mapped to first-order logic (FOL), which increased the interest for logic-based formalisms by the community and ultimately gave rise to Description Logics (DLs) [BCM$^+$03].

DLs are a family of ontological formalisms designed for modeling knowledge in terms of objects (individuals), their classes (concepts) and binary relationships between objects (roles). They are fragments of FOL, with one of the driving principles behind their design

being decidability. The knowledge in DLs is expressed in the form of a *Knowledge Base* (KB) consisting of the *extensional component* known as the ABox and the *intensional component* known as the TBox. In the ABox factual data regarding the objects of the domain are recorded, whereas in the TBox constraints between concepts and roles in the form of terminological statements are given. Since their inception in the 80s, DLs have become indispensable and the language of choice for expressing ontologies. They have been used successfully in several application use cases, such as in health and life sciences [RBG+97, HRG96, McG99], data and information integration [CDL+98, Len02], and natural language processing [Fra94, GBFF91]. Undoubtedly, the pinnacle of DLs success is their adoption by the W3C as the underlying logical formalism for the standardized Web Ontology Language (OWL) of the Semantic Web [OWL09]. The goal of DLs is not only to model the domain but to also allow for automated reasoning, inferring this way new 'knowledge' from the one stated in their knowledge base. They adopt the Open World Assumption (OWA) that allows for modeling of knowledge under the setting of incomplete data, more specifically it states that what is unknown is not necessarily false.

Although DLs allow for rich modeling of domains in their KBs they fall short in providing tools for querying the KB. The only possible way to query the KB in DLs with DL proprietary tools is to ask for instances of a concept or role. While this is helpful, it is not satisfactory as we can not pose queries that join instances in the data arbitrarily, a critical requirement in many use cases. On the other hand, conventional databases offer rich querying capabilities, but virtually no support for modeling knowledge. Moreover their schemas are very rigid when it comes to integrating information residing in different sources. Combination of the two, databases with a mature technology for storing data and rich querying capabilities in one hand, and DLs with their rich modeling constructs and clear cut semantics that allows for sound inferences from the knowledge base on the other hand, gave rise to a new paradigm known as *Ontology-based Data Access* (OBDA) [Len11]. OBDA has three components: the ontology, the data sources, and the mappings which semantically link the data sources with the terms in the ontology.

The benefits of having an ontology on top of the database are manifold, (i) the data is integrated through a common layer, the ontology, which pins down the semantic meaning of the domain of interest, (ii) the ontology adds the benefit of simpler knowledge transfer of the domain of discourse as it is written in a logical language which can be consumed and interpreted unambiguously, (iii) moreover the data is queried using a familiar vocabulary of the ontology, and (iv) the knowledge encoded in the ontology is used to infer new knowledge, providing this way more complete answers to queries than the data stored in the database. Since the introduction of OBDA paradigm, there has been considerable interest from the research community, which has been materialised in practical OBDA systems available today, the most well known being Mastro [CDL+11] and Ontop [BCH+14] which have been successfully deployed in industry.

The focus of this thesis is query answering in the ontological setting; a problem commonly referred to as *Ontology Mediated Query* (OMQ) answering. OMQ answering, in a nutshell,

is the simplification of OBDA (and a prerequisite thereof) that omits the mapping layer: where one assumes that the data is already given in an ABox, that is, a set of facts over the ontology vocabulary.

1.1 Motivation

Querying a database with an ontology in between is desirable. However, it introduces significant computational challenges. Getting the answers to a query amounts to logical inference, i.e., we have to check if the query is true in all possible situations (interpretations) in which the ontology and the database are interpreted consistently. We call such interpretations models. The challenges in query answering arise from two things: first, the sizes of the models that need to be considered are usually quite large, and in many cases infinite due to the use of existential quantification; second, the number of models we have to consider is in principle infinite for any knowledge base, this spurs from the ability to add unrelated information to the models while still preserving modelhood.

The large (infinite) size of the models is unavoidable in any DL. Still techniques that tame this property have been developed and successfully implemented for query answering in lightweight DLs. Lightweight DLs are Horn DLs that are characterized with the nice universal model property, i.e. for any given Horn DL KB we can devise a model that is sufficient for answering any query over it. On the other hand, in the case of expressive DLs, i.e. those that extend $\mathcal{ALC}$ (the most basic propositionally complete DL), even if we restrict to only the models with 'relevant' information, due to the use of disjunction we are forced to reason by cases, i.e., to consider different models for different cases.

Adopting techniques that might work well in practice for expressive DLs is not so obvious. Despite the fact that answering OMQs has received much attention in the last decade, the big gap between practicable algorithms for lightweight DLs, that are supported by implemented reasoners, and purely theoretical algorithms for expressive ontologies that are not amenable to implementation, has only increased.

Example 1. *Consider the following university project management KB shown in the table below. This example follows closely the example given by [Sch93]. The KB contains a set of facts related to researches and a set of terminological statements.*

From the facts given we notice four individuals: Ben, John, Mary, and Paul who are researchers. Moreover we see that John supervises both Mary and Ben, and Mary has Ben as a co-author in some work, and Ben has Paul as a co-author. The statement (a) *says that every researcher is an external staff or an university staff, whereas* (b) *states that nobody can be both external and university staff at the same time. The statement* (c) *states that anybody that supervises someone must be an university staff.*

$\mathsf{Researcher}(ben)$.	$\mathsf{Researcher}(john)$.	$\mathsf{Researcher}(mary)$.
$\mathsf{Researcher}(paul)$.	$\mathsf{ExternalStaff}(mary)$.	$\mathsf{UniveristyStaff}(paul)$.
$\mathsf{supervises}(john, mary)$.	$\mathsf{supervises}(john, ben)$.	$\mathsf{coAuthored}(mary, ben)$.
$\mathsf{coAuthored}(ben, paul)$.		

$$(a) \quad \mathsf{Researcher} \sqsubseteq \mathsf{ExternalStaff} \sqcup \mathsf{UniversityStaff}.$$
$$(b) \quad \mathsf{ExternalStaff} \sqcap \mathsf{UniversityStaff} \sqsubseteq \bot.$$
$$(c) \quad \exists \mathsf{supervises}.\top \sqsubseteq \mathsf{UniversityStaff}.$$

Table 1.1: An example ontology.

Now consider the following query through which we seek to find all the individuals that supervise some external staff that has co-authored with some member from the university staff:

$$\mathsf{q}(x) \leftarrow \exists y, z.\mathsf{supervises}(x, y), \mathsf{ExternalStaff}(y), \\ \mathsf{coAuthored}(y, z), \mathsf{UniveristyStaff}(z).$$

We are interested in getting the answers to the query that satisfy both the facts and the terminological statements. Note that in every possible scenario (interpretation) John has to be an university staff due to (c), whereas due to the facts in the database Mary is a member of external staff and Paul is a member of university staff. Ben on the other hand, depending on the interpretation will be either member of the external or university staff. In Figure 1.1 two viable interpretations of the given ontology are represented as directed graphs. The concept and role names are shortened for presentation purposes.

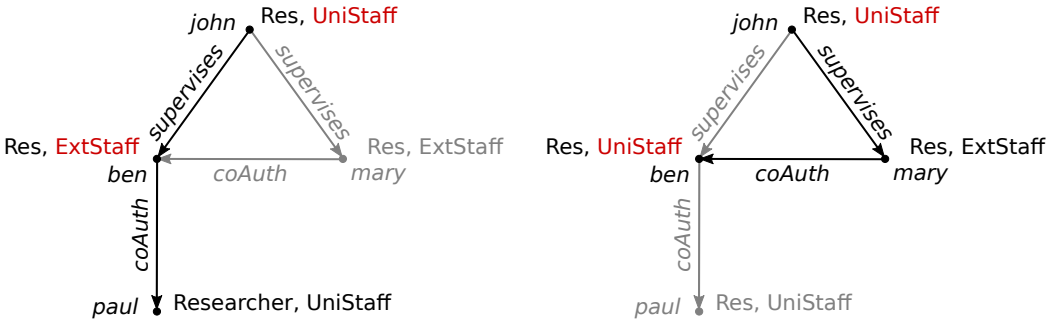


Figure 1.1: Representation of interpretations for the ontology in Table 1.1.

Note that the labels marked in red are inferred by the ontology while the rest are given as facts. If we also view the query as a directed graph, we can quickly check if there is an answer to the given query by checking if the graph of the query can be homomorphically mapped against the graph of the interpretation. In both interpretations we have given

such possible mappings via the edges and nodes colored in black. As can be seen, John is an answer to the given query in both interpretations. In fact, it is an answer in every interpretation that models the knowledge base, since in each of them we find at least the information in one of two intepretations above.

Cases like the one presented in the example above where disjunction is needed seem natural, and working towards algorithms that work for real-world settings is important in its own right. However, the high complexity of the query answering even for the most basic query languages has discouraged research towards practicable algorithms. To the best of our knowledge, to date there are no implementations of query answering algorithms for expressive DLs.

1.2 State of the Art

OMQ answering is one of the core reasoning services in the ontological setting, and as such, it has received much attention from the research community. There are two main approaches to OMQ answering problem. The first one is to saturate the knowledge base with all relevant inferences from the ontology before posing queries against it. However, this is widely regarded as impractical since it can typically blow up the size of the data which is presumed to be large to begin with. The second one is *query rewriting* and is the current standard approach in the field, where an OMQ $(\mathcal{T}, q)$ comprising of a DL TBox $\mathcal{T}$ and a query q in a standard language (e.g., conjunctive queries (CQs)) is written into a new query q' in a target query language. Obtaining q' may be costly, but it is independent of a concrete dataset (ABox). Then q' can be evaluated over any ABox using existing engines for the target language, typically SQL and DATALOG for more expressive DLs. We believe that a significant source of hardness comes from the data independent approaches to query rewriting which rewrite the query taking into account any database, including databases that would never be encountered in practice.

One usually distinguishes between two types of DLs, those that can express disjunction, typically known as expressive DLs, and those that can not, commonly referred to as Horn DLs.

Horn DLs enjoy the universal model property, which makes query answering more tameable. Moreover answering conjunctive queries (CQs)- a key class of queries, is known to be tractable in data complexity; an important feature that has made Horn DLs potential candidates for practical adaptation. Extensive research has been undertaken on choosing the 'right' constructors for designing DLs that are scalable in practical application use cases, from which the most notable are *DL-Lite* and $\mathcal{EL}$ families. *DL-Lite* was designed with low data complexity in mind, which for CQ answering coincides with the data complexity of answering CQs over plain databases. Consequently, *DL-Lite* was adopted as the underlying logic of OWL2 QL profile,[1] and efficient algorithms for query

[1] www.w3.org/TR/owl2-profiles/#OWL_2_QL

rewriting have been designed and successfully incorporated in the most prominent OBDA systems to date, like Mastro and Ontop. On the other hand, $\mathcal{EL}$ [BBL05, BLB08] is the underlying logc for the OWL2 EL profile,[2] which was designed to accommodate conjunction of concepts and existential quantifiers features used extensively in biomedical ontologies such as the National Cancer Institute (NCI) Ontology, [3] and SNOMED CT [4]. Query answering in $\mathcal{EL}$ has higher data complexity than *DL-Lite*, however is still in PTime. This is because one can encode reachability in $\mathcal{EL}$ TBoxes hence not every query is First-Order (FO) rewritable. However, attempts to implement algorithms that do CQ answering in $\mathcal{EL}$ exist, the authors of [LTW09] show how one can utilise RDBMS to answer FO-rewritable CQs over a variant of $\mathcal{EL}$ KBs, whereas in the works of [SM15] a CQ answering algorithm over a more expressive variant of $\mathcal{EL}$ that looks practicable for many cases is given. The algorithm in [SM15] computes certain consequences of the given KB via a Datalog program implemented in RDFox [MNP+14] and then evaluates the query over it.

Another very popular Horn DL worth mentioning is Horn-$\mathcal{SHIQ}$, which can be seen as the Horn fragment of OWL Lite. Although it's Horn it is still quite expressive, and in addition to supporting all the features of *DL-Lite* and $\mathcal{EL}$, it also supports transitive roles and some number restrictions. Horn-$\mathcal{SHIQ}$ is relatively well understood, and there are existing reasoners for traditional reasoning problems like satisfiability and classification [Kaz09]. As for query answering, unlike *DL-Lite* it is in general not possible to reduce query answering in the presence of a Horn-$\mathcal{SHIQ}$ ontology to plain SQL query evaluation. Some alternative approaches have been proposed in order to make OBDA with Horn-$\mathcal{SHIQ}$ feasible on top of existing database technologies. For example, to rewrite (exactly or approximately) an ontology into a weaker DLs [LWW07, RPZ10], or to compile some of the extra expressivity into the mappings [BCS+16]. Another possibility is to compile the query and the ontology into a more expressive query language than SQL, like Datalog, as done in the Clipper system [EOv+12b]. Clipper can handle realistic ontologies and queries, despite being a relatively simple prototype. It is among the richest query answering engines for Horn DLs, and has inspired recent adaptations [LMTV19, CDK19]. However, Clipper has stark limitations, and there are many ontologies that it cannot process in a reasonable time [CGK19a]. This is largely due to the ABox independence of the saturation step it employs: some axioms that could be omitted for simpler tasks like *classification* [Kaz09], must be inferred by Clipper since they may be made relevant by the assertions in some input ABox.

In the case of non-Horn DLs, the use of disjunction plays a pivotal role in the hardness of these logics, since it forces reasoning by cases, causing this way the loss of the universal model property. State-of-the-art reasoners for expressive DLs can handle very large ontologies (e.g., Pellet [SPG+07], HermiT [GHM+14], Konclude [SLG14]), but they usually use the tableau technique which aims at deciding if some model exists, and it

[2] www.w3.org/TR/owl2-profiles/#OWL_2_EL
[3] https://bioportal.bioontology.org/ontologies/NCIT
[4] https://bioportal.bioontology.org/ontologies/SNOMEDCT

remains unclear whether they could be adapted for OMQ answering. Adopting tableau algorithms even for the task of instance checking is hard, as authors [HM05] rightfully profess; developing of efficient tableaux algorithms is hard due to the fact that tableaux procedures are non-deterministic in nature and for retrieving the individuals that belong to a given concept name one may be forced to run the tableaux procedure for each individual. Modern tableaux reasoners such as Konclude [SLG14] use advanced caching techniques to overcome some of the difficulties for answering instance queries, however uplifting the tableaux algorithms to richer query languages beyond instance queries seems an implausible task to undertake. One of the key features that makes the query answering problem hard to tackle is the high data complexity of these logics. It is known that the data complexity of the problem of answering instance or conjunctive queries in DLs extending $\mathcal{ALC}$ is coNP-hard [Sch93]. Currently there is a wide gap between Horn DLs in one side, for which as mentioned algorithms have been developed, improved, and implemented in reasoners, and expressive DLs containing $\mathcal{ALC}$ on the other side, in the case of which most of the research on OMQ answering has had theory-oriented goals, like understanding decidability and worst-case complexity [Lut08]. Many algorithms employ tools that are not amenable to implementation, like automata [CDL08, CEO14]. Rewritings have been proposed (e.g.,[BtCLW14, AOS16, EOv12a]) but they appear impracticable, and to our knowledge, they have not led to implementation attempts. A rewriting into Datalog for $\mathcal{SHIQ}$ was implemented a decade ago in the KAON2 reasoner, but only for instance queries. A published extension to CQs did not yield a data-independent rewriting and was never implemented [HMS04].

In summary, up to the writing of this thesis, when it comes to OMQ algorithms, the standard technique is the data independent query rewriting, and most of the work done for expressive DLs was confined to addressing complexity theory goals, with no work done to push for query answering algorithms beyond instance queries. In the case of expressive Horn DLs such as Horn-$\mathcal{SHIQ}$ although an algorithm for CQ answering has been implemented in Clipper, there are still many cases in which it exhibits exponential behavior. We believe that a more constrained approach to the data independent approach is important for working towards obtaining practical algorithms for query answering expressive DLs.

1.3 Goals of the Thesis

This thesis is motivated by the following research question:

(R) *Can we leverage the structure of datasets to guide the reasoning process, in order to obtain algorithms for query answering in expressive DLs that have the potential for implementation?*

Guided by this question we define three main goals:

(G1) *Devise representations of data whose size is reasonably small compared to the datasets they cover, and easy to compute in real-world settings.*

We want to focus on representations of datasets that are practicable to deploy in real-world applications, hence the requirement for an easy computation in one hand, and to be small compared to the original data set on the other hand. These requirements increase the potential of techniques built on top of them for scaling in practice. In particular, we are interested in representations that can be obtained from OBDA specifications.

(G2) *Develop reasoning techniques for answering queries in the presence of expressive ontologies, parametrized and guided by suitable representations of the datasets of interest.*

We want to capture enough information about the models so that we can tackle query answering for different query languages that have so far been considered infeasible for expressive DLs. To achieve this, we will parametrize the algorithms by representations of data that are in line with the goal (G1). The parametrization has the purpose of separating the expensive reasoning about the ontology from reasoning with the data. This makes the computation more modular where the core ontological reasoning is reusable over different data sets.

(G3) *The algorithms should have the potential for implementation, and preferably they should reuse existing technologies like* Datalog *systems and ASP solvers.*

We seek to obtain practicable algorithms, hence a proof of concept is needed in order to test their potential for adoption in practice.

1.4 Contributions

To address the goals of the thesis, we focused on obtaining a representation that characterizes the structure of the data, and on developing algorithms that utilize the representation in order to answer different query languages for the expressive DL $\mathcal{ALCHI}$. Moreover, we investigated our approach in a more complex setting of hybrid languages that combine rules with an expressive ontology, and devised a practical translation of hybrid programs into plain *Answer Set Programming* (ASP).We also explored how our representation could be utilized to improve the efficiency of query answering for expressive Horn DLs, and provide an optimization technique for a well know query answering algorithm for Horn-$\mathcal{SHIQ}$. Lastly, as a real-world setting for obtaining the representation of the data, we considered OBDA specifications where the mapping layer is defined over relational databases (R2RML) and give an algorithm for this.

Our contributions can be summarized as follows:

(C1) We define a general way of representing the structure of ABoxes through what we call a set of *profiles* which encode the configuration of concepts and roles an individual in the ABox may participate in. We provide empirical evidence that our representations are typically small compared to the datasets in most cases.

(C2) We give a simple algorithm which given an OBDA specification with R2RML mappings, extracts the *profiles* (as defined in (C1)) that cover all possible ABoxes

that can result from evaluation of the mapping layer in the given OBDA specification over any concrete database. Considering the wide adoption of relational databases, this setting was considered of particular interest to address. This contribution together with (C1) address the first goal of the thesis.

(C3) We utilize *profiles* to obtain model representations for answering queries that are preserved under homomorphism for $\mathcal{ALCHI}$, and show that in most cases our model representations are feasible to compute and small compared to the ones we would have to consider in a data-independent approach. This contribution is the cornerstone for fulfilling the second goal of the thesis, as all our algorithms for reasoning in expressive DLs are parametrised by the computed model representation which in itself encapsulates a good part of expensive ontological reasoning. Computing the model representations from a set of profiles makes our reasoning technique modular, as they need to be computed only once, and are reusable for any dataset covered by the set of profiles. Moreover the model representations can: (i) be computed offline, an important feature especially for large and complex ontologies, and (ii) can be computed incrementally, i.e., if the structure of the data changes and our set of profiles do not cover certain cases, incorporating new profiles and recomputing the model representation can be done incrementally on top of the previous ones.

(C4) We present concrete algorithms that depend solely on the computed model representation in (C3) for answering *instance queries*, *reachability queries*, and *semi-full conjunctive queries with reachability atoms*. These algorithms rewrite the queries into ASP programs. We implemented the algorithms into a proof of concept Mod4Q which uses efficient ASP solvers to evaluate the rewritten queries, and give experimental evidence for successful evaluation of *instance queries* and *reachability queries* over all ontologies for which computation of model representation was feasible. The implemented and tested algorithms address all goals of the thesis.

(C5) We present *Clopen* knowledge bases, a rich hybrid language that generalizes the prominent language of Rosati r-hybrid, for which no implementations exist. We identify a useful fragment of Clopen KBs (CKBs) which we call separable CBKs, and provide an algorithm that translates separable CKBs to plain ASP programs using our model representation from (C3). We remark that separable CKBs still generalize r-hybrid KBs. Moreover, we have incorporated our translation of CKBs in our Mod4Q and tested against a real-world ontology with programs realizing interesting use cases. The implemented translation addresses all the goals of the thesis.

(C6) Lastly, we show that *profiles* can also be utilized for optimizing an algorithm for query answering in an expressive Horn DL. We use profiles to guide the derivation of new consequences in the TBox inference calculus used by a state of the art CQ answering algorithm for Horn-$\mathcal{SHIQ}$ [EOv12a]. The inference calculus in [EOv12a] is already decoupled from other steps of the query answering algorithm and can be

computed offline. However, it easily leads in an exponential blow up for many cases. To address the goals of the thesis, we constraint the derivation of redundant axioms by guiding the derivation of new axiom by the inference calculus with *activators* obtained from our *profiles*. We implemented our approach in Clipper, the system that uses the algorithm subject to our optimization, and evaluated it against the original version over a well-known ontology repository with significant gains.

The work on this thesis has been published in the following peer reviewed venues given in chronological order:

[BOS17] Labinot Bajraktari, Magdalena Ortiz, and Mantas Simkus.
Clopen Knowledge Bases: Combining Description Logics and Answer Set Programming.
In Proceedings of Thirtieth Description Logic Workshop, DL2017, July 18-21, Montpellier, France., 2017.

[BOS18a] Labinot Bajraktari, Magdalena Ortiz, and Mantas Simkus.
Combining Rules and Ontologies into Clopen Knowledge Bases.
In Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018, pages 1728–1735, 2018.

[BOS18b] Labinot Bajraktari, Magdalena Ortiz, and Mantas Simkus.
Compiling Model Representations for Querying Large ABoxes in Expressive DLs.
In Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden., pages 1691–1698, 2018.

[BOX19] Labinot Bajraktari, Magdalena Ortiz, and Guohui Xiao.
Optimizing Horn-SHIQ reasoning for OBDA.
To appear: In Proceedings of the Eighteenth International Semantic Web Conference, ISWC 2019, October 26-30, 2019, Auckland, New Zealand., 2019.

Contribution (C1) is reflected in all the publications, however it is explained more in depth in [BOS18b]. Contribution (C2) is presented in [BOX19], whereas contributions (C3,C4) are presented in [BOS18b]. Contribution (C5) is presented in [BOS18a] and [BOS17], and lastly contribution (C6) is presented in [BOX19].

1.5 Structure of the Thesis

This thesis consists of eight chapters. In Chapter 1 (this chapter), we gave a basic description of the field and motivated the work done in the thesis, as well as presented the goals and an overview of the key results achieved. The remainder of this thesis is followed by Chapter 2 in which we give the fundamental concepts used through the thesis,

with a focus on Description Logics and Answer Set Programming. This is followed by five core chapters of this thesis which describe the following work:

- In Chapter 3, we describe our representation of ABoxes via profiles and give a model compilation algorithm for $\mathcal{ALCHI}$ based on types. We prove the soundness and completeness of model compilation algorithm, i.e., that for any model $\mathcal{I}$ of the knowledge base one can construct a model from our computed representation that can be homomorphically mapped to $\mathcal{I}$. In this chapter we also give an overview of our proof of concept implementation Mod4Q including the ontologies used to test the algorithms in this thesis, and report the results on the sizes of profiles and the feasibility of model compilation for those ontologies. Contributions (C1) and (C3) are described in this chapter.

- In Chapter 4, we show that the computed model representation from the previous chapter is sufficient for answering any query preserved under homomorphism. Concrete algorithms for query answering that depend solely on the model compilation for reasoning are given. We report encouraging results from the evaluation of algorithms for answering *instance* and *reachability queries* against a range of ontologies. Contribution (C4) is described in this chapter.

- In Chapter 5, we define a new rich hybrid language *clopen* that combines rules and ontologies. We identify a restricted version of clopen knowledge bases (CKBs), we call *separable* CKBs which generalize some of the most prominent hybrid languages, and show how utilising the computed model representations in Chapter 2 one can obtain a plain ASP program with a potential for implementation in practice. We give a practical use cases and report on the positive results obtained for an expressive real-world ontology evaluated against real-world data. Contribution (C5) is described in this chapter.

- In Chapter 6, we show that profiles can be employed in a different context. Instead of using them to compile model representations, we utilise them for optimizing a very well known CQ answering algorithm for Horn-$\mathcal{SHIQ}$ by [EOv12a]. Furthermore we report very encouraging results from the comparison of our implemented optimization in Clipper versus the original version. Contribution (C6) is described in this chapter.

- In Chapter 7, we show that our chosen representation of ABoxes via profiles makes sense by giving an approach how one can obtain them from OBDA specifications. We give an analysis of three OBDA mappings with encouraging outlook for deploying them in practice. Contribution (C2) is described in this chapter.

Finally, we close this thesis with Chapter 8 in which we give a summary of the work that has been carried out, along with conclusions and future research directions we envision.

CHAPTER 2

Preliminaries

In this chapter, we present the foundations upon which this thesis builds on. We assume that the reader is equipped with basic knowledge of *First Order Logic* (FOL).

In Section 2.1 we start by giving an introduction to *Description Logics* (DLs) in general, and specifically we define the syntax and semantics of particular DLs of interest ($\mathcal{ALCH}$, $\mathcal{ALCHI}$, and Horn-$\mathcal{SHIQ}$). Further, we give a brief overview of the reasoning tasks with a focus on query answering in the presence of ontologies. For a more thorough introduction into DLs, the reader might want to check [BCM$^+$03], whereas for a lighter introduction to the subject one might consider [Rud11] and [KSH12].

In Section 2.2, we define *Answer Set Programming* (ASP), a declarative problem-solving approach that resulted from extensive work of the logic programming, knowledge representation, and constraint satisfaction research communities. ASP is our paradigm of choice for encoding the query answering algorithms presented in this thesis, and evaluating them with efficient ASP solvers. We open the section with an introduction to ASP, followed by a formal definition of their syntax and semantics, and a brief overview of reasoning with answer sets. For a more in-depth introduction to ASP, we point the reader to [EIK09], whereas for a more high-level read, one may consider [BET11].

2.1 Description Logic

Description logics (DLs)[BHLS17] are a family of knowledge representation languages that can be used to represent knowledge of an application domain in a structured and unambiguous way. They are fragments of FOL designed with a key principle in mind, decidability. DLs have found their way into a range of different application domains such as data and information integration [CDL$^+$98, Len02], health and life sciences [RBG$^+$97, HRG96, McG99], natural language processing [Fra94, GBFF91], and the most important one being their adoption as the underlying logical formalism for the

Semantic Web [DFH11], all of which have sparked a renewed interest in DLs from the research community since their inception in the 80s.

In DLs the domain knowledge is modeled as a *Knowledge Base* (KB), which represents the knowledge of the chosen domain in terms of relationships between concepts, roles, and individual names in the domain of interest. Typically individual names represent objects in the domain of discourse, e.g., if we would like to model a domain of people and their friendship as well as family relationships we may use their names as individuals names like *bato*, *fron* to identify persons *Bato* and *Fron*. We use concepts to represent objects that belong to a particular class, such as the concept child which represents all children. Then we use roles to represent binary relationships between objects, such as the role brotherOf which represents all pairs of objects that are in the relation brother of with one another.

So far everything we have said about DLs can be represented by a simple database containing unary and binary relations. In *databases*, one makes the *closed world assumption* (CWA), that only the recorded facts in the database are true, and the rest (the unknown) are assumed to be false. This is fine in many application use cases in which we consider to have complete information, e.g., consider an insurance database, we ask may ask "is person named *Bato* insured?", in which case one naturally expects the answer to be recorded in the database. However, there are application use cases where CWA is not well suited. Consider for example a hospital database in which we store the treatment information for each patient, we may want to ask the question, "will patient *Fron* have some reaction to a certain prescribed drug", in CWA the answer is no, however for these types of question a more appropriate answer would be 'don't know'. DLs make the open world assumption (OWA) which states that what is unknown is not necessarily untrue. However, the point of adopting OWA by DLs is not to answer to each question for which we don't have recorded facts in the database with 'don't know', but to phrase knowledge in logical axioms coupled with the database which would enable for automatic inferences that help us to complete answers to various questions as the one asked above. In the example above an ontology encoding the knowledge about drugs used may come in handy to answer the question posed.

In DLs, the KB is partitioned into two sets: the set of axioms in one hand which represent the terminological knowledge commonly referred to as the TBox, and the set of assertions on the other hand that represent the factual knowledge about the domain of discourse typically referred to as the ABox. The facts in ABox state what is known about the individuals in our domain, i.e., which classes they belong to and the relationships between them. ABox assertions that state concept memberships look like the following:

$$\mathsf{Female}(yllka)$$

which states that the individual *yllka* belongs to the class Female, in DL jargon we say *yllka* is an instance of the concept Female. Role membership assertions in ABox state the relationship between two individuals:

$$\mathsf{siblingOf}(fron, bato)$$

which is read *fron* is a sibling of *bato*. Note that the first component in the role memberships is the subject of the sentence, the role is the predicate, and the second component is the object of the sentence. Another type of assertion, is the so called individual (in)equality assertion, which states that two individual names do (not) represent the same object. This has to do with the so called *unique name assumption* (UNA) which is natural in the CWA where each object is treated to be different from others, however in DLs by default such assumption is not made, and in case we want to state that two individual names correspond to different individuals, we write:

$$bato \not\approx fron$$

In our daily discourse we usually make UNA assumpition, and there are riddles that illustrate this, consider the following for example: "Two fathers and their two sons went fishing together. Each of them caught a fish. On their way home, they did not lose any fish, and yet when they arrived, they had only three fish. How could this happen?". To us this doesn't make sense because by default we assume that there were 4 different individuals (i.e. we make UNA assumption), however in this case we have three persons, a grandfather who went fishing with his son and his grandson, and they each caught a fish. Note that in many application use cases UNA is preferred, and we have chosen to adopt UNA in our work since it is natural in the setting of OMQ answering [BO15] where ABoxes are viewed as data repositories.

The other part of the KB, the set of terminological axioms in TBox are used to specify subsumption or equivalence relationships between concepts or roles expressions, where concept and role expressions are built using different concept, respectively role constructors. For example, when we like to state that all boys are children, we state that all instances of concept Boy are also instances of concept Child through the following inclusion axiom:

$$\mathsf{Boy} \sqsubseteq \mathsf{Child}$$

One can also use equivalence relationships to define that two concept expressions are the same, i.e. they have the same instances. Consider for example the following equivalence axiom:

$$\mathsf{Mother} \equiv \mathsf{Person} \sqcap \mathsf{Female} \sqcap \exists \mathsf{hasChild}.\top$$

which states that all instances of the concept Mother belong also to the concepts Person, Female and $\exists$hasChild.$\top$ which represents individuals that should be in at least one relationship hasChild with some individual (notice the special concept $\top$ which represents

all individuals in the domain), and vice versa. The right hand side read as a sentence represents all persons of gender female that have some child. Note that in the right hand side we have used a concept expression which in this case contains two concept constructors, the conjunction $\sqcap$ which allows us to intersect two different concepts, and the *existential restriction* constructor which allows us to restrict the membership only to the individuals that have to take part in at least one role membership with another individual instance of a certain concept. Another core concept constructor of DLs is the *universal role restriction* constructor, an example of which might look like the following:

$$\forall \mathsf{isSiblingOf}.\mathsf{Male}$$

which restricts the membership to individuals who can not occur in a role relationship with an individual that does not belong to the stated concept, in this case we model all individuals who can only have siblings that are male.

Apart from stating relationships between concept expressions one often is interested in stating the relationships between role expressions, e.g.:

$$\mathsf{isBrotherOf} \sqsubseteq \mathsf{isSiblingOf}$$

which states that all pairs of individuals in isBrotherOf relationship are also in isSiblingOf relationship. One of the most common role constructors used in role expressions is the *inverse role* constructor, e.g.:

$$\mathsf{isParentOf} \sqsubseteq \mathsf{isChildOf}^{-}$$

which states that all pairs of individuals in isParentOf relationship are in inverse relationship with pairs of individuals in isChildOf relationship.

The multitude of combinations between concept and role constructors give rise to different DLs with different expressive power, hence different computational properties. This represents an interesting feature of DLs, which allow for the targeted design of DLs for different use cases and application domains by balancing between expressivity and the expense in complexity. Some of the most well known DL logics are *DL-Lite*, $\mathcal{EL}$ and $\mathcal{RL}$ which have been adopted by W3C for different semantic web entailment regimes. The above discussed elements of DLs are only a part of available constructors and elements, where the purpose was to give a high-level intuition on DLs as a formalism. In the next subsection, we give formal definitions of basic notions as well as syntax and semantics of DLs of interest in this thesis.

2.1.1 Basic Definitions, Syntax and Semantics

Previously we stated that the domain knowledge in DLs is modelled in terms of relationships between concept names, role names, and individual names in the domain of interest, these three constitute the DL vocabulary.

Definition 1 (DL Vocabulary)**.** *DL vocabulary is defined as a triple of the following countably infinite pairwise disjoint sets used for modelling some domain of interest:*

- $\mathsf{N_C}$ *the set of* concept names *which denote primitive classes, and*
- $\mathsf{N_R}$ *the set of* role names *which denote primitive binary roles, and*
- $\mathsf{N_I}$ *the set of* individuals *which denote objects in the domain.*

Throughout this thesis we assume that $\mathsf{N_C}$ additionally includes two special concept names $\bot$ and $\top$, known as *bottom* and *top* concept.

We use the following naming convention in this thesis:

- concept names starts with an upper letter and uses the math sans serif font $\mathsf{Concept}$.
- role names use math sans serif font with camel case $\mathsf{hasRole}$.
- individuals start with italic lower case roman font *ergosum*.

Concepts and roles are important building blocks in DLs. They are expressions defined recursively over different concept and role constructors available in certain DL. Depending on the DL we are talking about, the concept and role constructors at the disposal for forming concept and role expressions differ, and are the primary source of expressivity of reasoning in DLs. The trade-off between the computational cost and the gain in expressivity specific combination of constructors may bear is a subject of multiple studies throughout the field. While in one hand certain application use cases require for certain constructors to be present, the theoretical study of the same shades light on the computational efforts one may need for different reasoning task in the DL at hand.

Definition 2 (Concepts and Roles)**.** *Given a DL $\mathcal{L}$:*

- *Concepts are defined from concept names inductively with the concept constructors from Table 2.1 available in $\mathcal{L}$, and*
- *Roles are defined from role names inductively with the role constructors from Table 2.1 available in $\mathcal{L}$.*

If $r \in \mathsf{N_R}$, then r and r^- are roles; the set of all roles is denoted $\overline{\mathsf{N_R}}$. For readability, r^- stands for s whenever $r = s^-$ for $s \in \mathsf{N_R}$.

DLs use a naming scheme which reveals the constructors certain DL uses, e.g., the letter $\mathcal{H}$ used in the name of a DL represents the availability of role inclusion constructor, $\mathcal{I}$ the availability of the inverse role constructor, $\mathcal{O}$ the availability of the nominal concept constructor, $\mathcal{Q}$ the availability of qualified number restriction constructor, $\mathcal{S}$ denotes all the constructors available in the DL $\mathcal{ALC}$ with the addition of the transitive role

constructor, and $\mathcal{R}$ the availability of complex role inclusions of the form $r \circ \ldots \circ s \sqsubseteq t$ and further axioms that allow, for example to declare reflexive, irreflexive, and antisymmetric roles. The aforementioned list is by no means exhaustive but we listed some of the well known constructors to give an idea on the myriad of DLs one can obtain by combining different available constructors.

In DLs knowledge is expressed via axioms, which are divided into two groups: terminological axioms (TBox axioms for short), and assertion axioms (assertions for short).

Definition 3 (Assertion Axioms)**.** *Assertion axioms take one of the following forms:*

- $C(a)$ *where* C *is a concept and* $a \in \mathsf{N_I}$*, known as concept membership assertions.*
- $r(a, b)$ *where* r *is a role and* $a, b \in \mathsf{N_I}$*, known as role membership assertions.*

Sometimes inequality assertions of the form $a \not\approx b$ are considered. We omit them in this thesis because we make the UNA, hence they are meaningless in this thesis.

For any assertion of the form $C(a)$ and $r(a, b)$ we always assume that $C \in \mathsf{N_C}$, $r \in \mathsf{N_R}$, i.e., we always assert membership of individuals only in concept and role names.

TBox axioms are further divided into two sub groups *General Concept Inclusions* axioms (GCIs) and *Role Inclusion Axioms* (RIAs). Depending on the concrete DL, different terminological axioms can be used.

Definition 4 (GCIs and RIAs)**.** *A TBox axiom is:*

- *a general concept inclusions axiom that takes the following form:* $C \sqsubseteq D$ *where* C *and* D *are concepts, or*
- *a role inclusion axioms that takes the following form:* $r \sqsubseteq s$ *where* r *and* s *are roles.*

Next we define DL ABoxes, TBoxes and KBs.

Definition 5 (DL KBs, ABox and TBox)**.** *Given a DL* $\mathcal{L}$*, a* $\mathcal{L}$ *knowledge base is a pair* $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ *where:*

- $\mathcal{A}$ *is a finite set of assertion axioms in* $\mathcal{L}$*, known as the ABox, and*
- $\mathcal{T}$ *is a finite set of terminological axioms in* $\mathcal{L}$*, known as the TBox.*

For a given ABox $\mathcal{A}$ we define the set of individuals occurring in $\mathcal{A}$ with $\mathsf{N_I}(\mathcal{A})$.

The semantics of DLs is given in terms of interpretations $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ where $\Delta^{\mathcal{I}}$ is the domain of $\mathcal{I}$ a non-empty set defined over $\mathsf{N_I}$, and $\cdot^{\mathcal{I}}$ is a mapping function which maps each individual name $\mathsf{N_I}$ to an element in $\Delta^{\mathcal{I}}$, each concept name in $\mathsf{N_C}$ to a subset of elements $\Delta^{\mathcal{I}}$, and each role name in $\mathsf{N_R}$ to a subset of set of pairs of elements in the domain $(\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}})$.

Definition 6 (Interpretation)**.** *An* interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *consists of a non-empty* domain $\Delta^{\mathcal{I}}$ *and a* valuation function $\cdot^{\mathcal{I}}$ *that maps:*

- *each* $a \in \mathsf{N_I}$ *to an element* $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$,
- *each* $A \in \mathsf{N_C}$ *to a set* $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$,
- *each* $r \in \mathsf{N_R}$ *to a set* $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$,
- $\top$ *to the domain* $\Delta^{\mathcal{I}}$, *and*
- $\bot$ *to the empty set* $\emptyset$.

We recall that we make the UNA throughout the thesis, i.e. for any interpretation $\mathcal{I}$ and pair $a, b \in \mathsf{N_I}$ s.t. $a \neq b$, we require $a^{\mathcal{I}} \neq b^{\mathcal{I}}$.

Next we define the model of a TBox, an ABox and a KB.

Definition 7 (TBox, ABox, KB models)**.** *An interpretation* $\mathcal{I}$ *is a* model *of:*

- *a TBox* $\mathcal{T}$ *if*

  for each concept inclusion $C \sqsubseteq D \in \mathcal{T}$, $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, *and*
  for each role inclusion $R \sqsubseteq S \in \mathcal{T}$, $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$, *and*
  for each transitivity axiom $trans(R) \in \mathcal{T}$, $R^{\mathcal{I}} \circ R^{\mathcal{I}} \subseteq R^{\mathcal{I}}$.

- *an ABox* $\mathcal{A}$ *if*

  for each assertion $C(a) \in \mathcal{A}$, $a^{\mathcal{I}} \in C^{\mathcal{I}}$, *and*
  for each assertion $r(a, b) \in \mathcal{A}$, $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$.

- *a KB* $\mathcal{K}$ *if* $\mathcal{I}$ *is a model of* $\mathcal{T}$ *and* $\mathcal{A}$.

We say that a KB, TBox or an ABox is consistent (or, *satisfiable*) if it has a model. For an axiom α and a TBox $\mathcal{T}$ we say that α is entailed by (or a logical consequence of) $\mathcal{T}$ ($\mathcal{T} \models \alpha$), if for every model $\mathcal{I}$ of $\mathcal{T}$, $\mathcal{I}$ satisfies α.

For a TBox $\mathcal{T}$, we use $\sqsubseteq^*_{\mathcal{T}}$ for the transitive closure of the relation $\{(r, s) \mid r \sqsubseteq s \in \mathcal{T}$ or $r^- \sqsubseteq s^- \in \mathcal{T}\} \cup \{(r, r) | r \in \overline{\mathsf{N_R}}\}$.

Definition 8 (Homomorphism from $\mathcal{I}$ to $\mathcal{J}$)**.** *Let* $\mathcal{I}$ *and* $\mathcal{J}$ *be two interpretations.A* homomorphism $h : \mathcal{I} \longmapsto \mathcal{J}$ *is a mapping from* $\Delta^{\mathcal{I}}$ *to* $\Delta^{\mathcal{J}}$ *such that:*

- $d \in A^{\mathcal{I}}$ *implies* $h(d) \in A^{\mathcal{J}}$ *for all* $A \in \mathsf{N_C}$, *and*
- $(d, d') \in r^{\mathcal{I}}$ *implies* $(h(d), h(d')) \in r^{\mathcal{J}}$ *for all* $r \in \mathsf{N_R}$.

If a homomorphism from $\mathcal{I}$ *to* $\mathcal{J}$ *exists we write* $\mathcal{I} \triangleright \mathcal{J}$.

	Name	Syntax	Semantics
	CONCEPT CONSTRUCTORS		
1	Concept names	A	$A^{\mathcal{I}}$
2	Top concept	$\top$	$\Delta^{\mathcal{I}}$
3	Bottom concept	$\bot$	$\emptyset$
4	Negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
5	Conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
6	Disjunction	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
7	Existential restriction	$\exists r.C$	$\{d \in \Delta^{\mathcal{I}} \mid \exists (d, d') \in R^{\mathcal{I}}$ and $d' \in C^{\mathcal{I}})\}$
8	Universal restriction	$\forall r.C$	$\{d \in \Delta^{\mathcal{I}} \mid \forall (d, d') \in R^{\mathcal{I}}$ and $d' \in C^{\mathcal{I}})\}$
9	Qualified number restrictions	$\leqslant n\, r.C$ $\geqslant n\, r.C$	$\{d \mid n \leq \lvert(d, d') \in R^{\mathcal{I}}$ and $d' \in C^{\mathcal{I}}\rvert\}$ $\{d \mid n \geq \lvert(d, d') \in R^{\mathcal{I}}$ and $d' \in C^{\mathcal{I}}\rvert\}$
	ROLE CONSTRUCTORS		
10	Role names	r	$r^{\mathcal{I}}$
11	Inverse	r^{-}	$\{(d', d) \lvert (d, d') \in r^{\mathcal{I}}\}$
	ABOX AXIOMS		
12	Concept assertion	$A(a)$	$a^{\mathcal{I}} \in A^{\mathcal{I}}$
13	Role assertion	$t(a, b)$	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in t^{\mathcal{I}}$
	TBOX AXIOMS		
14	Concept inclusion axiom	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
15	Role inclusion axiom	$r \sqsubseteq s$	$r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$
16	Transitivity axiom	$trans(r)$	$r^{\mathcal{I}} \circ r^{\mathcal{I}} \subseteq r^{\mathcal{I}}$

Table 2.1: Syntax and semantics of some of the concept and role constructors in DLs, TBox, and ABox axioms relevant for this thesis. C, D represent concepts and r, s roles, whereas A is a concept name and t is a role name.

2.1.2 $\mathcal{ALCH}$ and $\mathcal{ALCHI}$ Description Logics

In this subsection, we present the syntax and semantics of two DLs used extensively throughout this thesis $\mathcal{ALCH}$ and $\mathcal{ALCHI}$. Both of these logics are extensions of $\mathcal{ALC}$, which is one of the most studied DLs, as it is the most basic logic to be propositionally complete and also to contain the existential and universal quantifier constructors deemed interesting for practical purposes. Referring back to the table 2.1 $\mathcal{ALC}$ concepts are defined inductively over the concept constructors listed (1–8), roles are simply role names (10), where as $\mathcal{ALC}$ TBoxes contain only general concept inclusion axioms (14) and ABox assertions (12–13). As already pointed out, role inclusions are represented with the letter $\mathcal{H}$ in the DL naming scheme, from where we get that $\mathcal{ALCH}$ extends $\mathcal{ALC}$ with role inclusions axioms in the TBox (15). In this thesis, we assume that $\mathcal{ALCH}$ TBoxes are always given in the following normal form:

Definition 9 ($\mathcal{ALCH}$ normal form)**.**

$$\bigsqcap A_i \sqsubseteq B \quad \text{(NF1)}$$
$$A \sqsubseteq \bigsqcup B_i \quad \text{(NF2)}$$
$$A \sqsubseteq \exists r.B \quad \text{(NF3)}$$
$$\exists r.A \sqsubseteq B \quad \text{(NF3')}$$
$$A \sqsubseteq \forall r.B \quad \text{(NF4)}$$
$$r \sqsubseteq s \quad \text{(NF5)}$$

where A, A_i, B, B_i are concept names and r, s role names.

It is well known that any $\mathcal{ALCH}$ TBox $\mathcal{T}$ can be normalized into a TBox $\mathcal{T}'$ in normal form presented in Definition 9 in polynomial time so that $\mathcal{T}$ and $\mathcal{T}'$ have the same models up to the original signature of $\mathcal{T}$ (see, e.g., [SKH11]). Note that [SKH11] uses a slightly different normal form than the one shown in the definition above. More precisely, [SKH11] instead of using axioms of the form NF1 and NF2, it uses a more generic form that can express both:

$$\bigsqcap A_i \sqsubseteq \bigsqcup B_i$$

which can be easily transformed into our given normal forms, by introducing a fresh concept name F for the right hand side:

$$\bigsqcap A_i \sqsubseteq F$$
$$F \sqsubseteq \bigsqcup B_i$$

$\mathcal{ALCHI}$ is obtained from $\mathcal{ALCH}$ by adding the inverse role constructor (11) listed in Table 2.1. For the rest of this thesis we assume each $\mathcal{ALCHI}$ TBox to be given in the following normal form:

Definition 10 ($\mathcal{ALCHI}$ normal form)**.**

$$\bigsqcap A_i \sqsubseteq B \quad \text{(NF1)}$$
$$A \sqsubseteq \bigsqcup B_i \quad \text{(NF2)}$$
$$A \sqsubseteq \exists r.B \quad \text{(NF3)}$$
$$A \sqsubseteq \forall r.B \quad \text{(NF4)}$$
$$r \sqsubseteq s \quad \text{(NF5)}$$

where A, A_i, B, B_i are concept names and r, s are roles.

A normalization procedure for an arbitrary $\mathcal{ALCHI}$ can be obtained from the $\mathcal{SHI}$ normalization procedure presented in [Sim13] by simply dropping transitivity axioms in the original ontology. Moreover, since $\mathcal{ALCHI}$ is basically an extension of $\mathcal{ALCH}$ with the role inverse constructor, we could use the normal form of $\mathcal{ALCH}$ but allowing for possibly inverse roles in the place of role names. We chose instead to get rid of the axioms of the form (NF3') in Definition 9 by making use of the following equivalence:

$$\exists r.A \sqsubseteq B \quad \Leftrightarrow \quad A \sqsubseteq \forall r^{-}.B$$

from where we get the normal form for $\mathcal{ALCHI}$ presented above.

2.1.3 Horn-$\mathcal{SHIQ}$ Description Logic

Horn-$\mathcal{SHIQ}$ is an expressive Horn DL that enjoys the nice property of being tractable in data complexity. One obtains Horn-$\mathcal{SHIQ}$ from the very expressive DL $\mathcal{SHIQ}$ by restricting the use of axioms such that they do not exhibit non-deterministic behaviour, i.e. they can not be used to encode the *disjunctive* concept constructor $\sqcup$ in the right hand side of GCIs. Hence, Horn-$\mathcal{SHIQ}$ does not use the disjuntcive concept constructor used in $\mathcal{ALC}$, however it add two new things not present in $\mathcal{ALCHI}$ the *qualified number restrictions* concept constructor and *transitivity axioms*. To properly define Horn-$\mathcal{SHIQ}$ we need to first define $\mathcal{SHIQ}$. We do so by following definition given by [Kaz09].

$\mathcal{SHIQ}$ concepts and roles are defined inductively over the constructors shown in table 2.1, whereas a $\mathcal{SHIQ}$ KB is a set of axioms defined in the table.

If $trans(r) \in \mathcal{T}$ or $trans(r^{-}) \in \mathcal{T}$ we call r transitive, and we consider a role $r \in \mathcal{T}$ *simple* if there are no transitive sub roles of r, i.e. there exists no transitive role $s \in \mathcal{T}$ s.t. $s \sqsubseteq^{*}_{\mathcal{T}} r$. Each role occurring in concepts of the form $\leqslant n\, s.C$ and $\geqslant n\, s.C$ is a simple role.

To define Horn-$\mathcal{SHIQ}$ we first need to define the polarities of $\mathcal{SHIQ}$ concepts occurrences in $\mathcal{SHIQ}$ concepts and axioms as done in [Kaz09]:

- C occurs positively in C, and
- C occurs positively (negatively) in $\neg C_{-}$ ($\neg C_{+}$), and
- C occurs positively (negatively) in $C_{+} \sqcap D$ ($C_{-} \sqcap D$), and
- C occurs positively (negatively) in $C_{+} \sqcup D$ ($C_{-} \sqcup D$), and
- C occurs positively (negatively) in $\exists r.C_{+}$ ($\exists r.C_{-}$), and
- C occurs positively (negatively) in $\forall r.C_{+}$ ($\forall r.C_{-}$), and
- C occurs positively (negatively) in $\geqslant n\, s.C_{+}$ ($\geqslant n\, s.C_{-}$), and
- C occurs positively (negatively) in $\leqslant n\, s.C_{-}$ ($\leqslant n\, s.C_{+}$), and
- C occurs positively (negatively) in $C_{-} \sqsubseteq D$ ($C_{+} \sqsubseteq D$), and
- C occurs positively (negatively) in $D \sqsubseteq C_{+}$ ($D \sqsubseteq C_{-}$).

where $C_{+}(C_{-})$ is some concept in which the concept C occurs positively (negatively), and D is some arbitrary concept.

We say then that some concept C occurs positively (negatively) in $\mathcal{T}$ if C occurs positively (negatively) in some axiom in $\mathcal{T}$. Note that a concept can occur at the same time positively and negatively in the same axiom or in $\mathcal{T}$. Then we are ready to define a Horn-$\mathcal{SHIQ}$ TBox.

Definition 11 (Horn-$\mathcal{SHIQ}$ TBox)**.** *A Horn-$\mathcal{SHIQ}$ TBox $\mathcal{T}$ is a $\mathcal{SHIQ}$ TBox where:*

- *no concept of the form $C \sqcup D$ or $\leqslant n\, r.C$ with $n > 1$ occurs positively in $\mathcal{T}$, and*
- *no concept of the form $\neg C$, $\forall r.C$, $\geqslant n\, r.C$ with $n > 1$, or $\leqslant m\, r.C$ occurs negatively in $\mathcal{T}$.*

For our purposes we assume w.l.o.g. that each Horn-$\mathcal{SHIQ}$ $\mathcal{T}$ is given in the normal form defined below.

Definition 12 (Horn-$\mathcal{SHIQ}$ normal form)**.**

$$\bigsqcap A_i \sqsubseteq B \tag{NF1}$$

$$A \sqsubseteq \exists r.B \tag{NF3}$$

$$A \sqsubseteq \forall r.B \tag{NF4}$$

$$A \sqsubseteq \leqslant 1\, s.B \tag{NF6}$$

where A, A_i, B are concept names, r is a role, and s is a simple role.

For a detailed description of the normalization process please refer to [Kaz09] or [KRH07].

2.1.4 Reasoning Tasks

In the DL literature a large number of reasoning tasks have been considered. In this thesis we are interesting in query answering which is considered an ABox reasoning task. The most basic ABox reasoning task is *instance checking* [BCM$^+$03], this is because other important reasoning tasks such as *knowledge base consistency*, *realization* and *instance retrieval* can be framed in terms of the task of *instance checking*.

Instance checking in a KB $\mathcal{K}$ is to decide:

- for a given individual a and a concept C, if $\mathcal{K} \models C(a)$, or
- for a given pair of individuals (a, b) and a role r, if $\mathcal{K} \models r(a, b)$.

In this case, we call a and instance of C, respectively (a, b) an instance of r.

In *knowledge base consistency* the problem is to decide if there exists an interpretation $\mathcal{I}$ that is a model of the knowledge base such that for each concept C in the knowledge

base $C^{\mathcal{I}}$ is non empty, where as in *instance retrieval* the problem is to find for a given concept, all the individuals in the KB that are instances of the given concept.

Similarly, another reasoning task closely related to instance checking is instance query answering where the task is to find for a given concept name or role name all the individuals respectively pairs of individuals in the KB that are instance of such concept or role.

Query Answering From the databases point of view, queries are seen as a mechanism for retrieving information. We call the data retrieved from the evaluation of the given query the answer to such query. There exist multiple query languages in which queries can be expressed, some of the most famous include: FOL queries, CQs which are a restricted version of FOL formulas that corresponds to select-project-join queries of SQL, and Datalog queries characterized by their support for recursion. Adopting these query languages for querying DL knowledge bases from the syntactical point of view is easy, since we only need to restrict the relation symbols in those queries to concept and role names occurring in the vocabulary of the ontology. However when it comes to the semantics of query answering in DLs, the solution is not so obvious since contrary to the setting of databases where semantics of the queries are tied to one interpretation (the database itself), in DLs KBs we have to deal with multiple interpretations.

For a query q in any given language we write $q(\vec{x})$ and refer to the variables in $\vec{x}$ as answer (free) variables of the query. For a tuple of individuals $\vec{a}$ we write $q(\vec{a})$ to denote the result of substituting each answer variables $x_i \in \vec{x}$ with $a_i \in \vec{a}$.

Given an interpretation $\mathcal{I}$ and a query $q(\vec{x})$ the semantics of answers of the query w.r.t to $\mathcal{I}$ is given as follows:

$$ans(q, \mathcal{I}) = \{(a_i^{\mathcal{I}}, \ldots, a_n^{\mathcal{I}}) \mid \mathcal{I} \models q(\vec{a}), (a_i, \ldots, a_n) \in (\mathsf{N_I}(\mathcal{A}))^n\}$$

Then the semantics of query answering in DL w.r.t to their knowledge base is given by the so called *certain answer semantics*. We call a tuple of individuals $\vec{a}$ in $\mathsf{N_I}(\mathcal{A})$ a certain answer of the $q(\vec{x})$ w.r.t. $(\mathcal{T}, \mathcal{A})$ if the ground query $q(\vec{a})$, is entailed in every model of the KB.

Definition 13 (Certain Answers)**.** *Let $(\mathcal{T}, \mathcal{A})$ be an arbitrary DL KB, and $q(\vec{x})$ an arbitrary query defined over the vocabulary of $\mathcal{T}$. Then the certain answers to $q(\vec{x})$ over $(\mathcal{T}, \mathcal{A})$ are defined as follows:*

$$cert(q, \mathcal{K}) = \{(a_1, \ldots, a_n) \mid (a_i^{\mathcal{I}}, \ldots, a_n^{\mathcal{I}}) \in ans(q, \mathcal{I}) \text{ for each } \mathcal{I} \models \mathcal{K}\}$$

Complexity Measures for Query Answering Complexity theory has proven itself as an indispensable tool for measuring the hardness of problems, and we will often refer to the complexity of certain problems throughout the thesis. We give here a very brief overview of some of the concepts from Complexity Theory used through the thesis, and

point the reader to the classical textbook [Pap94] for an in depth coverage of the topic. In complexity theory the problems are typically reformulated as decision problems, where the given problem is posed as a question with a yes or no answer. For example in the case of query answering problem in DL we give as an input a query q, a knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, and a tuple $\vec{a} \in \mathsf{N_I}(\mathcal{A})$, and the question we seek to answer is: "Is $\vec{a}$ found among the answers in $cert(q, \mathcal{K})$?".

In complexity theory the problems are categorised into complexity classes, which intuitively represent the set of problems whose solution can be computed within certain amount of time or space. The notion of the problem hardness, membership or complete for certain complexity class is the standard one, see [Pap94] for details. Below an overview of some of the important complexity classes is given.

$$\begin{aligned}
\mathsf{PTIME} &= \bigcup_{k>0} \mathsf{DTIME}(n^k) \\
\mathsf{NP} &= \bigcup_{k>0} \mathsf{NTIME}(n^k) \\
\mathsf{PSPACE} &= \bigcup_{k>0} \mathsf{DSPACE}(n^k) \\
\mathsf{NPSPACE} &= \bigcup_{k>0} \mathsf{NSPACE}(n^k) \\
\mathsf{EXPTIME} &= \bigcup_{k>0} \mathsf{DTIME}(2^{n^k}) \\
\mathsf{NEXPTIME} &= \bigcup_{k>0} \mathsf{NTIME}(2^{n^k})
\end{aligned}$$

Complexity class $\mathsf{DTIME}(f(n))(\mathsf{NTIME}(f(n)))$ represents the set of decision problems that can be decided by a (non)deterministic Turing machine in time $O(f(n))$, whereas $\mathsf{DSPACE}(f(n))(\mathsf{NSPACE}(f(n)))$ represents the set of decision problems that can be decided by a (non)deterministic Turing machine using space $O(f(n))$.

For query answering we distinguish two important ways of measuring the complexity of the task: (i) ***Combined Complexity*** where complexity is measured with respect to the whole size of the input ($|q| + |\mathcal{T}| + |\mathcal{A}| + |\vec{a}|$), and (ii) ***Data Complexity*** where complexity is measured with respect to the size of the data ($|\mathcal{A}|$), whereas $|q|$, $|\mathcal{T}|$, and $|\vec{a}|$ are assumed as fixed, i.e. they do not contribute to the complexity.

The combined complexity is the classical measure in which we seek to describe the complexity of the overall problem. The data complexity measure was introduced by [Var82] due to the observation that in the setting of databases the queries are typically very small compared to the data, hence data complexity was considered a more revealing measure for query answering algorithms. Although in the setting of OMQ answering, there are cases when the TBox can be quite large, still there are many application use cases where the size of the query and the TBox is negligible compared to the size of the ABox, therefore accounting for data complexity is important.

For decision problems sometimes it is useful to consider their complements, which are the problems obtained by rephrasing the question of the given problem such that the answers to it are flipped. This is generalized to complexity classes, where for a given class C its complement class is denoted co-C and is the set of all complements of a problems in C. Perhaps the most well known co class of interest to us is the coNP class, the complement class of NP, which contains the query answering problem for expressive DLs measured in terms of data complexity. In contrast, it is worth mentioning that the data complexity of query answering problem for *DL-Lite* falls in a quite low complexity class, in AC^0, which is a circuit complexity class and represents the set of problems that can be solved through the use of polynomially sized circuits of constant depth, with unlimited-fanin AND and OR gates. It is worth mentioning that the data complexity of query answering problem in relational databases is in AC^0 as well.

Finally we give a relation of the main complexity classes mentioned here:

$$AC^0 \subset \text{PTime} \subseteq \text{NP} \subseteq \text{PSpace} = \text{NPSpace} \subseteq \text{ExpTime} \subseteq \text{NExpTime} \subseteq 2\text{ExpTime}$$

2.2 Answer Set Programming

Answer set programming (ASP, in short) is a declarative knowledge representation formalism that falls in the area of *logic programming*. ASP programs correspond to disjunctive datalog programs with negation as failure ($\text{Datalog}^{\vee,\neg}$) under stable model semantics. In ASP paradigm one addresses the solving of a problem by first encoding the problem instance I into an ASP program $\mathcal{P}$, then computing the models M of $\mathcal{P}$ by using an ASP solver, and finally extracting a solution or multiple solutions therein from M depending on the setting. A diagram borrowed from [EIK09] that depicts the uniform approach to problem encoding in ASP is shown in Figure 2.1. Note that ASP has a useful feature when it comes to encoding the problem in a program. Usually the encoding of a solution to the problem $\mathcal{P}_S$ is separated from the data describing a concrete instance of the problem $\mathcal{P}_D$, which decouples the logic of the solution from concrete problem instances thus encouraging more modular encodings that allow reusability of encoded solutions over different problem instances. The ease of moving from the problem instance

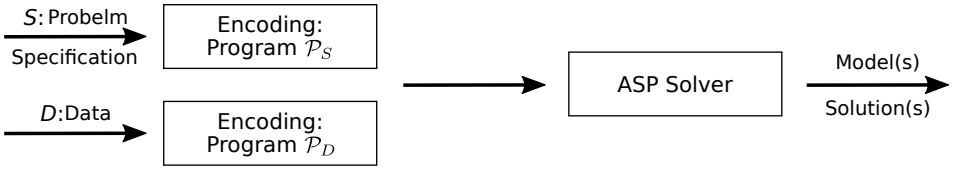


Figure 2.1: Uniform problem encoding in ASP

to modeling and solving has made ASP a successful paradigm which has been applied in a range of areas, among which are constraint satisfaction, robotics, bioinformatics, and security analysis. For a comprehensive list of areas of application, the reader may

consider [EGL16, EIK09], while for a list of latest industrial application of ASP we refer to [FFS+18].

ASP programs are a set of facts, and a set of rules which intuitively justify the derivation of new facts if the conditions they encode are met. In the beginning we will give an intuition of ASP programs and for this reason we present the rules rather informally. Rules in ASP are expressions of the following form:

$$\mathsf{h}_1 \vee \ldots \vee \mathsf{h}_k \leftarrow \mathsf{b}_1, \ldots, \mathsf{b}_\ell, not\ \mathsf{b}_{\ell+1}, \ldots, not\ \mathsf{b}_m$$

where h_i are head atoms, whereas $\mathsf{b}_i s$ are body atoms. One important feature to note in ASP programs is that the negation which appears in the negated atoms in the body of the rules is known as negation as failure, which is different from the standard negation operator, and stands for non provable. A classical example to show it's usefulness is the example from law that a person is innocent unless proven guilty. Consider the following rule:

$$\mathsf{innocent} \leftarrow \mathsf{accused}, not\ \mathsf{evidence}.$$

where the atom accused represents some accused individual, in this case for whom we have no evidence that has committed the crime he/she is being accused of. In classical setting for this example we would not be able to derive that the accused is innocent since we would need for the incriminating evidence to be explicitly state as false, where as in the setting with negation as failure since we can not derive any incriminating evidence we can justify that the accused is innocent. However, tackling negation as failure in a formal way posed a major challenge for the knowledge representation community and many semantics have been proposed. To address this ASP adopts the answer set semantics, also known as the stable model semantics introduced by [GL88].

We will proceed by giving a high level intuition of answer sets (or stable models) upon which answer set semantics is built on. First let's consider ASP programs that do not contain negation, and disjunction in the head. In these the establishing of what is derived is easy as we just need to proceed in bottom-up fashion starting from facts. Let's consider a simple example:

$$\begin{aligned} \mathsf{phd} &\leftarrow . \\ \mathsf{educated} &\leftarrow \mathsf{phd}. \\ \mathsf{smart} &\leftarrow \mathsf{successfull}. \\ \mathsf{successfull} &\leftarrow \mathsf{smart}, \mathsf{educated}. \end{aligned}$$

In this example we can establish phd since it is given as a fact, then from the second rule we establish educated. Since we can not establish neither smart nor successfull from what is given in the program, the only set that is justified is $\{\mathsf{phd}, \mathsf{educated}\}$, which is considered an answer set to the program. Computing justifications in a positive program

without negation is simple indeed, we only need to proceed in iterative fashion, at each iteration deriving new atoms, until we reach a point where no new atoms can be derived. However, in the presence of negation things become more tricky, consider the following simple example:

$$\begin{aligned} \mathsf{innocent} &\leftarrow \mathit{not}\ \mathsf{guilty}. \\ \mathsf{guilty} &\leftarrow \mathit{not}\ \mathsf{innocent}. \end{aligned}$$

Since neither guilty or innocent are given as a fact that means that both rules are reasonable candidates to derive new facts, however if the first rule derives innocent then the precondition of the second will not hold, and vice versa. In this case it is clear that the bottom-up fashion we considered for positive programs does not work. The intuition behind answer set semantics is to assume which atoms will not be derived. For example, if we assume that guilty will not be derived then we can use the first rule to derive innocent, from where we get that the set {innocent} is justified by the program, likewise if we assume that innocent can not be derived, then we can use the second rule of the program to derive the set {guilty}, hence we consider that the program has two answer sets {innocent} and {guilty}. The main idea in answer set semantic is that instead of focusing on computing the answer sets (model) from the program, one starts by proposing a model (everything that can be derived by a program), which gives a way for interpreting the negated atoms and rules with negated atoms, i.e. those that have a negated atom that is found in the proposed model become irrelevant, while the other negated atoms than can be discarded since they all evaluate to true, getting in this way a positive program which is known as the reduct of the program with respect to the proposed model. In the last step one checks if all the atoms in the proposed model are justified by the reduct, if yes then the proposed model is considered an answer set of the program.

2.2.1 Syntax and Semantics

We assume countably infinite set $\mathsf{N_D}$ of *relation symbols*, where each $R \in \mathsf{N_D}$ has an associated arity $\mathsf{arity}(R) \geq 0$. It will be convenient to allow concept names and role names in ASP programs, therefore we assume that $\mathsf{N_R} \cup \mathsf{N_C} \subseteq \mathsf{N_D}$ and let $\mathsf{arity}(A) = 1$ for each $A \in \mathsf{N_C}$, and $\mathsf{arity}(r) = 2$ for each $r \in \mathsf{N_R}$. We use the same countably infinite set $\mathsf{N_I}$ of individual names for *constants* in ASP, and also assume a countably infinite set $\mathsf{N_V}$ of *variables*. The elements of $\mathsf{N_I} \cup \mathsf{N_V}$ are called *terms*.

An *atom* (or, *positive literal*) is an expression of the form $R(\vec{t})$, where R is a relation symbol, and $\vec{t}$ is a tuple of terms with $|\vec{t}| = \mathsf{arity}(R)$. A *negative literal* is an expression of the form *not* $R(\vec{t})$, where $R(\vec{t})$ is an atom with $R \in \mathsf{N_D}$. A *literal* is either a positive literal, or a negative literal. We also refer to $R(\vec{t})$ (resp. *not* $R(\vec{t})$) as positive (negated) atom.

A *rule* ρ is an expression of the form

$$\mathsf{h_1} \vee \ldots \vee \mathsf{h_k} \leftarrow \mathsf{b_1}, \ldots, \mathsf{b_m} \tag{2.1}$$

such that $\{\mathsf{h}_1, \ldots, \mathsf{h}_k\}$ is a set of atoms, denoted $\mathsf{head}(\rho)$, and $\{\mathsf{b}_1, \ldots, \mathsf{b}_m\}$ is a set of literals, denoted $\mathsf{body}(\rho)$. We let $\mathsf{body}^+(\rho)$ denote positive atoms, where as $\mathsf{body}^-(\rho)$ the negative ones appearing in the body of the rule ρ.

A *program* $\mathcal{P}$ is any finite set of rules.

An atom, a literal, a rule, or a program is called *ground* if no variables occur in it. As usual, $\mathsf{dom}(f)$ and $\mathsf{ran}(f)$ denote the *domain* and *range* of a function f, respectively. A *substitution* σ is any partial function from $\mathsf{N_V}$ to $\mathsf{N_I}$. For a rule ρ and a substitution σ, we use $\sigma(\rho)$ to denote the rule that is obtained from ρ by replacing every variable $X \in \mathsf{dom}(\sigma)$ with $\sigma(X)$. The *grounding* of a program $\mathcal{P}$, denoted $\mathsf{ground}(\mathcal{P})$, is the ground program that consists of all ground rules ρ' such that $\rho' = \sigma(\rho)$ for some $\rho \in \mathcal{P}$ and some substitution σ.

Furthermore,

- A rule $\mathsf{p}(\vec{a}) \leftarrow$ consisting of a single ground head is called a *fact*. We write $\mathsf{r}(\vec{t}) \in \mathcal{P}$ in case the rule $\mathsf{r}(\vec{t}) \leftarrow$ is present in a program $\mathcal{P}$.
- A rule $\leftarrow \mathsf{b}_1, \ldots, \mathsf{b}_m$ with no head atoms (i.e., $k = 0$) is called a *constraint*.
- A rule $\mathsf{h} \leftarrow \mathsf{b}_1, \ldots, \mathsf{b}_m$ with only one atom in the head (i.e., $k = 1$) is called *disjunction-free*.
- A rule without negative literals in the body, is considered a *positive disjunctive rule*.

The semantics of ASP programs is given by *Herbrand interpretations*, which are sets of ground atoms. Note that ABoxes are Herbrand interpretations, and a Herbrand interpretation becomes an ABox when restricted to atoms over the predicates in $\mathsf{N_C} \cup \mathsf{N_R}$.

An Herbrand interpretation I is called a *model* of a ground positive program $\mathcal{P}$ if $\mathsf{body}^+(\rho) \subseteq I$ implies $\mathsf{head}(\rho) \cap I \neq \emptyset$ for all $\rho \in \mathcal{P}$. Furthermore, I is a *minimal model* of the ground positive program $\mathcal{P}$ if, I is a model of $\mathcal{P}$ and there is no $J \subsetneq I$ such that J is a model of $\mathcal{P}$.

An *answer set* (a.k.a. *stable model*) of $\mathcal{P}$ is a Herbrand interpretation M that is a minimal model of the *GL-reduct* [GL88] of $\mathsf{ground}(\mathcal{P})$ w.r.t. M, obtained in two steps:

(i) deleting every rule ρ that contains a negative body atom $\mathsf{r}(\vec{a})$ with $\mathsf{r}(\vec{a}) \in M$, and

(ii) deleting all negated atoms in the remaining rules.

Going back to our example:

$$\begin{aligned} \mathsf{innocent} &\leftarrow \mathit{not}\ \mathsf{guilty}. \\ \mathsf{guilty} &\leftarrow \mathit{not}\ \mathsf{innocent}. \end{aligned}$$

Let $M = \{\mathsf{innocent}\}$ be a Herbrand interpretation, then by the rule (i) of the reduct we delete the second rule of the program since we have $\mathsf{innocent} \in M$ and it's negated therefore the body of this rule can never be satisfied, we get:

$$\mathsf{innocent} \leftarrow \mathit{not}\ \mathsf{guilty}.$$

next by the rule (ii) of the reduct we delete the negated atom *not* guilty from the remaining rule. Intuitively, since we know that guilty is not found in M we know that *not* guilty will evaluate to true, hence its redundant. After this step the reduct looks like the following:

$$\mathsf{innocent} \leftarrow$$

Finally, since M is a minimal model of the reduct we conclude that $\{\mathsf{innocent}\}$ is an answer set of the given program.

Note that there is a close correspondence between ASP and variants of the famous language from logic programming Datalog. An ASP program where all rules are positive and disjunction free corresponds to a plain Datalog program where the answer sets of the ASP program coincide with the minimal models of the corresponding Datalog program. Whereas an ASP program is simply a datalog program with negation and disjunction ($\text{Datalog}^{\vee,\neg}$) interpreted under stable model semantics.

2.2.2 Reasoning over Answer Sets

As already showed with our previous example in ASP we commonly get more than one answer set for a given program $\mathcal{P}$. Depending on the task we are considering, the solution might be to pick one of the answer sets and extract a solution from it, or an atom may be part of the solution only if it is found in all the answer sets.

We differentiate two modes of reasoning when it comes to entailment of $r(\vec{c})$ from $\mathcal{P}$:

- Brave reasoning. A ground atom $r(\vec{c})$ is *bravely entailed* from $\mathcal{P}$ if $r(\vec{c}) \in M$ for some answer set M of $\mathcal{P}$.
- Cautious reasoning. A ground atom $r(\vec{c})$ is *cautiously entailed* from $\mathcal{P}$ if $r(\vec{c}) \in M$ for every answer set M of $\mathcal{P}$.

We are usually interested in the second mode of reasoning and cautious entailment is often in close correspondence with certain answer semantics of query answering, therefore for all the programs in this thesis we assume they are evaluated in cautious reasoning mode.

CHAPTER 3

Reasoning About Families of ABoxes

Reasoning in expressive DLs is known to have high complexity. Satisfiability checking of knowledge bases for the basic $\mathcal{ALC}$ DL is already hard for ExpTime in combined complexity [Neb90], whereas data complexity for different query languages ranging from *instance queries* (IQs) to *conjunctive queries* is in coNP [Sch93], making the task of OMQ answering in expressive DLs very challenging. The standard technique for answering OMQs is via rewriting, i.e., given a query and an ontology, the query is rewritten to take into account the knowledge encoded in the ontology such that when evaluating the query over any database its result coincides with the answers of the original query mediated by the ontology. Note that these rewritings do not depend on the data, which is an essential feature that allows the reuse of the rewriting over evolving databases. While data independent query rewriting technique has proven to work well in practice for many lightweight DLs such as *DL-Lite*, and practical algorithms have been implemented, in the case of expressive DLs there are practically no such algorithms implemented. Most of the algorithms for expressive DLs are based on techniques that are costly to implement and mostly target at proving complexity results. However, state-of-the-art reasoners for expressive DLs target standard reasoning task such as satisfiability checking and can handle very large ontologies (e.g., Pellet [SPG+ 07], HermiT [GHM+14], Konclude [SLG14]). But, they implement tableaux algorithms which are based on refutation procedure that is suitable for deciding model existence but regarded as impractical for query answering [HM05]. A rewriting into Datalog for $\mathcal{SHIQ}$ was implemented a decade ago in the KAON2 reasoner, but only for instance queries. A published extension to CQs did not yield a data-independent rewriting and was never implemented [HMS04].

In this chapter, we present a technique that paves the way for a middle ground approach between data dependent and independent query rewriting approaches, namely a structure based approach, in which we make use of structure-specific description of ABoxes of

interest for narrowing the search space for query answering algorithms. We propose *profile sets* as a general and straightforward way to describe families of ABoxes. We claim that, in most real-world cases, data residing in most databases is regular, i.e., one encounters multiple copies of the same patterns, and only a moderate number of profiles is relevant, even when the datasets are large. At the end of this chapter we provide empirical evidence for this.

We provide a type based algorithm that takes as an input a TBox $\mathcal{T}$ in $\mathcal{ALCHI}$, a set of profiles $\mathbb{P}$, and computes a structure $\mathbb{T}$ that represents a set of relevant models, for all knowledge bases of interest, and that can be used for answering OMQs. Specifically, for any ABox $\mathcal{A}$ that complies to the description given by the set of profiles $\mathbb{P}$, from the computed structure $\mathbb{T}$ we can construct a set of models of $(\mathcal{T}, \mathcal{A})$ that is sufficient for answering a range of interesting query languages.

On top of the fact that the computation of the structure $\mathbb{T}$ can be done offline, another important feature of the algorithm for computing the representation of models is the support for *incremental reasoning* on evolving families of ABoxes, i.e. if the representation of models has been obtained for a set of profiles, and a new family of ABoxes becomes of interest, new profiles can be incorporated easily.

The rest of this chapter is organized as follows: In Section 3.1, we lay out the definition of profiles as a means of representing families of ABoxes, from which we obtain base types- 'partially' complete types. Base types are further expanded into 'full' types in Section 3.2 by the type table computation algorithm and stored into a structure $\mathbb{T}$ from which one can construct the set of relevant models for answering queries preserved under homomorphism for any KB covered by the given profiles. In fact, we prove the latter statement in the next chapter. In Section 3.3 explanation of the experimental set up for the rest of the thesis is given, including the benchmarking ontologies used and the implemented prototype Mod4Q for DLs expressed in $\mathcal{ALCH}$. Finally, in Section 3.4 we provide our evaluation results of the tests run against a large repository of ontologies, which show that for a large real-world ABoxes are typically covered by a small number of profiles, and for most large complex ontologies the type computation algorithm is feasible and results into a small-sized structure $\mathbb{T}$.

3.1 Profiles for Representing Families of ABoxes

In practice, data residing in a database adheres to a certain structure, usually encoded via a schema. Hence it makes sense to consider only ABoxes that fit some constraints when rewriting queries mediated by an ontology. We start by presenting an abstract 'schema' for ABoxes and show how one can obtain such schema for specific ABoxes. Later on (in Chapter 7), we show that this representation can be obtained from OBDA specifications, making the algorithms that utilize our 'schema' representation deployable in practice.

To describe families of ABoxes, we define *profiles*, which are combinations of concepts names, role domains, and role ranges that an individual may be asserted to participate

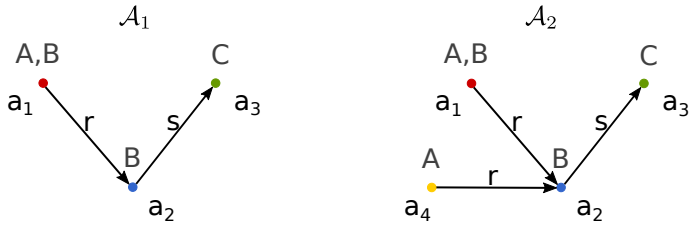
in.

Definition 14 (Profiles)**.** *Concept names in* $\mathsf{N_C}$*, and concepts of the forms* $\exists r$ *and* $\exists r^-$ *with* $r \in \mathsf{N_R}$ *are called* basic concepts*. A* profile *is a set of basic concepts. Given an ABox* $\mathcal{A}$*, the* profile of a in $\mathcal{A}$ *is:*

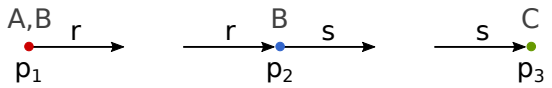
$$\mathsf{prof}^{\mathcal{A}}(a) = \{A \mid A(a) \in \mathcal{A}\} \cup \{\exists r \mid r(a,b) \in \mathcal{A}\} \cup \{\exists r^- \mid r(b,a) \in \mathcal{A}\}$$

A set $\mathbb{P}$ *of profiles* covers $\mathcal{A}$ *if* $\mathsf{prof}^{\mathcal{A}}(a) \in \mathbb{P}$ *for all* $a \in \mathsf{N_I}(\mathcal{A})$*.*

Example 2. *Consider following ABoxes represented as a graphs:*



and the following set of profiles $\mathbb{P} = \{p_1, p_2, p_3\}$*:*



Then $\mathbb{P}$ *covers* $\mathcal{A}_1$ *but doesn't cover* $\mathcal{A}_2$ *as* $\mathsf{prof}^{\mathcal{A}_2}(a_4) = \{A, \exists r\} \notin \mathbb{P}$*.*

Given an ABox $\mathcal{A}$ one can obtain the set of profiles that cover $\mathcal{A}$ by simply taking the set of profiles for each individual occurring in $\mathcal{A}$, i.e, $\{\mathsf{prof}^{\mathcal{A}}(a) | for\ each\ a \in \mathsf{N_I}(\mathcal{A})\}$.

In the rest of this section, we assume a fixed TBox $\mathcal{T}$, and a set $\mathbb{P}$ of profiles that covers all the ABoxes of interest. We then expand the profiles $p \in \mathbb{P}$ with concept names A such that, for some $\mathcal{A}$ covered by $\mathbb{P}$, and some individual a with $\mathsf{prof}^{\mathcal{A}}(a) = p$, it may be the case that $a \in A^{\mathcal{I}}$ holds in the models $\mathcal{I}$ of $(\mathcal{T}, \mathcal{A})$. Roughly, we first expand each profile with possibly different 'guesses' of concepts that their neighbors may propagate to it, and then partially complete it with concepts inferred from $\mathcal{T}$. This gives us *base types* which in the next stage are further expanded to satisfy the axioms in $\mathcal{T}$, or eliminated if we infer that they cannot occur in models.

Definition 15 (Relevant Guesses)**.** *The* relevant guesses for a profile p *are:*

$$Guess^{\mathcal{T}}(p) = \{B \mid \exists r^- \in p, A \sqsubseteq \forall s.B \in \mathcal{T}, r \sqsubseteq^*_{\mathcal{T}} s, B \notin p\}$$

To understand the relevant guesses, we observe that each ABox $\mathcal{A}$, by asserting specific relations between individuals, stipulates concepts that the individuals will need to participate in to satisfy the axioms in $\mathcal{T}$, particularly the ones of the forms (NF4). For example, assume $A \sqsubseteq \forall r^-.B \in \mathcal{T}$. If in a concrete $\mathcal{A}$ we have $r(a,b) \in \mathcal{A}$, then $a \in B^{\mathcal{I}}$ must hold in any model $\mathcal{I}$ of $(\mathcal{T},\mathcal{A})$ with $b \in A^{\mathcal{I}}$. However, this is not enforced if no relation assertions between a and b are present in $\mathcal{A}$, or if b turns out not to be an instance of $\mathcal{A}^{\mathcal{I}}$. To abstract away from the relations asserted in each concrete ABox, and the specific concepts that the neighbors of an object may satisfy, we take a simple approach: we consider all possible combinations (or 'guesses') of sets of concepts that may be enforced at an individual due to its neighborhood in $\mathcal{A}$. That is, we reason explicitly about both $a \in B^{\mathcal{I}}$ and $a \notin B^{\mathcal{I}}$, but only if a participates in a relation r involved in an axiom of the form (NF4).

The subsets of the guesses from the previous definition induce *base types of p*.

Definition 16 (Base Types)**.** *A* type *is a set* $\tau \subseteq \mathsf{N_C} \cup \{\bot, \top\}$*. For a type* τ*, we let*

$$det^{\mathcal{T}}(\tau) = \{B \mid \{A_1, \ldots, A_n\} \subseteq \tau \text{ and } A_1 \sqcap \ldots \sqcap A_n \sqsubseteq B \in \mathcal{T}\}.$$

For a profile p*, we define its* deterministic closure $detCl^{\mathcal{T}}(p)$ *as the smallest type* τ *such that* $(p \cap \mathsf{N_C}) \subseteq \tau$*, and*

(d1) $det^{\mathcal{T}}(\tau) \subseteq \tau$*,*

(d2) if $\exists r^- \in p$ *and* $\top \sqsubseteq \forall s.B \in \mathcal{T}$ *with* $r \sqsubseteq^*_{\mathcal{T}} s$*, then* $B \in \tau$*.*

For $S \subseteq Guess^{\mathcal{T}}(p)$*, we let* $\mathsf{btyp}(p,S) = detCl^{\mathcal{T}}(p \cup S)$*.*

We define base types *induced by profiles and sets of profiles.*

$$\mathsf{btyp}^{\mathcal{T}}(p) = \{\mathsf{btyp}(p,S) \mid S \subseteq Guess^{\mathcal{T}}(p), \bot \notin \mathsf{btyp}(p,S)\}$$
$$\mathsf{btyp}^{\mathcal{T}}(\mathbb{P}) = \bigcup_{p \in \mathbb{P}} \mathsf{btyp}^{\mathcal{T}}(p).$$

As we will show in our experiments in Section 3.4, despite being quite naive, this approach already leads to sets of base types of manageable size for many large ontologies.

Example 3. *Consider the profile set from the Example 2 and the following TBox* $\mathcal{T}$*:*

$$A \sqsubseteq \forall s^-.C \qquad C_2 \sqsubseteq \exists r.C$$
$$C \sqsubseteq C_1 \sqcup C_2 \qquad C_2 \sqsubseteq \exists s.D$$
$$C \sqsubseteq \forall r.B \qquad C_2 \sqcap B \sqsubseteq \bot$$
$$C_1 \sqsubseteq \forall s.A \qquad r \sqsubseteq s$$

Recall that each of the profiles have the following concept names:

$$\begin{aligned} p_1 \cap \mathsf{N_C} &= \{A, B\} \\ p_2 \cap \mathsf{N_C} &= \{B\} \\ p_3 \cap \mathsf{N_C} &= \{C\} \end{aligned}$$

From Definition 15 we get the following set of guesses:

$$\begin{aligned} \mathit{Guess}^{\mathcal{T}}(p_1) &= \{C\} \\ \mathit{Guess}^{\mathcal{T}}(p_2) &= \{A, B, C\} \\ \mathit{Guess}^{\mathcal{T}}(p_3) &= \{A\} \end{aligned}$$

For p_1 we have only one concept to be guessed: C, to accommodate for the possibility of (not) having an s successor that is A in which case the axiom $A \sqsubseteq \forall s^-.C$ would force the concept C, hence the need to reason about both possibilities. Similarly, for p_2 we have to guess A from the possibility of (not) having an s predecessor that is in C_1, and we have to guess C as in the case of p_1. Note that although B is included in the set of guesses for p_2, it is already contained in the set of concept names ($p_2 \cap \mathsf{N_C}$) therefore it doesn't constitute a 'true' guess. Lastly, for the profile p_3 we have to guess A as in the case of p_2, i.e. from the foreseen possibility of (not) having an s predecessor that is in C_1.

Accounting only for true guesses, i.e., discarding the guesses for concepts already contained in the profiles, based on Definition 16 we get the following base types for the given profiles:

$$\begin{aligned} \mathsf{btyp}(p_1, \{\}) &= \{A, B\} \\ \mathsf{btyp}(p_1, \{C\}) &= \{A, B, C\} \end{aligned}$$

$$\begin{aligned} \mathsf{btyp}(p_2, \{\}) &= \{B\} \\ \mathsf{btyp}(p_2, \{A\}) &= \{A, B\} \\ \mathsf{btyp}(p_2, \{C\}) &= \{B, C\} \\ \mathsf{btyp}(p_2, \{A, C\}) &= \{A, B, C\} \end{aligned}$$

$$\begin{aligned} \mathsf{btyp}(p_1, \{\}) &= \{C\} \\ \mathsf{btyp}(p_1, \{A\}) &= \{A, C\} \end{aligned}$$

Note from above that the set S can also be empty. Also, each base type is closed deterministically, that is, it contains its deterministic closure as defined in Definition 16. From the above we notice that we get six resulting base types in total for the given set of profiles and the TBox $\mathcal{T}$: $\tau_1 = \{A, B\}$, $\tau_2 = \{A, B, C\}$, $\tau_3 = \{B\}$, $\tau_4 = \{B, C\}$, $\tau_5 = \{C\}$, and $\tau_6 = \{A, C\}$.

Note that it is often the case that profiles share a considerable number of base types in between each other, which contributes to a relatively small number of types that represent families of ABoxes obtained via profiles.

3.2 Compiling Models in Expressive DLs

In this section, we present an algorithm for compiling a representation of the relevant models for answering different kinds of queries for a given $\mathcal{ALCHI}$ DL and a set of profiles. Roughly, for representing the relevant models we use the notion of types that represent configuration of concept names realizable in relevant models, and store relations between them that witness existential axioms in $\mathcal{T}$. We show that our algorithm is sound and complete in the sense that for any model of the knowledge base we can construct a model from our representation that can be homomorphically mapped to it. A significant feature of this approach is that it can be computed offline and reused to answer many different OMQ, as we will show in the next chapter.

3.2.1 Type Table Compilation

We start from the base types of $\mathbb{P}$ and an $\mathcal{ALCHI}$ TBox $\mathcal{T}$, and compute a representation of *all the relevant models* of the KBs whose ABox is covered by $\mathbb{P}$. We represent models by means of what we call *type tables* $\mathbb{T}$.

Definition 17. *For* $\mathbf{T}$ *a set of types, a type table* $\mathbb{T}$ *is a pair* $(\mathbf{L}, \mathbf{S})$ *with* $\mathbf{S} \subseteq \mathbf{T} \times (\overline{\mathsf{N_R}} \times \mathsf{N_C}) \times \mathbf{T}$, $\mathbf{L} \subseteq (\mathbf{T} \times \mathbf{T})$. *We let*

$$\mathbf{L}(\tau) = \{\tau' \mid (\tau, \tau') \in \mathbf{L}\}, \text{ and}$$
$$\mathbf{S}(\tau, r, B) = \{\tau' \mid (\tau, (r, B), \tau') \in \mathbf{S}\}.$$

A type table $\mathbb{T}$ covers a profile p *if* $\mathbf{L}(\tau) \neq \emptyset$ *for all* $\tau \in \mathsf{btyp}^{\mathcal{T}}(p)$, *and it* covers a set $\mathbb{P}$ of profiles *if it covers all* $p \in \mathbb{P}$. *The set* $\mathbb{T}^G$ *of* good types in $\mathbb{T}$ *contains each* τ *such that:*

(i) $\bot \notin \tau$, *and*

(ii) there is some τ_o *with* $\tau \in \mathbf{L}(\tau_0)$ *and* $\bot \notin \tau_0$.

We let, for each $\tau \in \mathbf{T}$, *and each* $r \in \overline{\mathsf{N_R}}$:

$$fwd^{\mathcal{T}}(\tau, r) = \{B \mid A \sqsubseteq \forall s.B \in \mathcal{T}, A \in \tau, \text{ and } r \sqsubseteq^*_{\mathcal{T}} s\}$$
$$bck^{\mathcal{T}}(\tau, r) = \{B \mid A \sqsubseteq \forall s.B \in \mathcal{T}, A \in \tau, \text{ and } r \sqsubseteq^*_{\mathcal{T}} \mathsf{inv}(s)\}$$

Intuitively the link table $\mathbf{L}$ stores the pairs of an *initial type* (the first component) and a respective extended type (the second component). An initial type is one of the two: (i) a base type of a profile in $\mathbb{P}$, or (ii) a fresh type which is added in order to satisfy some existential axiom. These initial types are 'partial' types which are extended into complete types by the rules of the algorithm. Due to non-deterministic nature of the logic for any of the initial types we may get multiple extended types. When doing these updates, we keep track of the original initial type for three reasons:

- Initial types make the reuse of the computation easier, since at any point during the computation when a new 'fresh' type is needed to satisfy some existential axiom, in case such type coincides with some other initial type in $\mathbf{L}$ we simply reuse it. This allows us to get the computed extended types of initial types in $\mathbf{L}$ for free.

- They are utilized for bookkeeping, allowing us to recall from where expanded types came from, thus making it possible to match them with base types of profiles.

- They play an important role in the termination of the algorithm.

On the other hand, in $\mathbf{S}$ we store the 'branches' of models, i.e. for each existential axiom $A \sqsubseteq \exists r.B$ and a type that would fire such an axiom, we store a triple $(\tau, (r, B), \tau')$ where:

- the type τ is the parent type, and

- the pair (r, B) witnesses the application of such axiom over the parent type, and

- the type τ' is the successor type.

Similarly to the case of $\mathbf{L}$, due to the non-deterministic nature of the logic, we may get in $\mathbf{S}$ multiple successor types for some parent type and a pair (r, B). These act as options when one constructs a model from $\mathbf{S}$ and $\mathbf{L}$.

For computing a type table $\mathbb{T}$ for a given set of profiles $\mathbb{P}$ and an $\mathcal{ALCHI}$ ontology $\mathcal{T}$ we employ the following algorithm:

Algorithm 3.1: Type table computation algorithm for $\mathcal{ALCHI}$

Input: an $\mathcal{ALCHI}$ TBox $\mathcal{T}$, and a set of profiles $\mathbb{P}$
Output: a $\mathcal{T}$-complete type table $\mathbb{T} = (\mathbf{L}_{fin}, \mathbf{S}_{fin})$

(S1) Initialize $\mathbf{L}_0 = \mathsf{btyp}^{\mathcal{T}}(\mathbb{P}) \times \mathsf{btyp}^{\mathcal{T}}(\mathbb{P})$, $\mathbf{S}_0 = \emptyset$.

(S2) We obtain $(\mathbf{L}_{i+1}, \mathbf{S}_{i+1})$ from $(\mathbf{L}_i, \mathbf{S}_i)$ by applying one of the following rules:

(rule-mark)
If there exists τ, r, B with $\mathbf{S}_i(\tau, r, B) \neq \emptyset$ such that $\tau \in \mathbb{T}_i^G$
and $\mathbf{S}_i(\tau, r, B) \cap \mathbb{T}_i^G = \emptyset$
then replace
each $(\tau_0, \tau) \in \mathbf{L}_i$ by $(\tau_0, \tau \cup \{\bot\})$, and
each $(\tau, (r, B), \tau') \in \mathbf{S}_i$ by $(\tau \cup \{\bot\}, (r, B), \tau')$.

(rule-det)
if there exists some $(\tau_0, \tau) \in \mathbf{L}_i$ with $det^{\mathcal{T}}(\tau) \not\subseteq \tau$ and $\tau \in \mathbb{T}_i^G$
then replace
each $(\tau_0, \tau) \in \mathbf{L}_i$ by $(\tau_0, \tau \cup det^{\mathcal{T}}(\tau))$,
each $(\tau, (r, B), \tau') \in \mathbf{S}_i$ by $(\tau \cup det^{\mathcal{T}}(\tau), (r, B), \tau')$,
each $(\tau', (r, B), \tau) \in \mathbf{S}_i$ by $(\tau', (r, B), \tau \cup det^{\mathcal{T}}(\tau))$.

(rule-nondet)
if there exists some $(\tau_0, \tau) \in \mathbf{L}_i$ with $\tau \in \mathbb{T}_i^G$
and an axiom $A \sqsubseteq B_1 \sqcup \ldots \sqcup B_n$ of the form (NF2) in $\mathcal{T}$
such that $A \in \tau$ and $\{B_1, \ldots B_n\} \cap \tau = \emptyset$
then replace, for $1 \leq i \leq n$:
each $(\tau_0, \tau) \in \mathbf{L}_i$ by all $(\tau_0, \tau \cup \{B_i\})$,
each $(\tau, (r, B), \tau') \in \mathbf{S}_i$ by all $(\tau \cup \{B_i\}, (r, B), \tau')$,
each $(\tau', (r, B), \tau) \in \mathbf{S}_i$ by all $(\tau', (r, B), \tau \cup \{B_i\})$.

(rule-addSucc)
if there exists some $(\tau_0, \tau) \in \mathbf{L}_i$ with $\tau \in \mathbb{T}_i^G$
and an axiom $A \sqsubseteq \exists r.B$ of the form (NF3) in $\mathcal{T}$
such that $A \in \tau$ and $\mathbf{S}_i(\tau, r, B) = \emptyset$
then let $\tau_n = \{\{B\} \cup fwd^{\mathcal{T}}(\tau, r)\}$ and:
if $\mathbf{L}_i(\tau_n) = \emptyset$, then add (τ_n, τ_n) to $\mathbf{L}_i$, and
add $(\tau, (r, B), \tau_n)$ to $\mathbf{S}_i$.
if $\mathbf{L}_i(\tau_n) \neq \emptyset$, then add $(\tau, (r, B), \hat{\tau})$ to $\mathbf{S}_i$ for each $\hat{\tau} \in \mathbf{L}_i(\tau_n)$.

(rule-forw)
if there exists some $(\tau, (r, B), \tau') \in \mathbf{S}_i$ with τ and τ' in $\mathbb{T}_i^G$
and $fwd^{\mathcal{T}}(\tau, r) \not\subseteq \tau'$
then we let $\tau_n = \tau' \cup fwd^{\mathcal{T}}(\tau, r)$, and
if $\mathbf{L}_i(\tau_n) = \emptyset$
then add (τ_n, τ_n) to $\mathbf{L}_i$, and
replace $(\tau, (r, B), \tau')$ in $\mathbf{S}_i$ by $(\tau, (r, B), \tau_n)$.
otherwise, replace $(\tau, (r, B), \tau')$ in $\mathbf{S}_i$ by
$(\tau, (r, B), \hat{\tau})$ for each $\hat{\tau} \in \mathbf{L}_i(\tau_n)$.

(rule-back)
if there exists some $(\tau, (r, B), \tau') \in \mathbf{S}_i$ with τ and τ' in $\mathbb{T}_i^G$
such that $bck^{\mathcal{T}}(\tau', r) \not\subseteq \tau$
then replace $(\tau, (r, B), \tau')$ in $\mathbf{S}_i$ by $(\tau_n, (r, B), \tau')$
where $\tau_n = \tau \cup bck^{\mathcal{T}}(\tau', r)$, and
for each τ_0 such that $(\tau_0, \tau) \in \mathbf{L}_i$:
if there is some $(\tau, (r, B), \tau'') \in \mathbf{S}_i$
s.t. $bck^{\mathcal{T}}(\tau', r) \neq bck^{\mathcal{T}}(\tau'', r)$

then add (τ_0, τ_n) to $\mathbf{L}_i$
otherwise, replace (τ_0, τ) in $\mathbf{L}_i$ by (τ_0, τ_n).

(S3) If no rules are applicable, the type table $(\mathbf{L}_i, \mathbf{S}_i)$ is called *$\mathcal{T}$-complete* and the algorithm terminates.

The algorithm starts with $(\mathbf{L}_0, \mathbf{S}_0)$, and generates $(\mathbf{L}_1, \mathbf{S}_1)$, $(\mathbf{L}_2, \mathbf{S}_2)$, ... until a $\mathcal{T}$-complete $(\mathbf{L}_{fin}, \mathbf{S}_{fin})$ is reached. Moreover, the algorithm builds the types in a goal oriented way, closing them deterministically with the rule **(rule-det)**, to satisfy the axioms of the form (NF1), and then using rule **(rule-nondet)** to expand the types into types that minimally satisfy each non-deterministic axiom (NF2), i.e. each type that contains the left-hand side (LHS) of some disjunctive axiom (NF2) is expanded into types where each of them contains exactly one of the disjuncts in right-hand side (RHS). The rule **(rule-addSucc)** adds suitable successor types for each (parent) type that contains the LHS of some existential axiom (NF3). The rule **(rule-forw)** propagates all the concepts that a parent type forces on its successor type due to axioms of the form (NF4), whereas the **(rule-back)** does this in the opposite direction, i.e., it propagates back all the concepts that a successor type forces on its parent type due to axioms of the form (NF4). Note that back-propagation may be non-deterministic, since for a certain branch in the successor structure we can have many options for the successor types and not all of them may propagate back the same concept. In this case rule **(rule-back)** properly expands the parent type. Finally the rule **(rule-mark)** marks bad types, i.e. those that contain $\bot$, and those for which we can not pick a consistent successor.

Although there is no difference in the sequence of rule application from the result point of view, i.e., the computed table would still encapsulate the relevant models, we give precedence to the **(rule-mark)**, **(rule-det)** or any of the applications of the rules **(rule-forw)** and **(rule-back)** when their applications have deterministic effect. Then, we consider **(rule-nondet)** and **(rule-back)** when it's application yields a non-deterministic effect. The last rule to be applied is **(rule-addSucc)**. This order is kept because in many cases speeds up the saturation as the rules with higher precedence are cheaper to apply and help us to identify types that can become bad earlier than later, resulting in this way in inactive types, from which no new entries for successors would be added.

Example 4. *Consider the base types and the ontology from our running example. We show a sequence of type table computation. As per the algorithm for computing $\mathbb{T}$, in the first step (S1) we initialize the* $\mathbf{L}$ *and* $\mathbf{S}$ *tables. According to Example 3 we have six base types, we get:*

$\mathbf{L}_0$	
$\tau_1 = \{A, B\}$	$\tau_1 = \{A, B\}$
$\tau_2 = \{A, B, C\}$	$\tau_2 = \{A, B, C\}$
$\tau_3 = \{B\}$	$\tau_3 = \{B\}$
$\tau_4 = \{B, C\}$	$\tau_4 = \{B, C\}$
$\tau_5 = \{C\}$	$\tau_5 = \{C\}$
$\tau_6 = \{A, C\}$	$\tau_6 = \{A, C\}$

$\mathbf{S}_0$

In the next steps, we apply the rules under (S2) until we reach the state (S3). We will develop the example per rule application. In some cases where we have n axioms applicable under some rule application we chose to divide the rule application in multiple steps for clarity of presentation, while in other cases we apply all at once. Also, in each table $\mathbf{L}_i$ *and* $\mathbf{S}_i$ *we mark with* $\triangleright$ *the component in* $\mathbf{L}_i$ *and (or) the entry in* $\mathbf{S}_i$ *over which the emphasized rule is being applied, while with* ***bold text*** *we mark the resulting change from the rule application at each step.*

(step:1) At this point the only applicable rule is ***(rule-nondet)****, more precisely the axiom* $C \sqsubseteq C_1 \sqcup C_2$ *over the types* τ_2*,* τ_4*,* τ_5 *and* τ_5*, we get:*

$\mathbf{L}_1$	
$\tau_1 = \{A, B\}$	$\tau_1 = \{A, B\}$
$\triangleright\ \tau_2 = \{A, B, C\}$	$\boldsymbol{\tau_{21} = \{A, B, C, C_1\}}$
$\triangleright\ \tau_2 = \{A, B, C\}$	$\boldsymbol{\tau_{22} = \{A, B, C, C_2\}}$
$\tau_3 = \{B\}$	$\tau_3 = \{B\}$
$\triangleright\ \tau_4 = \{B, C\}$	$\boldsymbol{\tau_{41} = \{B, C, C_1\}}$
$\triangleright\ \tau_4 = \{B, C\}$	$\boldsymbol{\tau_{42} = \{B, C, C_2\}}$
$\triangleright\ \tau_5 = \{C\}$	$\boldsymbol{\tau_{51} = \{C, C_1\}}$
$\triangleright\ \tau_5 = \{C\}$	$\boldsymbol{\tau_{52} = \{C, C_2\}}$
$\triangleright\ \tau_6 = \{A, C\}$	$\boldsymbol{\tau_{61} = \{A, C, C_1\}}$
$\triangleright\ \tau_6 = \{A, C\}$	$\boldsymbol{\tau_{62} = \{A, C, C_2\}}$

$\mathbf{S}_1$

(step:2) Next, the rules ***(rule-det)*** *and* ***(rule-addSucc)*** *become applicable. As a rule of thumb we apply* ***(rule-det)*** *first, in this case the axiom* $B \sqcap C_2 \sqsubseteq \bot$ *is applicable over the types* τ_{22} *and* τ_{42}*, from where we get:*

$\mathbf{L}_2$	
$\tau_1 = \{A, B\}$	$\tau_1 = \{A, B\}$
$\tau_2 = \{A, B, C\}$	$\tau_{21} = \{A, B, C, C_1\}$
$\tau_2 = \{A, B, C\}$	$\triangleright\ \boldsymbol{\tau_{22} = \{A, B, C, C_2, \bot\}}$
$\tau_3 = \{B\}$	$\tau_3 = \{B\}$
$\tau_4 = \{B, C\}$	$\tau_{41} = \{B, C, C_1\}$
$\tau_4 = \{B, C\}$	$\triangleright\boldsymbol{\tau_{42} = \{B, C, C_2, \bot\}}$
$\tau_5 = \{C\}$	$\tau_{51} = \{C, C_1\}$
$\tau_5 = \{C\}$	$\tau_{52} = \{C, C_2\}$
$\tau_6 = \{A, C\}$	$\tau_{61} = \{A, C, C_1\}$
$\tau_6 = \{A, C\}$	$\tau_{62} = \{A, C, C_2\}$

$\mathbf{S}_2$

(step:3) The only applicable rule is ***(rule-addSucc)*** *over the types τ_{52} and τ_{62}. More precisely, the axioms $C_2 \sqsubseteq \exists r.C$ and $C_2 \sqsubseteq \exists s.D$. Let's apply $C_2 \sqsubseteq \exists r.C$ first. Note $fwd^{\mathcal{T}}(\tau_5, r) = fwd^{\mathcal{T}}(\tau_6, r) = B$, therefore the (r, C) successor type is $\{B, C\}$. From $\mathbf{L}(\{B, C\}) = \{\tau_{41}, \tau_{42}\}$, however since $\bot \in \tau_{42}$ we have that only $\tau_{41} \in \mathbb{T}^G$, therefore we get an entry for each of the parent types τ_{52}, τ_{62} in the successors table $\mathbf{S}$ with τ_{41} as an r, C successor:*

$\mathbf{L}_3$	
$\tau_1 = \{A, B\}$	$\tau_1 = \{A, B\}$
$\tau_2 = \{A, B, C\}$	$\tau_{21} = \{A, B, C, C_1\}$
$\tau_2 = \{A, B, C\}$	$\tau_{22} = \{A, B, C, C_2, \bot\}$
$\tau_3 = \{B\}$	$\tau_3 = \{B\}$
$\tau_4 = \{B, C\}$	$\tau_{41} = \{B, C, C_1\}$
$\tau_4 = \{B, C\}$	$\tau_{42} = \{B, C, C_2, \bot\}$
$\tau_5 = \{C\}$	$\tau_{51} = \{C, C_1\}$
$\tau_5 = \{C\}$	$\triangleright\ \tau_{52} = \{C, C_2\}$
$\tau_6 = \{A, C\}$	$\tau_{61} = \{A, C, C_1\}$
$\tau_6 = \{A, C\}$	$\triangleright\ \tau_{62} = \{A, C, C_2\}$

$\mathbf{S}_3$

$\boldsymbol{\tau_{52}}$	$\boldsymbol{(r, C)}$	$\boldsymbol{\tau_{41}}$
$\boldsymbol{\tau_{62}}$	$\boldsymbol{(r, C)}$	$\boldsymbol{\tau_{41}}$

(step:4) Finally, we apply ***(rule-addSuc)*** *over the types τ_{52} and τ_{62}, i.e. the axiom $C_2 \sqsubseteq \exists s.D$ and since $fwd^{\mathcal{T}}(\tau_{52}, s) = fwd^{\mathcal{T}}(\tau_{52}, s) = \emptyset$, both types τ_{52} and τ_{62} get the same (s, D) successor type $\{D\}$. And since there are no initial types that match the successor type, the fresh type $\tau_7 = \{D\}$ is added in the type link table $\mathbf{L}$:*

$\mathbf{L}_{fin}$	
$\tau_1 = \{A, B\}$	$\tau_1 = \{A, B\}$
$\tau_2 = \{A, B, C\}$	$\tau_{21} = \{A, B, C, C_1\}$
$\tau_2 = \{A, B, C\}$	$\tau_{22} = \{A, B, C, C_2, \bot\}$
$\tau_3 = \{B\}$	$\tau_3 = \{B\}$
$\tau_4 = \{B, C\}$	$\tau_{41} = \{B, C, C_1\}$
$\tau_4 = \{B, C\}$	$\tau_{42} = \{B, C, C_2, \bot\}$
$\tau_5 = \{C\}$	$\tau_{51} = \{C, C_1\}$
$\tau_5 = \{C\}$	$\triangleright\ \tau_{52} = \{C, C_2\}$
$\tau_6 = \{A, C\}$	$\tau_{61} = \{A, C, C_1\}$
$\tau_6 = \{A, C\}$	$\triangleright\ \tau_{62} = \{A, C, C_2\}$
$\boldsymbol{\tau_7 = \{D\}}$	$\boldsymbol{\tau_7 = \{D\}}$

$\mathbf{S}_{fin}$		
τ_{52}	(r, C)	τ_{41}
τ_{62}	(r, C)	τ_{41}
$\boldsymbol{\tau_{52}}$	$\boldsymbol{(s, D)}$	$\boldsymbol{\tau_7}$
$\boldsymbol{\tau_{62}}$	$\boldsymbol{(s, D)}$	$\boldsymbol{\tau_7}$

Notice that in the example above, in case we would have applied the **(rule-addSucc)** in *(step:2)* instead of **(rule-det)**, we would have gotten two entries in the successor table for each of the types τ_{52} and τ_{62}, more precisely $(\tau_{52}, (r, C), \tau_{41})$, $(\tau_{52}, (r, C), \tau_{42})$, $(\tau_{62}, (r, C), \tau_{41})$ and $(\tau_{62}, (r, C), \tau_{42})$. However this would have not affected the reasoning since the type τ_{42} would have become bad whenever the rule **(rule-det)** was applied, and the algorithm that builds the relevant models based on $\mathbb{T}$ ignores bad types. However, in that sequence of rule applications, the size of the type table $\mathbb{T}_{fin}$ would have grown larger than in our example above. Therefore we apply the precedence of the rule application mentioned earlier.

Example 5. *Consider the ABox $\mathcal{A}_1$, the base types and the computed link table $\mathbf{L}_{fin}$ from our running example. We can visualize the corresponding extended types for each individuals in $\mathcal{A}_1$ in Figure 3.1.*

The rule applications always lead to a $\mathcal{T}$-complete type table, in at most exponential time:

Lemma 1. *The number of different tables $(\mathbf{L}_i, \mathbf{S}_i)$ that can be produced by the rule applications, and the number of rule applications required to reach a $\mathcal{T}$-complete $(\mathbf{L}_{fin}, \mathbf{S}_{fin})$, are bounded by an exponential in the vocabulary of $\mathcal{T}$, and by a polynomial in the number of existential axioms in $\mathcal{T}$.*

Proof. First of all we emphasize that the algorithm is monotonic, i.e. each rule applied by the algorithm may:

- introduce a fresh type (initial type) τ in case $\mathbf{L}(\tau) = \emptyset$, but never remove or alter an initial type in $\mathbf{L}(\tau)$,

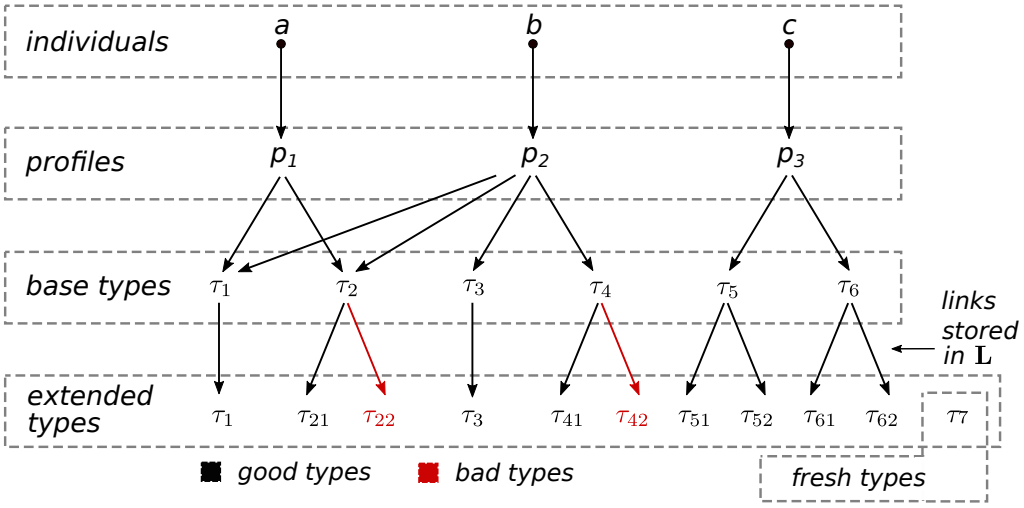


Figure 3.1: Visualisation of the extended types of individuals in $\mathcal{A}_1$.

- add new concepts to extended types but never remove them,
- add new entries in $\mathbf{S}$ for some extended type τ and a pair (r, B) with $\mathbf{S}(\tau, r, B) = \emptyset$, as a result of applying some existential axiom in $\mathcal{T}$.

Since the number of initial types (all types) and the number of existential axioms is finite, and since the algorithm may only add new types that are not found in $\mathbf{L}$, and only add new entries in $\mathbf{S}$ for each (r, B) that witnesses some existential axiom, we get that the number of rule applications is finite and algorithm terminates.
Now the maximal size of table $\mathbf{L}$ is bounded by $2^{\mathsf{N_C}} \times 2^{\mathsf{N_C}}$, and the size of the table $\mathbf{S}$ is bounded by $2^{\mathsf{N_C}} \times |\alpha| \times 2^{\mathsf{N_C}}$, where $|\alpha|$ is the number of existential axioms in $\mathcal{T}$. Since the algorithm is monotonic, the number of different tables that can be produced by the algorithm coincides with their maximal sizes. It follows that there are at most $2^{\mathsf{N_C}} \times 2^{\mathsf{N_C}} + 2^{\mathsf{N_C}} \times |\alpha| \times 2^{\mathsf{N_C}}$ rule applications. □

3.2.2 Incremental Reasoning for Dynamic ABoxes

In practice it is often the case that a database will evolve during time. Naturally most databases will constantly get new facts, and they may evolve in structure in order to incorporate new use cases. Therefore it is important to have a technique that is suitable of dealing with such changes.

In this thesis we see the database as an ABox, but note that this does not cause a loss of generality in our approach, as most of the databases can be transformed into an ABox, moreover as we will show in Chapter 7 we can extract the profiles from a given OBDA specification's mapping layer, which cover any ABox that may be generated when the mappings are evaluated over any possible database.

In our framework we have that if a type table $\mathbb{T}$ covers $\mathbb{P}$, then it also covers every $\mathbb{P}' \subseteq \mathbb{P}$. This follows from the Definition 17, since for each $p \in \mathbb{P}'$ we have that $p \in \mathbb{P}$, and from the assumption that $\mathbb{T}$ covers $\mathbb{P}$ we get that $\mathbb{T}$ covers p, i.e. $\mathbb{T}$ covers $\mathbb{P}'$. The fact that the type table $\mathbb{T}$ contains potentially more types not linked with a set of profiles covering a certain ABox does not affect the reasoning, since only the types linked with the profiles and their respective entries in $\mathbf{S}$ will be utilised during reasoning.

The information stored in the $\mathbf{L}$ table allows us to expand a type table if we need to cover additional profiles, while reusing as much as possible from previous computations. We need only to add the missing base types in $\mathbf{L}$ for a profile that is not covered by $\mathbb{T}$ and initiate the computation (S2) of the Algorithm.

Definition 18 (Incremental Types Computation)**.** *Let $\mathbb{T} = (\mathbf{L}, \mathbf{S})$ be a $\mathcal{T}$-completed type table for a set of profiles $\mathbb{P}$, and let $\mathbb{P}'$ be the set of profiles that are not covered by $\mathbb{T}$. Then we can obtain a $\mathcal{T}$-completed type table $\mathbb{T}'$ that covers $\mathbb{P}'$ by applying exhaustively the rules in (S2) above to:*

$$(\mathbf{L} \cup \{(\tau, \tau) | \tau \in \mathsf{btyp}^{\mathcal{T}}(\mathbb{P}') \ s.t. \ \mathbf{L}(\tau) = \emptyset\}, \mathbf{S})$$

Example 6. *Consider the ABox $\mathcal{A}_1$ from our running example, let's suppose that it's updated with the following assertions:*

$$A(d), r(a, d)$$

yielding a new ABox $\mathcal{A}'$. The profile for d is: $\mathsf{prof}^{\mathcal{A}'}(d) = \{A, \exists r^-\}$ which is not included in the set of profiles $\mathbb{P}$ that cover $\mathcal{A}_1$. Note that although the individual a has a new asserted membership in r, it still has the same profile $\mathsf{prof}^{\mathcal{A}'}(a) = \mathsf{prof}^{\mathcal{A}_1}(a)$. We have the following base types for $\mathsf{prof}^{\mathcal{A}'}(d)$:

$$Guess^{\mathcal{T}}(\mathsf{prof}^{\mathcal{A}'}(d)) = \{A, B\} \qquad \mathsf{btyp}^{\mathcal{T}}(\mathsf{prof}^{\mathcal{A}'}(d)) = \{\{A\}, \{A, B\}\}$$

Note that the candidate type $\{A, B\}$ coincides with τ_1, where as there is no introduced type in $\mathbf{L}$ that coincides with the type $\{A\}$, therefore, the initialization of the $\mathbf{L}$ and $\mathbf{S}$ table for the incremental computation looks like the following:

$\mathbf{L}_0$	
$\tau_1 = \{A, B\}$	$\tau_1 = \{A, B\}$
$\tau_2 = \{A, B, C\}$	$\tau_{21} = \{A, B, C, C_1\}$
$\tau_2 = \{A, B, C\}$	$\tau_{22} = \{A, B, C, C_2, \bot\}$
$\tau_3 = \{B\}$	$\tau_3 = \{B\}$
$\tau_4 = \{B, C\}$	$\tau_{41} = \{B, C, C_1\}$
$\tau_4 = \{B, C\}$	$\tau_{42} = \{B, C, C_2, \bot\}$
$\tau_5 = \{C\}$	$\tau_{51} = \{C, C_1\}$
$\tau_5 = \{C\}$	$\tau_{52} = \{C, C_2\}$
$\tau_6 = \{A, C\}$	$\tau_{61} = \{A, C, C_1\}$
$\tau_6 = \{A, C\}$	$\tau_{62} = \{A, C, C_2\}$
$\tau_7 = \{D\}$	$\tau_7 = \{D\}$
$\tau_8 = \{A\}$	$\tau_8 = \{A\}$

$\mathbf{S}_0$		
τ_{52}	(r, C)	τ_{41}
τ_{62}	(r, C)	τ_{41}
τ_{52}	(s, D)	τ_7
τ_{62}	(s, D)	τ_7

and since, there are no rules in (S2) that are applicable, in this case the initialized tables are already $\mathcal{T}$-complete hence the algorithm terminates.

3.2.3 Type Tables as Model Representations

In the rest of this section, we assume a given ABox $\mathcal{A}$ covered by $\mathbb{P}$, and a $\mathcal{T}$-complete $\mathbb{T} = (\mathbf{L}, \mathbf{S})$ that covers $\mathbb{P}$.

Different models of $(\mathcal{T}, \mathcal{A})$ can be constructed by selecting good types from $\mathbb{T}$. First, the $\mathbf{L}$ relation allows us to assign good types to the input profiles.

Definition 19. *For $p \in \mathbb{P}$, the set of* good types for p in $\mathbb{T}$ *is:*

$$GT_{\mathbb{T}}(p) = \{\tau \in \mathbb{T}^G \mid (\tau_0, \tau) \in \mathbf{L} \text{ for some } \tau_0 \in \mathsf{btyp}^{\mathcal{T}}(p)\}$$

To capture different models of $(\mathcal{T}, \mathcal{A})$, we need to consider the different ways of assigning types from $GT_{\mathbb{T}}(\mathsf{prof}^{\mathcal{A}}(a))$ to the individuals, so that the axioms of the form (NF4) in $\mathcal{T}$ are compatible with the role assertions in $\mathcal{A}$.

Definition 20 ($\mathbb{T}$-assignment)**.** *A $\mathbb{T}$-assignment for $\mathcal{A}$ is a mapping t that assigns a type from $GT_{\mathbb{T}}(\mathsf{prof}^{\mathcal{A}}(a))$ to each $a \in \mathsf{N_I}(\mathcal{A})$ so that:*

(t1) for each $r(a, b) \in \mathcal{A}$, with $A \in t(a)$,
if $A \sqsubseteq \forall s.B \in \mathcal{T}$ for some $r \sqsubseteq^*_{\mathcal{T}} s$, then $B \in t(b)$, and

(t2) for each $r(a, b) \in \mathcal{A}$, with $A \in t(b)$,
if $A \sqsubseteq \forall s.B \in \mathcal{T}$ for some $r \sqsubseteq^*_{\mathcal{T}} s^-$, then $B \in t(a)$.

Note that $GT_{\mathbb{T}}(\mathsf{prof}^{\mathcal{A}}(a)) \neq \emptyset$ for each $a \in \mathsf{N_I}(\mathcal{A})$ is necessary (but not sufficient) for the existence of $\mathbb{T}$-assignments.

Example 7. *In our running example, for $\mathcal{A}_1$ we have:*

$$\begin{aligned} GT_{\mathbb{T}_{fin}}(p_1) &= \{\tau_1, \tau_{21}\} \\ GT_{\mathbb{T}_{fin}}(p_2) &= \{\tau_1, \tau_{21}, \tau_3, \tau_{41}\} \\ GT_{\mathbb{T}_{fin}}(p_3) &= \{\tau_{51}, \tau_{52}, \tau_{61}, \tau_{62}\} \end{aligned}$$

in which case we have 32 ways of assigning types to individuals check Table 3.1, from which only the following eight are $\mathbb{T}_{fin}$-assignments for the ABox $\mathcal{A}_1$:

$t_1(a) = \tau_1$	$t_1(b) = \tau_3$	$t_1(c) = \tau_{51}$
$t_2(a) = \tau_1$	$t_2(b) = \tau_3$	$t_2(c) = \tau_{52}$
$t_3(a) = \tau_1$	$t_3(b) = \tau_{41}$	$t_3(c) = \tau_{61}$
$t_4(a) = \tau_1$	$t_4(b) = \tau_{41}$	$t_4(c) = \tau_{62}$
$t_5(a) = \tau_{21}$	$t_5(b) = \tau_1$	$t_5(c) = \tau_{51}$
$t_6(a) = \tau_{21}$	$t_6(b) = \tau_1$	$t_6(c) = \tau_{52}$
$t_7(a) = \tau_{21}$	$t_7(b) = \tau_{21}$	$t_7(c) = \tau_{61}$
$t_8(a) = \tau_{21}$	$t_8(b) = \tau_{21}$	$t_8(c) = \tau_{62}$

We define a special kind of models of $(\mathcal{T}, \mathcal{A})$ that can be constructed by taking $\mathcal{A}$ and a $\mathbb{T}$-assignment t, and adding successors according to the $\mathbf{S}$ in our type table.

Definition 21. *Let t be a $\mathbb{T}$-assignment. An $(\mathcal{A}, t, \mathbb{T})$-*interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *is defined as follows:*

• *Its domain $\Delta^{\mathcal{I}}$ is a set of sequences of the form $ar_1B_1\tau_1 \dots r_nB_n\tau_n$ with $n \geq 0$, $a \in \mathsf{N_I}(\mathcal{A})$, and*

(d1) For each $0 \leq i < n$, where τ_0 denotes $t(a)$, we have, $(\tau_i, (r_{i+1}, B_{i+1}), \tau_{i+1}) \in \mathbf{S}$ and τ_{i+1} in $\mathbb{T}^G$, and

(d2) For each $a \in \mathsf{N_I}(\mathcal{A})$ ($a \in \Delta^{\mathcal{I}}$) and each pair (r, B) with $\mathbf{S}(t(a), r, B) \neq \emptyset$, there is exactly one $arB\tau \in \Delta^{\mathcal{I}}$, where $\tau \in \mathbb{T}^G$ and

(d3) For each $a \dots \tau_n \in \Delta^{\mathcal{I}}$ and pair (r, B) with $\mathbf{S}(\tau_n, r, B) \neq \emptyset$, there is exactly one $a \dots \tau_n r B \tau'$ in $\Delta^{\mathcal{I}}$.

• *The interpretation function $\cdot^{\mathcal{I}}$ is defined as follows:*

(i1) For each $a \in \mathsf{N_I}(\mathcal{A})$, $a^{\mathcal{I}} = a$, and

(i2) For each $A \in \mathsf{N_C}$,

$$A^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} \mid A \in \mathit{tail}(d)\}$$

where $\mathit{tail}(d) = t(d)$ if $d \in \mathsf{N_I}$, and $\mathit{tail}(d) = \tau_n$ if $d = a \dots \tau_n$, and

$t(a)$	$t(b)$	$t(c)$	Axioms violated
$\{A, B\}$	$\{A, B\}$	$\{C, C1\}$	$A \sqsubseteq \forall s^-.C$
$\{A, B\}$	$\{A, B\}$	$\{C, C2\}$	$A \sqsubseteq \forall s^-.C$
$\{A, B\}$	$\{A, B\}$	$\{A, C, C1\}$	$A \sqsubseteq \forall s^-.C$
$\{A, B\}$	$\{A, B\}$	$\{A, C, C2\}$	$A \sqsubseteq \forall s^-.C$
$\{A, B\}$	$\{A, B, C, C1\}$	$\{C, C1\}$	$A \sqsubseteq \forall s^-.C$
$\{A, B\}$	$\{A, B, C, C1\}$	$\{C, C2\}$	$A \sqsubseteq \forall s^-.C$
$\{A, B\}$	$\{A, B, C, C1\}$	$\{A, C, C1\}$	$A \sqsubseteq \forall s^-.C$
$\{A, B\}$	$\{A, B, C, C1\}$	$\{A, C, C2\}$	$A \sqsubseteq \forall s^-.C$
$\{A, B\}$	$\{B\}$	$\{C, C1\}$	
$\{A, B\}$	$\{B\}$	$\{C, C2\}$	
$\{A, B\}$	$\{B\}$	$\{A, C, C1\}$	$A \sqsubseteq \forall s^-.C$
$\{A, B\}$	$\{B\}$	$\{A, C, C2\}$	$A \sqsubseteq \forall s^-.C$
$\{A, B\}$	$\{B, C, C1\}$	$\{C, C1\}$	$A \sqsubseteq \forall s^-.C$
$\{A, B\}$	$\{B, C, C1\}$	$\{C, C2\}$	$A \sqsubseteq \forall s^-.C$
$\{A, B\}$	$\{B, C, C1\}$	$\{A, C, C1\}$	
$\{A, B\}$	$\{B, C, C1\}$	$\{A, C, C2\}$	
$\{A, B, C, C1\}$	$\{A, B\}$	$\{C, C1\}$	
$\{A, B, C, C1\}$	$\{A, B\}$	$\{C, C2\}$	
$\{A, B, C, C1\}$	$\{A, B\}$	$\{A, C, C1\}$	$A \sqsubseteq \forall s^-.C$
$\{A, B, C, C1\}$	$\{A, B\}$	$\{A, C, C2\}$	$A \sqsubseteq \forall s^-.C$
$\{A, B, C, C1\}$	$\{A, B, C, C1\}$	$\{C, C1\}$	$C_1 \sqsubseteq \forall r.A$
$\{A, B, C, C1\}$	$\{A, B, C, C1\}$	$\{C, C2\}$	$C_1 \sqsubseteq \forall r.A$
$\{A, B, C, C1\}$	$\{A, B, C, C1\}$	$\{A, C, C1\}$	
$\{A, B, C, C1\}$	$\{A, B, C, C1\}$	$\{A, C, C2\}$	
$\{A, B, C, C1\}$	$\{B\}$	$\{C, C1\}$	$C_1 \sqsubseteq \forall r.A$
$\{A, B, C, C1\}$	$\{B\}$	$\{C, C2\}$	$C_1 \sqsubseteq \forall r.A$
$\{A, B, C, C1\}$	$\{B\}$	$\{A, C, C1\}$	$C_1 \sqsubseteq \forall r.A$, $A \sqsubseteq \forall s^-.C$
$\{A, B, C, C1\}$	$\{B\}$	$\{A, C, C2\}$	$C_1 \sqsubseteq \forall r.A$, $A \sqsubseteq \forall s^-.C$
$\{A, B, C, C1\}$	$\{B, C, C1\}$	$\{C, C1\}$	$C_1 \sqsubseteq \forall r.A$
$\{A, B, C, C1\}$	$\{B, C, C1\}$	$\{C, C2\}$	$C_1 \sqsubseteq \forall r.A$
$\{A, B, C, C1\}$	$\{B, C, C1\}$	$\{A, C, C1\}$	$C_1 \sqsubseteq \forall r.A$
$\{A, B, C, C1\}$	$\{B, C, C1\}$	$\{A, C, C2\}$	$C_1 \sqsubseteq \forall r.A$

Table 3.1: All possible type assignment to individuals in $\mathcal{A}_1$ based on the extended types from Example 4.

(i3) For each $r \in \mathsf{N_R}$,

$$\begin{aligned} r^{\mathcal{I}} = \; & \{(a,b) \mid s(a,b) \in \mathcal{A} \text{ with } s \sqsubseteq^*_{\mathcal{T}} r\} \cup \\ & \{(b,a) \mid s(a,b) \in \mathcal{A} \text{ with } s \sqsubseteq^*_{\mathcal{T}} r^-\} \cup \\ & \{(d, dsB\tau) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid s \sqsubseteq^*_{\mathcal{T}} r\} \cup \\ & \{(dsB\tau, d) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid s \sqsubseteq^*_{\mathcal{T}} r^-\} \end{aligned}$$

The set of $(\mathcal{A}, t, \mathbb{T})$*-interpretations is denoted* $mods_t(\mathcal{A}, \mathbb{T})$*, and* $mods(\mathcal{A}, \mathbb{T})$ *denotes the union of* $mods_t(\mathcal{A}, \mathbb{T})$ *for all* t*.*

Next we give a claim that for any good type that contains entries 'branches' in the successor table we can always pick a suitable successor (i.e. successor type that is good).

Claim 1. *Let $\mathbb{T} = (\mathbf{L}, \mathbf{S})$ be a $\mathcal{T}$-complete type table for some $\mathcal{T}$ and $\mathbb{P}$. For each $\tau \in \mathbb{T}^G$ and a pair (r, B) s.t. $\mathbf{S}(\tau, r, B) \neq \emptyset$, there exists a τ_s in $\mathbf{S}(\tau, r, B)$ s.t. $\tau_s \in \mathbb{T}^G$.*

Proof. This claim follows immediately from the rules of the $\mathbb{T}$ computation algorithm. Let $\mathbb{T} = (\mathbf{L}, \mathbf{S})$ be a $\mathcal{T}$-complete type table for some $\mathcal{T}$ and $\mathbb{P}$. Now let's assume that there exists a type $\tau \in \mathbb{T}^G$ and a pair (r, B) s.t. $\mathbf{S}(\tau, r, B) \neq \emptyset$ and for each $\tau_s \in \mathbf{S}(\tau, r, B)$ we have that $\tau_s \notin \mathbb{T}^G$, then by the rule **(rule-mark)** we get that $\bot \in \tau$ which contradicts our assumption that $\tau \in \mathbb{T}^G$ i.e. that $\bot \notin \tau$. □

Claim 2. *Let $\mathbb{T} = (\mathbf{L}, \mathbf{S})$ be a $\mathcal{T}$-complete type table for some $\mathcal{T}$ and $\mathbb{P}$. If a $\mathbb{T}$-assignment function t for $\mathcal{A}$ exists then $mods_t(\mathcal{A}, \mathbb{T}) \neq \emptyset$.*

Proof. By Definition 20 we have that for each $a \in \mathsf{N_I}(\mathcal{A})$ the function t assigns some type $\tau \in \mathbb{T}^G$. Then we get that for each $a \in \mathsf{N_I}(\mathcal{A})$ the assigned type is a good type, from the Claim 1 we have that we can always pick suitable successors for each element with an assigned good type, therefore all the conditions in Definition 21 are satisfied, we have that an $(\mathcal{A}, t, \mathbb{T})$-interpretation is defined. Since we got that an $(\mathcal{A}, t, \mathbb{T})$-interpretation is defined we have that $mods_t(\mathcal{A}, \mathbb{T}) \neq \emptyset$. □

Each interpretation in $mods(\mathcal{A}, \mathbb{T})$ is a model of $(\mathcal{T}, \mathcal{A})$:

Theorem 1. *If $\mathcal{I} \in mods(\mathcal{A}, \mathbb{T})$, then $\mathcal{I} \models (\mathcal{T}, \mathcal{A})$.*

Proof. Let $\mathcal{I} \in mods(\mathcal{A}, \mathbb{T})$. By construction we have that $\mathcal{I}$ is an $(\mathcal{A}, t, \mathbb{T})$-interpretation where t is a $\mathbb{T}$-assignment. Let $\mathbb{T} = (\mathbf{L}, \mathbf{S})$ be a $\mathcal{T}$-complete type table.

First we prove that $\mathcal{I} \models \mathcal{A}$, by proving that each assertion α in $\mathcal{A}$ is modelled by $\mathcal{I}$. Let α be an arbitrary assertion in $\mathcal{A}$, we have two cases:

- α is of the form $A(a)$. By the rule *(i1)* of the construction of $\mathcal{I}$ we have that $a^{\mathcal{I}} = a$. From Definition 20 $\mathbb{T}$-assignment for $\mathcal{A}$ we have that $t(a) \in GT_{\mathbb{T}}(\mathsf{prof}^{\mathcal{A}}(a))$ is assigned to a. Notice that from definition of $GT_{\mathbb{T}}$ (Definition 19) and the definition of base types (Definition 15) the following relation holds:

  $$\mathsf{prof}^{\mathcal{A}}(a) \cap \mathsf{N_C} \subseteq \mathsf{btyp}^{\mathcal{T}}(\mathsf{prof}^{\mathcal{A}}(a)) \subseteq t(a)$$

  Since by the definition of profiles (Definition 14) we have that $A \in \mathsf{prof}^{\mathcal{A}}(a)$, from the relation above we get that $A \in t(a)$ as well. Then, by the *(i2)* rule of the construction of $\mathcal{I}$ in Definition 21 we get that $a \in A^{\mathcal{I}}$ as well.

- α is of the form $r(a,b)$. By the rule *(i1)* of the construction of $\mathcal{I}$ we have that $a^{\mathcal{I}} = a$, $b^{\mathcal{I}} = b$. By the *(i3)* rule of the construction of $\mathcal{I}$ in Definition 21 we get that $(a,b) \in r^{\mathcal{I}}$ as well.

Next we prove that $\mathcal{I} \models \mathcal{T}$, by proving that each axiom in $\mathcal{T}$ is modelled by $\mathcal{I}$. First observe that by construction of $\mathcal{I}$ in Definition 21 we have that for each element $d \in \Delta^{\mathcal{I}}$ the assigned type $tail(d) \in \mathbb{T}^G$ and $tail(d) = \{A \in \mathsf{N_C} \mid d \in A^{\mathcal{I}}\}$.

Let α be an arbitrary axiom in $\mathcal{T}$. By the normal form for $\mathcal{ALCHI}$ we have that α takes one of the following forms:

- (NF1) $\bigsqcap A_i \sqsubseteq B \in \mathcal{T}$. Let d be an arbitrary element in $\Delta^{\mathcal{I}}$, s.t for each A_i we have $d \in A_i^{\mathcal{I}}$. By construction we have that $\bigcup\{A_i\} \subseteq tail(d)$. Lets assume that $B \notin tail(d)$, then the rule **(rule-det)** would be applicable to $tail(d)$, which contradicts our initial assumption that $\mathbb{T}$ is $\mathcal{T}$-complete, i.e. it violates the condition (S3) of the algorithm that all rules have been applied. Therefore $B \in tail(d)$, i.e. $d \in B^{\mathcal{I}}$, α is satisfied.

- (NF2) $A \sqsubseteq B_1 \sqcup \ldots \sqcup B_n \in \mathcal{T}$. Let d be an arbitrary element in $\Delta^{\mathcal{I}}$, s.t. $d \in A^{\mathcal{I}}$. By construction we have that $A \in tail(d)$. Lets assume that $tail(d) \cap \{B_1, \ldots, B_n\} \neq \emptyset$, then the rule **(rule-nondet)** would be applicable to $tail(d)$ replacing it with n other types of the form $tail(d) \cup \{B_i\}$ which contradicts our initial assumption that $\mathbb{T}$ is $\mathcal{T}$-complete. Therefore we get that $tail(d) \cap \{B_1, \ldots, B_n\} \neq \emptyset$, i.e. $d \in B_i^{\mathcal{I}}$ for some $B_i \in \{B_1, \ldots, B_n\}$, α is satisfied.

- (NF3) $A \sqsubseteq \exists r.B \in \mathcal{T}$. Let d be an arbitrary element in $\mathcal{I}$, s.t. $d \in A^{\mathcal{I}}$. By construction we have that $A \in tail(d)$. Lets assume that there exists no pair $(d,d') \in r^{\mathcal{I}}$ s.t. $d' \in B^{\mathcal{I}}$. This means either that (i) $\mathbf{S}(tail(d), r, B) = \emptyset$ or (ii) each of the types in $\mathbf{S}(tail(d), r, B)$ are bad. In the case of (i) we have that the rule **(rule-addSucc)** would be applicable over the type $tail(d)$ for the axiom $A \sqsubseteq \exists r.B$ from where we get that $\mathbf{S}(tail(d), r, B) \neq \emptyset$, which contradicts our assumption that $\mathbb{T}$ is $\mathcal{T}$-complete. Moreover for each $\tau \in \mathbf{S}(tail(d), r, B)$ we have that $B \in \tau$, due to $B \in fwd(tail(d), r)$ and $fwd(tail(d), r) \subseteq \tau$. In the case of (ii) due to rule **(rule-mark)** we would get that $\bot \in tail(d)$ which is a contradiction to our initial assumption that $tail(d) \in \mathbb{T}^G$. This way we have reached a contradiction to our assumption that there exists no pair $(d,d') \in r^{\mathcal{I}}$ s.t. $d' \in B^{\mathcal{I}}$, i.e. the axiom α is satisfied.

- (NF4) $A \sqsubseteq \forall r.B \in \mathcal{T}$. Let $(d,d') \in r^{\mathcal{I}}$ such that $d \in A^{\mathcal{I}}$. By construction of $\mathcal{I}$ we distinguish four possible cases how the pair (d,d') was introduced in $r^{\mathcal{I}}$:

  - There exists $s(d,d') \in \mathcal{A}$ and $s \sqsubseteq^*_{\mathcal{T}} r$, i.e. $d, d' \in \mathsf{N_I}(\mathcal{A})$. Then by construction of $\mathcal{I}$ we have $tail(d) = t(d)$ and $tail(d') = t(d')$ where t is an $\mathbb{T}$-assignment function. Since $A \in t(d)$ and $A \sqsubseteq \forall r.B \in \mathcal{T}$, by the rule *t1* of $\mathbb{T}$-assignment

function (Definition 20), we get that $B \in t(d')$, i.e. $d' \in B^{\mathcal{I}}$, the axiom is satisfied.

- There exists $s(d', d) \in \mathcal{A}$ and $s \sqsubseteq^*_{\mathcal{T}} r^-$, i.e. $d, d' \in \mathsf{N_I}(\mathcal{A})$. Then by construction of $\mathcal{I}$ we have $tail(d) = t(d)$ and $tail(d') = t(d')$ where t is an $\mathbb{T}$-assignment function. Since $A \in t(d)$ and $A \sqsubseteq \forall r.B \in \mathcal{T}$, by the rule *t2* of $\mathbb{T}$-assignment function (Definition 20), we get that $B \in t(d')$, i.e. $d' \in B^{\mathcal{I}}$, the axiom is satisfied.
- There exists a $(\tau, (s, C), \tau') \in \mathbf{S}$ where $\tau = tail(d)$, $\tau' = tail(d')$ and $s \sqsubseteq^*_{\mathcal{T}} r$. Then by the rule *i3* of the construction of $\mathcal{I}$ we have that $(d, d') \in r^{\mathcal{I}}$. By assumption we have that $d \in A^{\mathcal{I}}$, hence we get that $A \in \tau$. Since $B \in fwd^{\mathcal{T}}(\tau, s)$ rule **(rule-fwd)** would be applicable over $(\tau, (s, C), \tau')$ in which case we get that $B \in \tau'$, i.e. $d' \in B^{\mathcal{I}}$ therefore the axiom is satisfied.
- There exists a $(\tau', (s, C), \tau)) \in \mathbf{S}$ where $\tau = tail(d)$, $\tau' = tail(d')$ and $s \sqsubseteq^*_{\mathcal{T}} r^-$. Then by the rule *i3* of the construction of $\mathcal{I}$ we have that $(d, d') \in r^{\mathcal{I}}$. By assumption we have that $d \in A^{\mathcal{I}}$, hence by construction $A \in \tau$. Since $B \in bck^{\mathcal{T}}(\tau, s)$ rule **(rule-back)** would be applicable over $(\tau', (s, C), \tau)$ in which case we get that $B \in \tau'$, i.e. $d' \in B^{\mathcal{I}}$ therefore the axiom is satisfied.

- (NF5) $r \sqsubseteq s \in \mathcal{T}$. Let $(d, d') \in r^{\mathcal{I}}$. From $r \sqsubseteq s$ we have that $r \sqsubseteq^*_{\mathcal{T}} s$, then by the rule *(i3)* of the construction of $\mathcal{I}$ we get that $s(d, d') \in \mathcal{I}$ as well, therefore the axiom is satisfied.

It remains to show that $\bot^{\mathcal{I}} = \emptyset$. By construction we have that for each element $d \in \mathcal{I}$ it's type $tail(d)$ as observed earlier is one of the good types i.e. we have that $\bot \notin tail(d)$. Therefore it follows that for each $d \in \Delta^{\mathcal{I}}$ we have $d \notin \bot^{\mathcal{I}}$. □

Conversely, every model is reflected in $mods(\mathcal{A}, \mathbb{T})$.

Note that given an interpretation $\mathcal{I}$ and an individual $d \in \mathcal{I}$ we use the notation $type^{\mathcal{I}}(d)$ to refer to the type of $d \in \mathcal{I}$, i.e. $type^{\mathcal{I}}(d) = \{A \in \mathsf{N_C} | d \in A^{\mathcal{I}}\}$.

Theorem 2. *If $\mathcal{I} \models (\mathcal{T}, \mathcal{A})$, then there is some $\mathcal{J} \in mods(\mathcal{A}, \mathbb{T})$ such that $\mathcal{J} \triangleright \mathcal{I}$.*

Proof. Let $\mathcal{I}$ be an arbitrary interpretation such that $\mathcal{I} \models (\mathcal{A}, \mathcal{T})$. To this end we use $\mathcal{I}$ to define a new model $\mathcal{J} \in mods(\mathcal{A}, \mathbb{T})$ together with a mapping π from $\Delta^{\mathcal{J}}$ to $\Delta^{\mathcal{I}}$ so that the following hold:

(c1) For every $d \in \Delta^{\mathcal{J}}$, $type^{\mathcal{J}}(d) \subseteq type^{\mathcal{I}}(\pi(d))$, and

(c2) If $(d, d') \in r^{\mathcal{J}}$ for some role name r, then $(\pi(d), \pi(d')) \in r^{\mathcal{I}}$.

We build $\mathcal{J}$ inductively, starting by building $\mathcal{J}_0$ from individuals in $\mathsf{N_I}(\mathcal{A})$, and adding suitable successors. At each step, we define π, and show that (c1)–(c2) still hold. In the

end we show that $\mathcal{J}$ is an $(\mathcal{A}, t, \mathbb{T})$ interpretation, i.e. $\mathcal{J} \in \mathit{mods}(\mathcal{A}, \mathbb{T})$.

Step $\mathcal{J}_0$

- $\Delta^{\mathcal{J}_0} = \mathsf{N_I}(\mathcal{A})$
- $\pi(a) = a^{\mathcal{I}}$
- for each $a \in \mathsf{N_I}(\mathcal{A})$ we pick a type $\tau_a \in \mathbb{T}$ such that the following hold:

  (f1) $(\mathsf{btyp}(\mathsf{prof}^{\mathcal{A}}(a), S), \tau_a) \in \mathbf{L}$ where:

  $$S = \{B \mid r(a,b) \in \mathcal{A}, A \sqsubseteq \forall s.B \in \mathcal{T}, r \sqsubseteq^*_{\mathcal{T}} s^-, B \in \mathit{type}^{\mathcal{I}}(b)\} \cup \{B \mid r(b,a) \in \mathcal{A}, A \sqsubseteq \forall s.B \in \mathcal{T}, r \sqsubseteq^*_{\mathcal{T}} s, B \in \mathit{type}^{\mathcal{I}}(b)\}$$

  (f2) $\tau_a \subseteq \mathit{type}^{\mathcal{I}}(a)$, and
- for each $A \in \mathsf{N_C}$, $A^{\mathcal{J}_0} = \{a | A \in \tau_a \text{ and } a \in \mathsf{N_I}\}$.
- for each $r \in \mathsf{N_R}$, $r^{\mathcal{J}_0} = \{(a,b) \mid s(\pi(a), \pi(b)) \in \mathcal{I} \text{ and } s(a,b) \in \mathcal{A} \text{ and } s \sqsubseteq^*_{\mathcal{T}} r\}$

We argue that for each $a \in \mathsf{N_I}(\mathcal{A})$ such τ_a exists. To prove this let's pick an arbitrary individual $a \in \mathsf{N_I}(\mathcal{A})$. First we show that we can pick a type in $\mathbb{T}$ such that (f1) holds.

Note that the set S from (f1) corresponds to one of the subset's of the guess set $\mathit{Guess}^{\mathcal{T}}(\mathsf{prof}^{\mathcal{A}}(a))$. Also, $S \subseteq \mathit{type}^{\mathcal{I}}(a)$, otherwise one of the axioms of the form (NF4) that enforce the concepts that an individual gets from it's neighbourhood in $\mathcal{I}$ would be violated, from which would follow that $\mathcal{I} \not\models (\mathcal{T}, \mathcal{A})$. From definition of base types we have:

$$\mathsf{btyp}^{\mathcal{T}}(p, S) = \mathit{detCl}^{\mathcal{T}}(p \cup S)$$

Since we already established that $\mathit{type}^{\mathcal{I}}(a) \supseteq S$, and we know that $\{A | A(a) \in \mathcal{A}\} \subseteq \mathit{type}^{\mathcal{I}}(a)$ we have that $\mathit{detCl}^{\mathcal{T}}(\mathsf{prof}^{\mathcal{A}}(a) \cup S)$ is included as well, therefore we get the following:

$$\mathsf{btyp}(\mathsf{prof}^{\mathcal{A}}(a), S) \subseteq \mathit{type}^{\mathcal{I}}(a)$$

Note that $\bot \notin \mathsf{btyp}(p, S)$, otherwise we would have $a \in \bot^{\mathcal{I}}$. By (S1) of type table computation $\mathbb{T}$ we have that $(\mathsf{btyp}(\mathsf{prof}^{\mathcal{A}}(a), S), \mathsf{btyp}(\mathsf{prof}^{\mathcal{A}}(a), S))$ would be added during the initialization of $\mathbf{L}_0$. Note that the type table computation is monotonic, each rule application may only add new concepts to second component of the $\mathbf{L}$ table. Hence, for each $\tau \in \mathbf{L}(\mathsf{btyp}(\mathsf{prof}^{\mathcal{A}}(a), S))$ we have $\mathsf{btyp}(\mathsf{prof}^{\mathcal{A}}(a), S) \subseteq \tau$, i.e. fulfills (f1). Next, it

remains to show that we can pick a type $\tau_a \in \mathbf{L}_{fin}(\mathsf{btyp}(\mathsf{prof}^{\mathcal{A}}(a), S))$ such that (f2) is fulfilled.

To this end, we let $w_n^{\tau} = \tau_0 \dots \tau_{n-1}\tau$ denote the sequence of computed types from the initialisation τ_0 up to the completed type τ, i.e $(\tau_0, \tau) \in \mathbf{L}$. Furthermore each w_{i+1}^{τ} is obtained from w_i^{τ} by applying one of the rules of the type table computation, which results in τ_i alteration. Let $w_n^{\tau_a} = \mathsf{btyp}(\mathsf{prof}^{\mathcal{A}}(a), S) \dots \tau_a$. For $w_0^{\tau_a} = \mathsf{btyp}(\mathsf{prof}^{\mathcal{A}}(a), S)$ we already showed that $\mathsf{btyp}(\mathsf{prof}^{\mathcal{A}}(a), S) \subseteq type^{\mathcal{I}}(a)$. Now assuming that for $w_i^{\tau_a}$ we have $\tau_i \subseteq type^{\mathcal{I}}(a)$ we prove that $\tau_{i+1} \subseteq type^{\mathcal{I}}(a)$ as well. Let $\tau_{i+1} = \tau_i \cup der$, then der is the set of concepts added to τ_{i+1} by applying one of the following rules:

- **(rule-det)** $der = \{B\}$ from application of some axiom $A_1 \sqcap \dots \sqcap A_n \sqsubseteq B \in \mathcal{T}$ to τ_i. By assumption we have that $\{A_1, \dots, A_n\} \subseteq type^{\mathcal{I}}(a)$, and since $\mathcal{I} \models \mathcal{T}$ we get that $B \in type^{\mathcal{I}}(a)$ as well.

- **(rule-nondet)** $der = \{B_j\}$ from application of some axiom $A \sqsubseteq B_1 \sqcup \dots \sqcup B_n \in \mathcal{T}$ to τ_i. Note that we get n different types $\tau_{i+1}^j = \tau_i \cup \{B_j\}$ (where $B_j \in \{B_1, \dots, B_n\}$). By assumption we have that $\{A\} \in type^{\mathcal{I}}(a)$, and since $\mathcal{I} \models \mathcal{T}$ we get that at least some $B_j \in \{B_1, \dots, B_n\}$ is included in $type^{\mathcal{I}}(a)$, otherwise $\mathcal{I} \not\models \mathcal{T}$. Therefore at least for one of the τ_{i+1}^j types we have that $\tau_{i+1}^j \subseteq type^{\mathcal{I}}(a)$.

- **(rule-forw)** by definition this rule does not change existing types, it can only introduce new ones, hence τ_i would not be affected by this rule application.

- **(rule-back)** $der = bck^{\mathcal{T}}(s, \tau')$ s.t. there exists an entry $(\tau_i, (s, B), \tau') \in \mathbf{S}$. Observe that, each entry $(\tau_i, (s, B), \tau') \in \mathbf{S}$ is a result of applying the rule **(rule-addSucc)** over τ_i for some axiom $A \sqsubseteq \exists s.B \in \mathcal{T}$ s.t. $A \in \tau_i$. But then, by assumption we have that $a \in A^{\mathcal{I}}$ as well, and since $\mathcal{I} \models \mathcal{T}$ there exists an individual $b \in \Delta^{\mathcal{I}}$ such that $(a, b) \in r^{\mathcal{I}}$ and $s \sqsubseteq_{\mathcal{T}}^* r$. We know that $fwd^{\mathcal{T}}(\tau_i, s) \subseteq type^{\mathcal{I}}(b)$ otherwise $\mathcal{I}$ does not model some axiom of the form (NF4) in $\mathcal{T}$ that is applicable to $type^{\mathcal{I}}(a)$ or the existential axiom $A \sqsubseteq \exists s.B \in \mathcal{T}$. Now since $fwd^{\mathcal{T}}(\tau_i, s) \subseteq type^{\mathcal{I}}(b)$ it follows that $der = bck^{\mathcal{T}}(\tau', r) \subseteq type^{\mathcal{I}}(a)$ as well.

- **(rule-mark)** Note that propagation of $\bot$ to parent types from their immediate successor types of some 'branch' r, B is done in the case when all of the successor types contain $\bot$. Note that in case all the types in $\mathbb{T}$ are good, then the rule **(rule-mark)** would never be triggered, and the only way for such rule to ever become applicable, $\bot$ has to be added to all of the successor types of some parent type for some branch r, B through **(rule-det)** applications over each of the successor types. Therefore, as long as we can find successor types from $\mathbf{S}$ that maps to elements in $\Delta^{\mathcal{I}}$ we can guarantee that rule **(rule-mark)** wasn't applicable to any of the types used in the process, otherwise at some point we will reach a successor type τ which gets $\bot$ due to some **(rule-det)**, and since the type is mapped to some individual $d \in \Delta^{\mathcal{I}}$ that means that $d \in \bot^{\mathcal{I}}$, i.e. $\mathcal{I} \not\models (\mathcal{T}, \mathcal{A})$.

Since we proved that for each $a \in \mathsf{N_I}(\mathcal{A})$ we can pick a type in $\mathbb{T}$ such that (f1) and (f2) are satisfied, we get that for each $a \in \Delta^{\mathcal{J}_0}$, $type^{\mathcal{J}_0}(a) \subseteq type^{\mathcal{I}}(\pi(a))$, i.e. (c1) is satisfied.

Since $\mathcal{I} \models (\mathcal{T}, \mathcal{A})$, for each $s(a,b) \in \mathcal{A}$ and $s \sqsubseteq^*_{\mathcal{T}} r$ we have $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$. Considering that π maps each $a \in \Delta^{\mathcal{J}_0}$ to $a^{\mathcal{I}}$, from the last step of $\mathcal{J}_0$ construction we get that for each $(a,b) \in r^{\mathcal{J}_0}$, $(\pi(a), \pi(b)) \in r^{\mathcal{I}}$ as well, i.e., (c2) is satisfied as well.

Next we prove the inductive case, let (c1) and (c2) hold for some $\mathcal{J}_k$, we show that they hold also for $\mathcal{J}_{k+1}$. For the *Step* $\mathcal{J}_{k+1}$ we have:

Step $\mathcal{J}_{k+1}$

We add the successors for this step in the following manner:

For each $a \dots \tau_k \in \Delta^{\mathcal{J}_k}$ s.t. $\pi(a \dots \tau_k) = d$, if there exists a $r^{\mathcal{I}}(d,e)$ with $B \in type^{\mathcal{I}}(e)$ s.t. $\mathbf{S}(\tau_k, s, B) \neq \emptyset$, $a \dots \tau_k s B \tau' \notin \Delta^{\mathcal{J}_{k+1}}$, and $s \sqsubseteq^*_{\mathcal{T}} r$, we proceed as follows:

- pick some $\tau_e \in (\tau_k, s, B) \in \mathbf{S}$ such that $\tau' \subseteq type^{\mathcal{I}}(e)$, and
- add $a \dots \tau_k s B \tau_e$ to $\Delta^{\mathcal{J}_{k+1}}$, and
- set $\pi(a \dots \tau_k s B \tau_e) = e^{\mathcal{I}}$, and
- for each $A \in \mathsf{N_C}$, let $A^{\mathcal{J}_{k+1}} = A^{\mathcal{J}_k} \cup \{w | A \in \tau_e \text{ and } a \dots \tau_k s B \tau_e \in \Delta^{\mathcal{J}_{k+1}}\}$, and
- for each $r \in \mathsf{N_R}$, let $r^{\mathcal{J}_{k+1}} = r^{\mathcal{J}_k} \cup \{r(a \dots \tau_k, a \dots \tau_k s B \tau_e) | s \sqsubseteq^*_{\mathcal{T}} r\}$.

We want to prove that (c1) and (c2) hold for $\mathcal{J}_{k+1}$ as well. To this end, let $a \dots \tau_k$ be an arbitrary element in $\mathcal{J}^k$ s.t. $\pi(a \dots \tau_k) = d$, and let (d,e) be an arbitrary instances of $r^{\mathcal{I}}$ s.t. $\mathbf{S}(\tau_k, s, B) \neq \emptyset$, $s \sqsubseteq^*_{\mathcal{T}} r$, $a \dots \tau_k s B \tau' \notin \Delta^{\mathcal{J}_{k+1}}$, and $B \in type^{\mathcal{I}}(e)$. We argue that we can pick a type τ_e from $\mathbf{S}(\tau_k, s, B)$ s.t. $\tau_e \subseteq type^{\mathcal{I}}(e)$. Now, for each of $\tau' \in \mathbf{S}(\tau_k, s, B)$ it is the case that $fwd^{\mathcal{T}}(\tau_k, s) \cup \{B\} \subseteq \tau'$, moreover as per **(rule-addSucc)**, $(\{B\} \cup fwd^{\mathcal{T}}(\tau_k, s), \{B\} \cup fwd^{\mathcal{T}}(\tau_k, s))$ would have been added to $\mathbf{L}$, hence for each τ' we have $\mathbf{L}_{fin}(\{B\} \cup fwd^{\mathcal{T}}(\tau_k, s), \tau')$.

Since, the type computation algorithm is monotonic, we get that for each $\tau' \in \mathbf{S}(\tau_k, (s, B))$ its sequence of computed types looks like the following: $w^{\tau'} = \{B\} \cup fwd^{\mathcal{T}}(\tau_k, s) \dots \tau'$. Similarly as in the base case we show by induction in the sequence of computed types that there exists a $\tau_e \in \mathbf{S}(\tau_k, (s, B))$ s.t. $\tau_e \subseteq type^{\mathcal{I}}(e)$. Note that for the base case, we have that $fwd^{\mathcal{T}}(\tau_k, s) \cup \{B\} \subseteq type^{\mathcal{I}}(e)$ as well, in the converse $\mathcal{I}$ does not model some axiom of the form (NF4) in $\mathcal{T}$ that is applicable to d and e. Now assuming that for $w_i^{\tau_e}$ we have $\tau_i \subseteq type^{\mathcal{I}}(e)$ we prove that $\tau_{i+1} \subseteq type^{\mathcal{I}}(e)$ as well. Let $\tau_{i+1} = \tau_i \cup der$, then der is the set of concepts added to τ_{i+1} by applying one of the rules of the algorithm. The proof for each rule is identical to the one in the base case for $\mathcal{J}_0$. Hence we can set $\pi(a \dots \tau_k s B \tau_e) = e^{\mathcal{I}}$ s.t. $\tau_e \subseteq type^{\mathcal{I}}(e)$, i.e. (c1) holds. From the last step of $\mathcal{J}_{k+1}$ we get

that $(a \dots \tau_k, a \dots \tau_k sB\tau_e) \in r^{\mathcal{J}_{k+1}}$, where as from our assumption that $\pi(a \dots \tau_k) = d^{\mathcal{I}}$, $(d, e) \in r^{\mathcal{I}}$, and since we established $\pi(a \dots \tau_k sB\tau_e) = e^{\mathcal{I}}$ we get that (c2) holds as well.

Since we proved that (c1) and (c2) hold for the inductive case as well we have established that $\mathcal{J} \triangleright \mathcal{I}$. It remains to be proven that $\mathcal{J} \in mods(\mathcal{A}, \mathbb{T})$. To prove this, we show that $\mathcal{J}$ is an $(\mathcal{A}, t, \mathbb{T})$-interpretation.

First, note that each assigned type in $\mathcal{J}$ is good, otherwise as per our observation in the case **(rule-mark)** of the construction of $\mathcal{J}_0$, we know that at some point we would get some type that satisfies some non-deterministic axiom, and since each type is a subset of the type of some individual in $\mathcal{I}$, it would follow that $\mathcal{I}$ is not a model of $\mathcal{T}$ which is a contradiction.

Furthermore, the type assignments for each individual in $\Delta^{\mathcal{J}_0}$ fulfill the conditions $t1$ and $t2$ of $\mathbb{T}$-assignment function due to condition $(f1)$ of the $\mathcal{J}_0$ construction. From where it follows that the type assignments in $\mathcal{J}_0$ correspond to some $\mathbb{T}$-assignment function t, therefore the basic condition for $\mathcal{J}$ being an $(\mathcal{A}, t, \mathbb{T})$-interpretation is fulfilled. Moreover, the construction of $\mathcal{J}_i$ satisfies all conditions $(d1)$-$(d3)$ of the Definition 21 of $(\mathcal{A}, t, \mathbb{T})$. Let's assume the converse that there exists an individual $a \dots \tau \in \mathcal{J}$ and a pair (s, B) s.t. $\mathbf{S}(\tau, s, B) \neq \emptyset$ and there exists no individual $a \dots \tau sB\tau' \in \mathcal{J}$ for some $\tau' \in \mathbf{S}(\tau, s, B)$. From this would follow that there exists some axiom $A \sqsubseteq \exists s.B \in \mathcal{T}$, such that $A \in \pi(a \dots \tau)$ and no pair $(\pi(a \dots \tau), e) \in s^{\mathcal{I}}$, which is a contradiction. Furthermore, from the construction of each $\mathcal{J}_i$ we get that there exists only one $a \dots \tau sB\tau' \in \Delta^{\mathcal{J}}$. Finally, since for each $a \in \mathsf{N_I}(\mathcal{A})$ we have $a^{\mathcal{J}} = a$ the condition $(i1)$ of a $(\mathcal{A}, t, \mathbb{T})$-interpretation is satisfied, where as the last two steps of $\mathcal{J}_i$ construction faithfully treat the concept and role memberships as per conditions $(i2)$ and $(i3)$ of a $(\mathcal{A}, t, \mathbb{T})$-interpretation, therefore we arrive at the conclusion that $\mathcal{J}$ is a $(\mathcal{A}, t, \mathbb{T})$-interpretation, i.e. $\mathcal{J} \in mods(\mathcal{A}, \mathbb{T})$. □

Finally, we remark that, since the algorithm guarantees that good types always have suitable successors in $\mathbf{S}$ to continue the model construction, the existence of a $\mathbb{T}$-assignment already implies the existence of an $(\mathcal{A}, t, \mathbb{T})$-interpretation.

Lemma 2. *$(\mathcal{T}, \mathcal{A})$ is satisfiable iff $\mathcal{A}$ has a $\mathbb{T}$-assignment.*

Proof. First we prove the $(\Rightarrow)$ direction. Let t be a $\mathbb{T}$-assignment. By Claim 2 we have that $mods_t(\mathcal{A}, \mathbb{T}) \neq \emptyset$, i.e. there exists at least one $(\mathcal{A}, t, \mathbb{T})$-interpretation $\mathcal{I}$, then from Theorem 1 we get that $\mathcal{I} \models (\mathcal{T}, \mathcal{A})$, hence $(\mathcal{T}, \mathcal{A})$ is satisfiable.

$(\Leftarrow)$. Let $(\mathcal{T}, \mathcal{A})$ be satisfiable, then there exists an interpretation $\mathcal{I}$ such that $\mathcal{I} \models (\mathcal{T}, \mathcal{A})$. From Theorem 2 we have that there exists a $\mathcal{J} \in mods$ s.t. $\mathcal{J} \triangleright \mathcal{I}$. From Definition 21 we have that each $\mathcal{J} \in mods_t(\mathcal{A}, \mathbb{T})$ is constructed starting from a $\mathbb{T}$-assignment t of types to ABox individuals, therefore there exists a $\mathbb{T}$-assignment t for $\mathcal{A}$. □

In the next section, we explain the benchmarking set up for testing the feasibility of the developed algorithms in this thesis that make use of the model compilation showed in this chapter.

3.3 Benchmarks Set-up for Expressive DLs

Recall that one of the goals of the thesis was that the developed algorithms should have the potential for implementation (G3). Moreover, we required that the data representations over which our algorithms reason should be small compared to the data and easy to be computed (G1).

In this section, we describe the experimental set up for assessing the achievement of the aforementioned goals for expressive DLs, i.e., we present the instances (ontologies), the implemented prototype, and the hardware environment used in the experiments we will present in Chapter 3, 4 and 5. Note that in Chapter 6 we use the same source to obtain the ontologies, but define different criteria for selecting them.

3.3.1 Software and Hardware

We have implemented the type table $\mathbb{T}$ computation algorithm for $\mathcal{ALCH}$ including most of the algorithms presented in this thesis who utilize $\mathbb{T}$ into a proof of concept implementation, which we call Mod4Q[1] (an abbreviation for models for querying). The fact that Mod4Q prototype is implemented for the DL $\mathcal{ALCH}$, rather than for $\mathcal{ALCHI}$; is mostly due to historic reasons and the way this work evolved. The extension to $\mathcal{ALCHI}$ would be relatively simple, it just requires a little more engineering effort.

Our implementation is written in Java and uses a PostgreSQL 9.5.5 as a back-end database for storing and computing $\mathbb{T}$. For managing ontologies it uses OWLAPI [HB11], whereas for evaluating the resulting translations it uses Clingo 4.2.1 [GKK+11], one of the most prominent ASP solvers to date. All experiments presented in this thesis were run on a PC with an i7 2.4 GHz CPU with 4 cores running 64bit Linux-Mint 17, with a JAVA heap space of 12GB.

Mod4Q has two modes of work, the *full* and *incremental mode*. In full mode it clears the profiles and entries in $\mathbb{T}$ from previous computations, and computes everything from scratch, whereas in incremental mode it checks if there are individuals that are not covered by the profiles stored in previous computation, in which case it performs the type table computation algorithm only for the new profiles reusing the entries in $\mathbb{T}$. A workflow diagram of Mod4Q is given in Figure 3.2.

[1] http://www.kr.tuwien.ac.at/research/systems/Mod4Q/

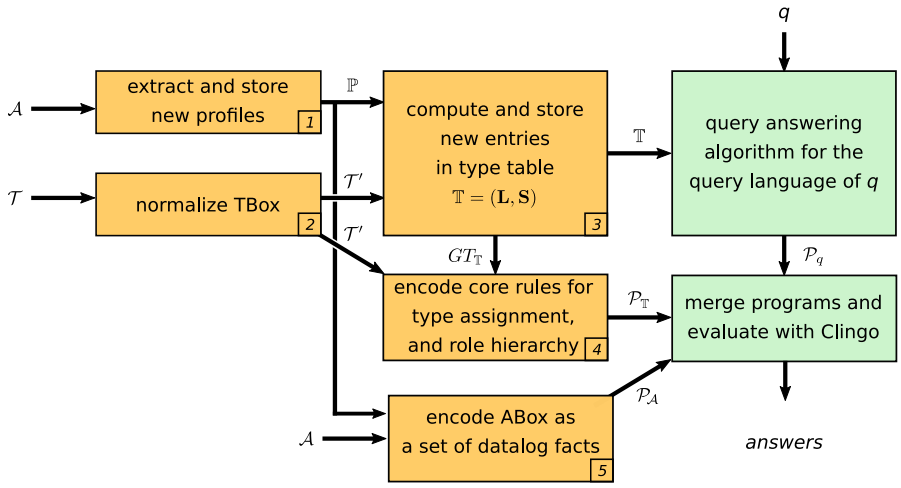


Figure 3.2: Workflow Diagram of Mod4Q

The boxes marked with orange in Figure 3.2 depict the steps adhering to offline computation phase of our approach, where as those marked with green depict the steps adhering to online phase, i.e., the query answering phase. The reasoner works like follows:

- In the offline phase it takes as an input an ABox $\mathcal{A}$ and an $\mathcal{ALCH}$ TBox $\mathcal{T}$, from which, in step 1 it obtains the profiles of individuals in $\mathsf{N_I}(\mathcal{A})$, whereas in step 2 it normalizes $\mathcal{T}$ into $\mathcal{T}'$ by dropping axioms not in $\mathcal{ALCH}$ and transforming others into axioms that conform to the normal form shown in Definition 9. Moreover, in step 1, in case the working mode is *incremental*, it first matches the obtained profiles against the stored ones ($\mathbb{P}_s$) in the database and passes to the next step only the new ones ($\mathbb{P}$). In the case when it is working in *full mode* it simply passes the obtained profiles. In step 3 it extracts the base types from the passed profiles $\mathbb{P}$ and runs the type table computation algorithm described in Section 3.2.1 to obtain a $\mathcal{T}$-complete type table $\mathbb{T}$ that covers $\mathbb{P} \cup \mathbb{P}_s$. During the whole time the reasoner maintains designated indexes that point from individuals to profiles and from profiles to base types, and from base type to extended types. In step 4 it uses the normalized TBox $\mathcal{T}'$ from step 2 and the $\mathbb{T}$ including the indexes stored in step 3 to encode the core rules that enforce all possible $\mathbb{T}$-assignment including the role hierarchy of τ' ($\mathcal{P}_{\mathbb{T}}$). In step 5, we encode $\mathcal{A}$ as a set of datalog facts, ensuring that the fact $prof_p(a)$ ($\mathcal{P}_{\mathcal{A}}$) is present for each individual a with profile $p = \mathsf{prof}^{\mathcal{A}}(a) \in \mathbb{P}$.

- In the online phase it takes as an input a query q and uses the type table $\mathbb{T}$ stored in the database to encode the query into an ASP program $\mathcal{P}_q$. Lastly it merges $\mathcal{P}_q$ with $\mathcal{P}_{\mathcal{A}}$ and $\mathcal{P}_{\mathbb{T}}$ and evaluates it using clingo.

3.3.2 Ontologies

Ultimately, the algorithms designed in this thesis reason over the computed structure $\mathbb{T} = (\mathbf{L}, \mathbf{S})$ in order to answer the queries. Therefore when designing the tests for expressive DLs, as important as testing for the size of the set of profiles covering the ABox is, so is testing for the size and feasibility of computing $\mathbb{T} = (\mathbf{L}, \mathbf{S})$ from profiles by the type table algorithm. This is a necessary precondition towards fulfilling the goal ($G3$) of this thesis.

In order to design proper benchmarks for assessing the $\mathbb{T}$ computation algorithm and the query answering algorithms that reason over them in chapter 4 and 5, we had to keep the following in mind: while the size of the set of profiles depends entirely on the size and structure of the data, the size of $\mathbb{T}$ depends on the number of profiles, but even more so on the combinatorics that the TBoxes encode, hence the larger and more expressive a TBox is, the more likely the $\mathbb{T}$ computation algorithm may under perform.

Finding suitable benchmarking ontologies that come with real-world data is a challenge that the DL community faces, and in fact, finding ontologies with TBox expressed in $\mathcal{ALCH}$ and a large ABox was impossible. However, since the process of obtaining the profiles is dependent exclusively on the data, for evaluating the size of profiles w.r.t. the data any ontology with preferentially large real-world ABox was seen as a good option. On the other hand, for testing the feasibility of algorithms (i.e. addressing ($G3$)), we considered all ontologies with TBoxes expressed in DLs that contain $\mathcal{ALC}$, and have ABoxes.

After extensive search we have identified three real-world ontologies with large data (ABoxes), and a repository[2] with numerous ontologies, some of which quite complex. While the first three played a pivotal role on showing that a moderate number of profiles cover even large ABoxes (important for addressing ($G1$)), the large repository helped us show that even for quite complex ontologies our approach is feasible for a lot of cases, i.e., showing that ($G3$) was reached. Towards showing that the ($G1$) is addressed in full, in Chapter 7 we show that profiles can be obtained from OBDA specifications with mappings expressed in R2RML.

Ontologies with real world data we considered include:

- NPD, a petroleum ontology, and
- IMDb, a film ontology, and
- MyITS, a transport ontology.

NPD: stands for Norwegian Petroleum Directorate, which is a body that regulates and monitors the exploitation of oil and gas in Norway. NPD dataset in fact refers to NPDs

[2] https://www.cs.ox.ac.uk/isg/ontologies/

FactPages[3] that are mostly known to the semantic web and description logic community due to EU Optique[4] project that adopted it as one of the two real-world use cases from the energy sector. NPD FactPages record all kind of activities of companies who operate petroleum fields in Norway, including numerous other facts such as seismic activities in Norwegian Continental Shelf (NCS) etc. The data spans from 1970s and serve as the basis for future planning of one of the major stake holders in the Norwegian economy. The ontology we used here, was obtained from [GKT17] which included data in ABox format, with TBox expressed in Horn-$\mathcal{SHIF}$. This ontology fit our purpose of testing for the size of the profile set covering the ABox, due to its sizeable real-world data from an important application scenario.

MyITS: stands for My Personalized Intelligent Mobility Service; a project that was developed by Austrian Institute of Technology together with Vienna University of Technology with a goal of building new approaches to routing services that would incorporate user intentions during route search. The project utilized Semantic Web technologies to achieve its goals, and an ontology (TBox) was developed. The project also published a methodology [EPS+15] on how to obtain real world data for the provided ontology from OpenStreetMap[5] data and transform it into an ABox. This was of particular interest to us, since it provided a tool for shaping and obtaining RDF data from real world geo spatial data.

Following the methodology in [EPS+15] we extracted the data describing the city of Vienna, which were taken as database dumps at BBBike[6]. The extracted data contains facts about 19517 geographical points in the map treated as individuals. Concept assertions were extracted from tags in the mapping data, for points of interest that correspond to a particular point like Hotel, Restaurant, Shop, Hospital, MetroStation etc. There are also role assertions involving these points and other individuals representing objects of interest such as metro lines, types of cuisine, dishes they offer etc. Further more we modeled the transport network, and we manually extracted a binary relation next, relating pairs of points whose distance is below a certain threshold set in meters. By considering different thresholds, ranging from 50 to 250 meters, we obtained ABoxes of different sizes. Other roles extracted to describe the Vienna metro network are locatedAlong and nextStation. The former relates a metro station to the corresponding metro line, and the latter relates pairs of consecutive stations on the same line. The extracted relations that also occur in the TBox include roles like hasCuisine and serves, which relate a Restaurant to a Cuisine or a Dish, respectively. Originally the ontology was in *DL-Lite*, but we enriched it with $\mathcal{ALCH}$ axioms like the following:

[3] https://npdfactpages.npd.no/factpages/
[4] http://optique-project.eu/
[5] https://www.openstreetmap.org
[6] http://download.bbbike.org/osm/bbbike/Wien/

$$\begin{aligned} \mathsf{Restaurant} &\sqsubseteq \mathsf{LocalRestaurant} \sqcup \mathsf{InternationalRestaurant} \\ \exists \mathsf{serves.Schnitzel} &\sqsubseteq \mathsf{LocalRestaurant} \\ \exists \mathsf{hasCuisine.Japanese} &\sqsubseteq \mathsf{InternationalRestaurant} \end{aligned}$$

The ontologies we got from this methodology contained large ABoxes obtained from real world data and as such were useful for testing the size of the profiles covering them. Also, the TBox obtained was in $\mathcal{ALCH}$, hence they were seen as suitable for assessing $(G3)$ for algorithms in Chapter 4 and 5. Moreover, since we obtained ABoxes of sizes ranging from 125 thousand to 1 million assertions, they helped us observe the behaviour of algorithms executed over scaled examples.

IMDb: stands for Internet Movie Database, the most comprehensive movie database on the web. Here, it represents the movie ontology with a sizeable data set represented as an ABox obtained from [GKT17]. The original movie ontology[7] was developed by University of Zurich, with an aim to model the movie application domain through providing concept categorization for movies and relationship with other concepts of interest. The movie ontology we got from [GKT17] is expressed in Horn-$\mathcal{SHOIF}$. Our primary purpose for selecting this source in particular had to do with the large size of its ABox obtained from IMDb with over four million assertions, and as such fit the purpose of assessing $(G1)$.

Oxford Ontology Repository: is an online repository[8] of 787 ontologies collected by the Oxford University from numerous studies. Only the ontologies with data (ABoxes) were of interest to us. Most of the ontologies had small ABoxes, which from our inspection looked as ABoxes representing possible combinations that appear in the data. We focused only on those that included role assertions in their ABoxes, as this allowed us to assess the generation of base type w.r.t. potential guesses resulting from incoming and outgoing roles of profiles obtained from their ABoxes.

From 787 ontologies, 370 had ABoxes, out of which 95 were extensions of $\mathcal{ALC}$ that contained role assertions. From them, 6 ontologies yielded an error while loading with OWLApi, which left us with 89 ontologies that were interesting for our purpose of assessing $(G3)$. Particularly they were interesting for assessing the feasibility of type table computation due to their TBox sizes and composition. In Figure 3.3 a graph showing the composition of the selected 89 ontologies is given. In the x-axis the file names of the ontologies as they appear in the repository are given,[9] where as in the y-axis different factors of interest in the ontology are given, where CN stands for concept names, RN for role names, CA for concept assertions, RA for role assertions, IND for individuals, TBox for TBox axioms, NF1—NF5 for axioms of the respective $\mathcal{ALCH}$ normal form. The

[7] http://www.movieontology.org/
[8] https://www.cs.ox.ac.uk/isg/ontologies/
[9] https://www.cs.ox.ac.uk/isg/ontologies/

colour coded dots show the size (number) of each factor, where the green color stands for the numbers between 1 and 100, the blue for the number between 101 and 1000, and the red for the number above 1000. The purpose of this graph is to show the composition of the selected ontologies for testing. As can be observed from the graph we obtained a fair mix of ontologies of different sizes, moreover, the colors show that most of the TBoxes were quite large.

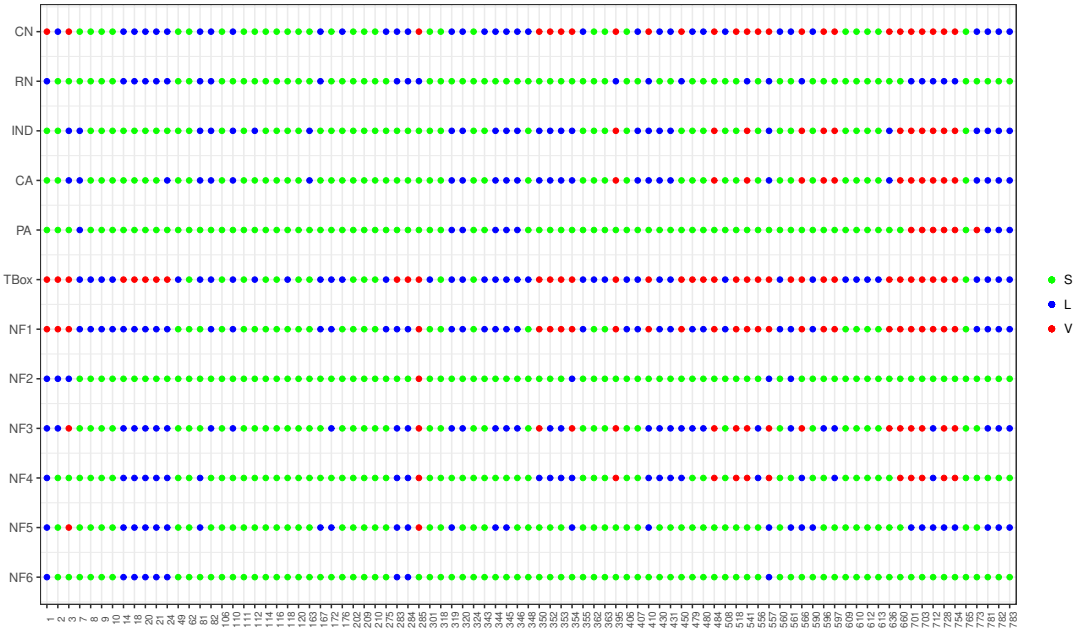


Figure 3.3: Key features of the selected ontologies.

3.4 Evaluation

In this section we show the results obtained from evaluating the size of profiles for showing that they fulfill the objectives from ($G1$), and results obtained from testing the feasibility of computing $\mathbb{T}$ which is a precondition for feasibility of our approach in general, i.e., for reaching the goal ($G3$).

For evaluating the size of profiles we used the three real-world ontologies with large datasets: NPD, MyITS and IMdb. We used Mod4Q to obtain the profiles from their ABoxes and report the results in Table 3.2. As can be observed the sizes of profiles obtained were small compared to the size the large ABoxes covered. This confirmed our expectations, since databases in real-world are very regular, i.e., they record multiple copies of individuals belonging to certain classes.

Ontology	$\lvert\mathcal{A}\rvert$	$\lvert\mathsf{N_I}(\mathcal{A})\rvert$	$\lvert\mathbb{P}\rvert$	$\lvert\mathbb{P}\rvert/\lvert\mathsf{N_I}(\mathcal{A})\rvert$
MyITS50	125K	20K	206	0.01056
MyITS150	501K	20K	206	0.01056
MyITS250	1073K	20K	206	0.01056
NPD	856K	1510K	173	0.00011
IMDb	4736K	3765K	190	0.00005

Table 3.2: Size of $\mathbb{P}$ for real world ontologies.

For evaluating the feasibility of compiling models into $\mathbb{T}$ we used the selected ontologies of Oxford Ontology Repository as explained in the Subsection 3.3.2 and run Mod4Q on them. For roughly for 80% of the ontologies (70) Mod4Q computed the model representation $\mathbb{T}$ successfully, while for the remaining 19 cases the computation was infeasible since we got $\geq 2^{15}$ base types for some profiles. In Figure 3.4 a distribution of the cases in which Mod4Q succeed is given.

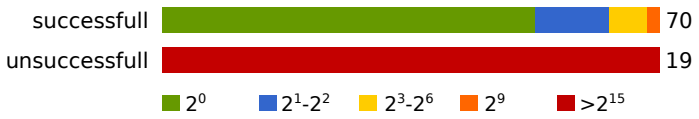


Figure 3.4: Ontologies categorised by successful computation of $\mathbb{T}$ and the maximal number of base types per profile.

Moreover, the graph in Figure 3.5 shows that the same graph in Figure 3.3 restricted only to successful instances, in which it can be observed that the type table computation algorithm succeeded in large complex ontologies, some of which included significant number of non-deterministic type of axioms NF2 and NF3'. From our observations, the size of the TBox as long as we could load it in our hardware didn't play a role in failure. The main bottleneck of our approach was the high number of guesses we got for 19 ontologies.

	CN	$\lvert\mathcal{T}\rvert$	$\lvert\mathcal{T}'\rvert$	$\lvert\mathbb{P}\rvert$	BT_{max}	GT	$\mathcal{P}_{\mathbb{T}}$
DOLCE21	0.3K	1.3K	0.7K	11	2^9	17	14s
Gardin.81	0.3K	0.4K	0.3K	27	2^0	224	0.3s
Gardin.283	0.2K	1K	0.5K	11	2^6	17	9.4s
Gardin.284	0.3K	1.3K	0.7K	22	2^1	73	9.5s
OBI350	3.2K	10K	0.4K	38	2^{25}	–	–
OBO354	4.5K	7.2K	0.6K	9	2^0	69	0.4s
WINE781	0.6K	0.7K	0.2K	62	2^{28}	–	–

Table 3.3: Compilation details for selected complex ontologies.

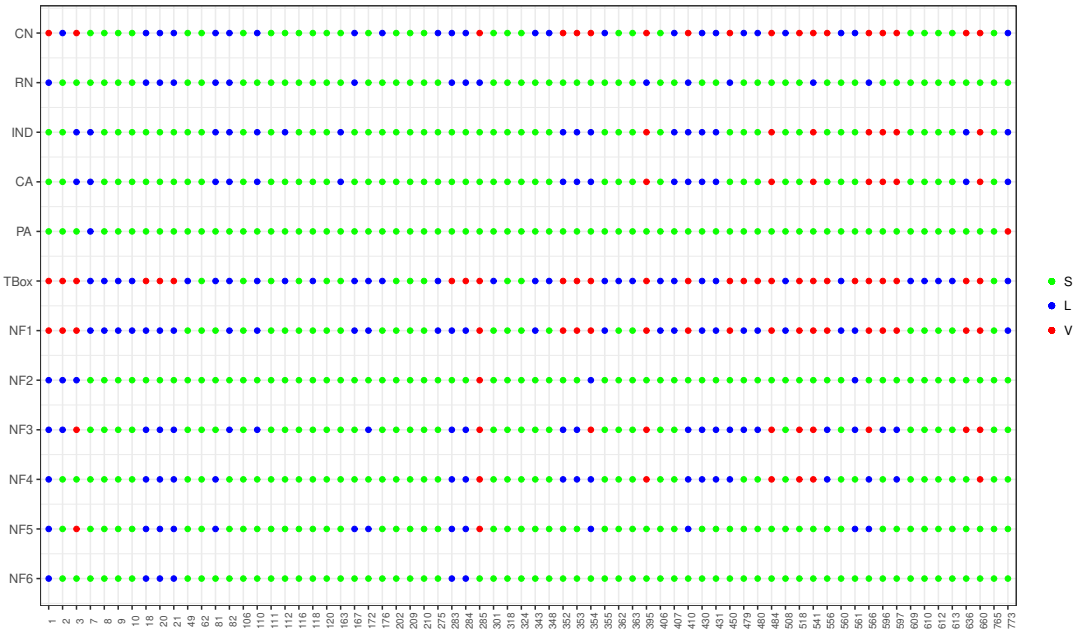


Figure 3.5: Key features of the selected ontologies with successful computation of $\mathbb{T}$.

For the successful cases, on average the number of profiles per ontology was 10, while the number of base types 24, and the number of computed good types was 23. Moreover the time for computing $\mathbb{T}$ and producing the encoded rules in program $\mathcal{P}_{\mathbb{T}}$ ranged from 171 milliseconds to 14 seconds, with an average of 1.3 seconds. We report selected results in Table 3.3, where CN denotes the number of concept names occurring in $\mathcal{T}$, $|\mathcal{T}|$ the size of the TBox, $|\mathcal{T}'|$ the size of the TBox reduced only to axioms of forms (NF2, NF3') in the normalized TBox, $BT_{max}{=}max_{p\in\mathbb{P}}(\mathsf{btyp}^{\mathcal{T}}(p))$, and $GT = |GT_{\mathbb{T}}(\mathbb{P})|$.

Our experiments showed that:

- In most cases, the number of base types generated from the profiles used in the ABox is sufficiently small.

• Our model compilation can handle complex ontologies. Both the number of good types in the computed $\mathbb{T}$, and the time required to produce $\mathbb{T}$, were very small even for ontologies with thousands of concept names and axioms.

Not surprisingly, the evaluation on complex ontologies made apparent that, while computing all the base types for the given profiles is feasible in most cases, it is also the main bottle neck of our approach. Indeed, in all cases where our prototype failed to compute the model compilation, there were profiles with over 9 relevant guesses, thus 2^9 base types. An interesting observation is that some ontologies, like *DOLCE21*, have a large number of base types, but result in few good types that are often shared by profiles.

3.5 Discussion and Related Work

In this chapter we have presented a middle ground approach to data dependent and data independent query rewritings which utilises a set of profiles, an abstract way of defining the structure of families of ABoxes, from which we can compute a structure $\mathbb{T} = (\mathbf{L}, \mathbf{S})$ that represents all the relevant models for answering queries. We note that similar directions to using the structure of the ABox exist, albeit in different fashion. The abstraction refinement technique of [GKT17, GKL$^+$14] builds on similar intuitions, but tackles ABox materialization in Horn DLs. It works in two stages: the *abstraction* stage in which the ABox individuals are partitioned into equivalence classes based on the asserted information, and a representative individual is used for identifying the equivalence class. In the second stage, named the *refinement* stage the equivalence classes are split when new assertions are derived that distinguish individuals represented by the same class. In comparison to our representation the equivalent classes obtained in the first stage coincide with our profiles, however they are used for running the materialization over them and transferring the entailments to the original ABox using homomorphisms. The process of obtaining abstractions, computing materializations and transferring them back to the ABox is repeated iteratively until no new entailments can be added to the ABox. Hence, abstract ABoxes are used to drive the materialization and changes in the original ABox iteratively, and get updated after each abstraction step. Moreover, they are based on a concrete ABox. The technique is tailored for Horn-$\mathcal{SHOIF}$. Adding disjunctive axioms may require to keep track of multiple ABoxes during the course of computation which requires a complicated machinery and seems infeasible. Moreover, it is not obvious how this approach tailored for the task of materialization can be efficiently adopted for answering different query languages.
Another approach, less related to ours, but which exploits the structure of the ABox is the one by [WM12]. Here ontology modularization is used for addressing the problem of instance checking for description logic $\mathcal{SHI}$. The basic idea is to partition the ABox into smaller parts, called 'individual islands': that are sufficient for computing the entailed assertions. This way the problem of computing becomes parallelizable since multiple instances of reasoners can be used concurrently for reasoning on 'individual islands' supposedly much smaller than the original ABox. The authors propose several ways of splitting the ABox. The first one known as the *component based* modularization,

partitions the ABox into connected components, i.e., if two individuals are members of some role assertion in the ABox they end up in the same module. Since this naive approach in realistic cases may yield few large modules which is unsatisfactory, the authors propose a more involved technique. In *intensional-based* modularization individuals that are connected via some role assertion may be split in different modules if certain conditions are met. Although their technique utilizes the structure of the ABox with the goal of reducing the size of the ABoxes, from their work is not obvious how one can extend their approach to query languages beyond instance queries.

The aforementioned techniques are appealing and scale well for large ABoxes, however they are built around concrete ABoxes, which requires them to run the whole procedure online. In contrast our computation of type tables can be done offline and be reused for any ABox that is covered by the given profiles. The check for ABox coverage consist on simple partition of the ABox which can be implemented efficiently. Moreover, the techniques mentioned are complete only for the task of ontology materialization and instance query answering. To the best of our knowledge our approach is the first to utilise the structure of ABoxes which can be leveraged by query rewriting algorithms, and that targets expressive DLs, ($\mathcal{ALCHI}$). Moreover the simple structure representation via profiles has its advantages: (i) we can efficiently check if an individual in some given ABox is covered by the given set of profiles; and (ii) obtaining profiles from practical settings such as from the OBDA settings is possible and feasible as we show in Chapter 7. Another interesting observation from our experiments, is that even for large ABoxes we witnessed few profiles that cover them. This may sometimes be partially explained by automated processes that produce the data (mappings, forms, scripts, etc), which naturally restrict its shape. However, we expected this results since in practice it is common for databases to store multiple copies of the same patterns. Currently, the main bottleneck of our algorithm is the computation of 'base types' from profiles, which expands the profiles with sets of guesses. Although the number of guesses was usually small for the considered ontologies, it became unmanageable in roughly 20% of the cases, therefore investigating more refined alternatives than current naive guessing seems crucial.

CHAPTER 4

Query Answering in Expressive DLs

In this chapter we present OMQ answering algorithms for expressive ontologies in $\mathcal{ALCHI}$ that are designed to utilise the computed type table $\mathbb{T}$. We illustrate the usefulness of our model representation via $\mathbb{T}$ with sound and complete algorithms for three different query languages: (a) *instance queries* (IQs), (b) a restricted class of regular path queries (RPQs) known as *reachability queries* (RQs), and (c) a query language we call *semi-full conjunctive queries with reachability atoms* that combines RQs and CQs with restrictions on the existential quantification of variables. Our approach is modular, and a significant part of the computation is done offline; we have shown in the previous chapter that in most of the tested cases computing the type table $\mathbb{T}$ is feasible and that they are usually of tameable size. This leaves a lesser computational burden on the algorithms that utilize $\mathbb{T}$ to answer queries, and as such makes them suitable candidates for practical consideration. As we will show in our experiments with our proof of concept implementation Mod4Q, they behaved well computationally against the ontologies for which we could compute the structure $\mathbb{T}$, both in terms of the size of the data as well as the size of the ontology expressed in $\mathcal{ALCH}$.

We focus on queries that are preserved under homomorphisms, we call such queries *monotone*.

Definition 22 (Monotone query)**.** *We call a query q* monotone *if $\mathcal{I} \triangleright \mathcal{J}$ and $\mathcal{I} \models q$ implies $\mathcal{J} \models q$. We call our OMQ $(\mathcal{T}, q)$ monotone if q is monotone.*

The nice feature of our model representation $\mathbb{T}$ is that it represents all the relevant models for answering any monotone OMQ over any ABox covered by the profiles. In fact, from Theorems 1 and 2 we get the following lemma:

Lemma 3. *Let $(\mathcal{T}, q)$ be a monotone OMQ, $\mathbb{P}$ a set of profiles, and $\mathbb{T}$ a $\mathcal{T}$-complete type table that covers $\mathbb{P}$. Given an ABox $\mathcal{A}$ covered by $\mathbb{P}$, we have $(\mathcal{T}, \mathcal{A}) \models q$ iff for each $\mathcal{I} \in mods(\mathcal{A}, \mathbb{T})$ we have $\mathcal{I} \models q$.*

Proof. ($\Rightarrow$) Let $\mathcal{I} \models q$ for each $\mathcal{I} \in mods(\mathcal{A}, \mathbb{T})$. We want to prove that $(\mathcal{T}, \mathcal{A}) \models q$. Let's assume that $(\mathcal{T}, \mathcal{A}) \not\models q$ then there exists an interpretation $\mathcal{J} \models (\mathcal{T}, \mathcal{A})$ such that $\mathcal{J} \not\models q$. From the Theorem 2 we know that there exists an interpretation $\mathcal{I} \in mods(\mathcal{A}, \mathbb{T})$ s.t. $\mathcal{J} \triangleright \mathcal{I}$, but then since q is a monotone query and $\mathcal{I} \models q$ we get that $\mathcal{J} \models q$ which is a contradiction, therefore we established that $(\mathcal{T}, \mathcal{A}) \models q$.

($\Leftarrow$) Let $(\mathcal{T}, \mathcal{A}) \models q$. We want to prove that then for each $\mathcal{I} \in mods(\mathcal{A}, \mathbb{T})$, $\mathcal{I} \models q$. We proceed by contrapositive, i.e. let $\mathcal{I} \in mods(\mathcal{A}, \mathbb{T})$ s.t. $\mathcal{I} \not\models q$. By Theorem 1 we have that $\mathcal{I} \models (\mathcal{T}, \mathcal{A})$, and since $\mathcal{I} \not\models q$ it follows that $(\mathcal{T}, \mathcal{A}) \not\models q$ as well. □

Note that the above lemma is important since practically all the families of queries that have been considered in the context of DLs are monotone, including conjunctive queries (CQs), regular path queries (RPQs), fragments of Datalog, etc. In fact, decidability results for non-monotone OMQs are very limited, e.g., [GIKK15].

The set $\mathbb{T}$ represents all relevant models, but we need to test query entailment over all represented models in $\mathbb{T}$, which may be infinitely many and of infinite size. Therefore, different query languages need different algorithms for $\mathbb{T}$. We present in this chapter algorithms for evaluating queries of different query languages over all the relevant models using the computed type table $\mathbb{T}$. We use these algorithms to rewrite the given queries into ASP programs, reducing OMQ answering to cautious entailment.

The rest of this chapter is organised as follows. In Section 4.1 an ASP encoding for answering IQs and an explanation of why such encoding is suitable for ABox materialization is given. In Section 4.2 we present an encoding of RQs. In Section 4.3 we show how results from previous sections can be combined to form an interesting query language which can be viewed as a conjunction of CQs and multiple RQs, named *semi-full CQs with reachability atoms* (s-CRQ). In Section 4.4 we show promising results of experiments carried with a proof concept implementation Mod4Q over a range of ontologies, and finally we close the chapter with the Section 4.5 on related work and discussions.

4.1 Instance Queries and ABox Materialization

In this section we present a direct algorithm which makes use of $\mathbb{T}$ to answer instance queries and show how it can be encoded into an ASP program.

For each relevant model $\mathcal{I}$ of $(\mathcal{T}, \mathcal{A})$, from Theorem 2 we have that there exists an interpretation $\mathcal{J} \in mods(\mathcal{A}, \mathbb{T})$ such that $\mathcal{J} \triangleright \mathcal{I}$, and from Theorem 1 we have that $\mathcal{J} \models (\mathcal{T}, \mathcal{A})$. Then from the way $\mathcal{J}$ is constructed we have that types of individuals $\mathsf{N_I}(\mathcal{A})$ coincide with one of the possible $\mathbb{T}$-assignments. Therefore for answering any instance

query $q = A(x)$, it suffices to iterate through each individual $a \in \mathsf{N_I}(\mathcal{A})$, and check if for each possible $\mathbb{T}$-assignment t we have that A is contained in $t(a)$.

We consider a set of ABoxes which intuitively capture the models of $mods_t(\mathcal{A}, \mathbb{T})$ for different $\mathbb{T}$-assignments t, which then allows us to test for an entailment of some assertion α, i.e. we test $\mathcal{I} \models \alpha$ for all $\mathcal{I} \in mods(\mathcal{A}, \mathcal{T})$ by checking if $\mathcal{I} \models \alpha$ for any t such that $\mathcal{I} \in mods_t(\mathcal{A}, \mathcal{T})$.

Definition 23. *Let t be a $\mathbb{T}$-assignment. $\mathcal{A}^t$ is the smallest $\mathcal{A} \subseteq \mathcal{A}^t$ such that:*
(a1) $A(a) \in \mathcal{A}^t$ for all $A \in t(a)$ and all $a \in \mathsf{N_I}(\mathcal{A})$.
(a2) $s(a, b) \in \mathcal{A}^t$ for each $r(a, b) \in \mathcal{A}$ and $r \sqsubseteq^*_{\mathcal{T}} s$.

The ABoxes $\mathcal{A}^t$ capture the entailment of assertions in all the models in $mods_t(\mathcal{A}, \mathbb{T})$.

Lemma 4. *Let α be an assertion, then $\mathcal{I} \models \alpha$ for all $\mathcal{I} \in mods(\mathcal{A}, \mathbb{T})$ iff $\alpha \in \mathcal{A}^t$ for each $\mathbb{T}$-assignment function t.*

Proof. ($\Rightarrow$) First we prove that if $\alpha \in \mathcal{A}^t$ for each $\mathbb{T}$-assignment function t, then $\mathcal{I} \models \alpha$ for each $\mathcal{I} \in mods(\mathcal{A}, \mathbb{T})$. Let α be an arbitrary assertion such that $\alpha \in \mathcal{A}^t$ for each possible $\mathbb{T}$-assignment function t. We want to show that for any $\mathcal{I} \in mods(\mathcal{A}, \mathbb{T})$ we have that $\mathcal{I} \models \alpha$. Let $\mathcal{I}$ be an arbitrary interpretation in $mods(\mathcal{A}, \mathbb{T})$. Note that by Definition 21 $\mathcal{I}$ is an $(\mathcal{A}, t, \mathbb{T})$-interpretation. We have two cases:

- α is of the form $A(a)$. By construction of $\mathcal{I}$ we have that for each $a \in \mathsf{N_I}(\mathcal{A})$ $type^{\mathcal{I}}(a) = t'(a)$ for some $\mathbb{T}$-assignment function t'. By our assumption we have $A(a) \in \mathcal{A}^{t'}$ as well, then from the condition *(a1)* of $\mathcal{A}^t$ definition we get that $A \in t'(a)$, hence we get that $A \in type^{\mathcal{I}}(a)$, i.e. $\mathcal{I} \models A(a)$.

- α is of the form $r(a, b)$. By construction of $\mathcal{I}$ we have that for each $s(a, b) \in \mathcal{A}$ and $s \sqsubseteq^*_{\mathcal{T}} r$, $(a, b) \in s^{\mathcal{I}}$, which coincides with *(a2)* of definition of $\mathcal{A}^t$ ABoxes, therefore $\mathcal{I} \models r(a, b)$.

($\Leftarrow$) Next we prove if $\mathcal{I} \models \alpha$ for each $\mathcal{I} \in mods(\mathcal{A}, \mathbb{T})$ then $\alpha \in \mathcal{A}^t$ for each $\mathbb{T}$-assignment function t. Let α be an arbitrary assertion such that $\mathcal{I} \models \alpha$ for each $\mathcal{I} \in mods(\mathcal{A}, \mathcal{T})$. Let's pick an t be an arbitrary $\mathbb{T}$-assignment function, we want to prove that $\alpha \in \mathcal{A}^t$ as well. We have two cases:

- α is of the form $A(a)$. Note that since t is a $\mathbb{T}$-assignment function from the Claim 2 we get that $mods_t(\mathcal{A}, \mathbb{T}) \neq \emptyset$. Let $\mathcal{I}$ be an arbitrary interpretation in $mods_t(\mathcal{A}, \mathbb{T})$, we have that $\mathcal{I} \in mods(\mathcal{A}, \mathbb{T})$ as well. By construction of $\mathcal{I}$ we have that for each $a \in \mathsf{N_I}(\mathcal{A})$ $type^{\mathcal{I}}(a) = t(a)$, and since $\mathcal{I} \models \alpha$ we get that $A \in t(a)$. Then, by the condition *(a1)* of A^t definition we get that $A(a) \in \mathcal{A}^t$ as well.

- α is of the form $s(a,b)$. There are two cases (i) $\alpha \in \mathcal{A}$ in which case $\alpha \in \mathcal{A}^t$ as well by definition, and (ii) there exists a $r(a,b) \in \mathcal{A}$ s.t. $s \sqsubseteq^*_{\mathcal{T}} r$, hence $s(a,b) \in \mathcal{I}$ for each $\mathcal{I} mods(\mathcal{A}, \mathbb{T})$. But then, since $A^t \supseteq \mathcal{A}$ by definition of $\mathcal{A}^t$ and $r(a,b) \in \mathcal{A}$ from the condition *(a2)* of the $\mathcal{A}^t$ definition we get that $\alpha \in \mathcal{A}^t$ as well. □

From the above lemma we get that in order to decide for entailment of some assertion we need to iterate through each possible $\mathcal{A}^t$ and check if the assertion is included in each of them. Hence in order to achieve a practicable algorithm we need a language supported by scalable tools. Given that data complexity of query answering in $\mathcal{ALCHI}$ is coNP, and our need for guessing the good types, ASP is a natural candidate, also due to the fact that efficient ASP solvers like clingo and dlv have been around for quite a while with committed development teams.

To this end we present a rewriting to ASP answer sets of which characterize each $\mathcal{A}^t$ ABox. We achieve the characterization of all $\mathcal{A}^t$ ABoxes in our rewriting by encoding in the program the types assignment from $\mathbb{T}$ and encoding the conditions *(t1)* and *(t2)* of $\mathbb{T}$-assignment to enforce that the assignment of types to constants in the program corresponds to some $\mathbb{T}$-assignment function.

Note that we assume a $\mathcal{T}$-complete type table $\mathbb{T} = (\mathbf{L}, \mathbf{S})$ that covers a given set $\mathbb{P}$ containing all the profiles of interest. We use names in $\mathsf{N_C}$ as unary predicates and (possibly inverse) roles in $\overline{\mathsf{N_R}}$ as binary predicates. We also use a unary predicate $prof_p$ for each profile $p \in \mathbb{P}$ and an unary predicate $type_\tau$ for each good type $\tau \in \mathbb{T}^G$.

We split the rewriting into two parts, a program that contains the rules $\mathcal{P}_\mathbb{T}$, and a program $\mathcal{P}_\mathcal{A}$ that contains the facts related to $\mathcal{A}$. $\mathcal{P}_\mathcal{A}$ (see Figure 4.1 below) represents a given ABox $\mathcal{A}$ via facts $r(a,b) \leftarrow$ for all role assertions $r(a,b) \in \mathcal{A}$, and facts $prof_p(a) \leftarrow$ for each $a \in \mathsf{N_I}(\mathcal{A})$ with $p = \mathsf{prof}^\mathcal{A}(a)$.

$$prof_p(a) \leftarrow \qquad \text{for each } a \in \mathsf{N_I} \qquad \text{where } p = \mathsf{prof}^\mathcal{A}(a) \tag{4.1}$$

$$r(a,b) \leftarrow \qquad \text{for each } r(a,b) \in \mathcal{A} \tag{4.2}$$

Figure 4.1: $\mathcal{P}_\mathcal{A}$ for a given $\mathcal{A}$

Note that we do not need to add the facts for concept assertions in $\mathcal{A}$ as each constant a in the program gets those from the type that is inferred by the rules in the program $\mathcal{P}_\mathbb{T}$ which is comprised of the following rules (4.3 – 4.7) shown in the Figure 4.2. Intuitively, assuming that the fact $prof_p(a)$ holds for each $a \in \mathsf{N_I}(\mathcal{A})$ with $p = \mathsf{prof}^\mathcal{A}(a)$, the rule (4.3) guesses the assignments of types to constants (individuals). Rules (4.4) and (4.5) generate the assertions *(a1)* and *(a2)* in $\mathcal{A}^t$ for each guess, while (4.6) and (4.7) enforce the neighbourhood conditions *(t1)* – *(t2)* in Definition 20.

$$\bigvee_{\tau \in GT_{\mathbb{T}}(p)} type_\tau(x) \leftarrow prof_p(x) \qquad \text{for each } p \in \mathbb{P} \tag{4.3}$$

$$A(x) \leftarrow type_\tau(x) \qquad \text{for each } \tau \in \mathbb{T}^G \text{ and each } A \in \tau \tag{4.4}$$
$$s(x,y) \leftarrow r(x,y) \qquad \text{for each } r \sqsubseteq^*_{\mathcal{T}} s \tag{4.5}$$
$$\bot \leftarrow r(x,y), A(y), \mathit{not}\ B(x) \qquad \text{for each } A \sqsubseteq \forall r^-.B \in \mathcal{T} \tag{4.6}$$
$$\bot \leftarrow r(x,y), A(x), \mathit{not}\ B(y) \qquad \text{for each } A \sqsubseteq \forall r.B \in \mathcal{T} \tag{4.7}$$

Figure 4.2: $\mathcal{P}_{\mathbb{T}}$ for instance queries in $\mathcal{ALCHI}$

Recalling the use of negation in the rewriting, and note that we use stratified negation in the context of constraints, i.e. they can not be used to derive new facts, but instead are used to prune the models that violate the encoded condition. Hence transforming the rules (4.6) and (4.7) in $\mathcal{P}_{\mathbb{T}}$ into positive rules is easy. In such a case we need to use unary predicates $\overline{A}$ which encode the fact that a certain individual is not an instance of A. An alternative rewriting in DATALOG$^\vee$ is shown in Figure 4.3.

$$\bigvee_{\tau \in GT_{\mathbb{T}}(p)} type_\tau(x) \leftarrow prof_p(x) \qquad \text{for each } p \in \mathbb{P} \tag{4.8}$$

$$A(x) \leftarrow type_\tau(x) \qquad \text{for each } \tau \in \mathbb{T}^G \text{ and each } A \in \tau \tag{4.9}$$
$$\overline{A}(x) \leftarrow \tau(x) \qquad \text{for each } \tau \in \mathbb{T}^G \text{ and each } A \in \mathsf{N_C} \setminus \tau \tag{4.10}$$
$$\bot \leftarrow A(x), \overline{A}(x) \qquad \text{for each } A \in \mathsf{N_C} \text{ and} \tag{4.11}$$
$$\tag{4.12}$$
$$s(x,y) \leftarrow r(x,y) \qquad \text{for each } r \sqsubseteq^*_{\mathcal{T}} s \tag{4.13}$$
$$B(x) \leftarrow r(x,y), A(y) \qquad \text{for each } \exists r.A \sqsubseteq B \in \mathcal{T} \tag{4.14}$$
$$B(y) \leftarrow r(x,y), A(x) \qquad \text{for each } A \sqsubseteq \forall r.B \in \mathcal{T} \tag{4.15}$$

Figure 4.3: ASP rewriting of $\mathcal{P}_{\mathbb{T}}$ with positive rules.

The rules in Figure 4.3 in figure resamble closely the rules in the chosen rewriting in Figure 4.2. The new program has two additional types of rules 4.10 and 4.11, which together with the rules 4.14 and 4.15 encode the behaviour of the original rules (4.6) and (4.7), namely they ensure that each constant gets only the concepts from the type assignment and that those concepts do not violate one of the $\mathbb{T}$-assignment conditions.

Note that the answer sets of $\mathcal{P}_\mathbb{T} \cup \mathcal{P}_\mathcal{A}$ are in close correspondence with the ABoxes $\mathcal{A}^t$. This is due to the rules 4.14 and 4.15. The reduct of $\mathcal{P}_\mathbb{T} \cup \mathcal{P}_\mathcal{A}$ with respect to a proposed model M will not justify some assignment that deviates from the $\mathbb{T}$-assignments. Since each $\mathcal{A}^t$ reflects some $\mathbb{T}$-assignment t, the answer sets of $\mathcal{P}_\mathbb{T} \cup \mathcal{P}_\mathcal{A}$ coincide with the ABoxes $\mathcal{A}^t$. Therefore, answering an IQ q amounts to cautious entailment over the program $\mathcal{P}_\mathbb{T} \cup \mathcal{P}_\mathcal{A}$. As anticipated, the rewriting does not depend on a specific ABox, but only on a $\mathbb{T}$ that covers a set $\mathbb{P}$ of profiles, and it can be used for answering q over any ABox that is covered by $\mathbb{P}$.

From the fact that the answer sets of $\mathcal{P}_\mathbb{T} \cup \mathcal{P}_\mathcal{A}$ coincide (on the common vocabulary) with the ABoxes $\mathcal{A}^t$ for the different $\mathbb{T}$-assignments, and Lemma 4 we get that the next theorem follows:

Theorem 3. *Let $\mathcal{A}$ be an ABox covered by $\mathbb{P}$. For any assertion α, we have $(\mathcal{T}, \mathcal{A}) \models \alpha$ iff $\alpha \in M$ for all answer sets M of $\mathcal{P}_\mathbb{T} \cup \mathcal{P}_\mathcal{A}$.*

Example 8. *Let $\mathcal{A}, \mathcal{T}, \mathbb{T}$ be taken from the running example in the previous chapter, more precisely refer to the ABox $\mathcal{A}_1$ and profiles from Example 2, and $\mathcal{T}$-complete $\mathbb{T}$ from Example 4. Let $q_1 = A(x)$, $q_2 = C_1(x)$. We have get the following ASP encoding (here we use τ_i and p_i rather than $type_{\tau_i}$ and $prof_{p_i}$ as predicates):*

$$\mathcal{P}_\mathcal{A} = \{ \quad p_1(a) \leftarrow . \quad p_2(b) \leftarrow . \quad p_3(c) \leftarrow . \quad r(a,b) \leftarrow . \quad s(b,c) \leftarrow . \}$$

$$\begin{aligned}
\mathcal{P}_\mathbb{T} = \{ \quad & \tau_1(X) \vee \tau_{21}(X) \leftarrow p_1(X). \\
& \tau_1(X) \vee \tau_{21}(X) \vee \tau_3(X) \vee \tau_{41}(X) \leftarrow p_2(X). \\
& \tau_{51}(X) \vee \tau_{52}(X) \vee \tau_{61}(X) \vee \tau_{62}(X) \leftarrow p_3(X). \\
& \\
& A(X) \leftarrow \tau_1(X). \quad B(X) \leftarrow \tau_1(X). \\
& A(X) \leftarrow \tau_{21}(X). \quad B(X) \leftarrow \tau_{21}(X). \quad C(X) \leftarrow \tau_{21}(X). \quad C_1(X) \leftarrow \tau_{21}(X). \\
& B(X) \leftarrow \tau_3(X). \\
& B(X) \leftarrow \tau_{41}(X). \quad C(X) \leftarrow \tau_{41}(X). \quad C_1(X) \leftarrow \tau_{41}(X). \\
& C(X) \leftarrow \tau_{51}(X). \quad C_1(X) \leftarrow \tau_{51}(X). \\
& C(X) \leftarrow \tau_{52}(X). \quad C_2(X) \leftarrow \tau_{52}(X). \\
& A(X) \leftarrow \tau_{61}(X). \quad C(X) \leftarrow \tau_{61}(X). \quad C_1(X) \leftarrow \tau_{61}(X). \\
& A(X) \leftarrow \tau_{62}(X). \quad C(X) \leftarrow \tau_{62}(X). \quad C_2(X) \leftarrow \tau_{62}(X). \\
& \\
& s(X,Y) \leftarrow r(X,Y). \\
& \\
& \bot \leftarrow r(x,y), A(y), \mathit{not}\ C(x). \\
& \bot \leftarrow r(x,y), C(x), \mathit{not}\ B(y). \\
& \bot \leftarrow s(x,y), C_2(x), \mathit{not}\ A(y). \qquad \}
\end{aligned}$$

The ASP program $\mathcal{P}_\mathbb{T} \cup \mathcal{P}_\mathcal{A}$ has eight answer sets:

$$
\begin{aligned}
M_1 = \{&\tau_1(a), \tau_3(b), \tau_{51}(c), A(a), B(a), B(b), C(c), C_1(c),\\
&r(a,b), s(a,b), s(b,c)\}.\\
M_2 = \{&\tau_1(a), \tau_3(b), \tau_{52}(c), A(a), B(a), B(b), C(c), C_2(c),\\
&r(a,b), s(a,b), s(b,c)\}.\\
M_3 = \{&\tau_1(a), \tau_{41}(b), \tau_{61}(c), A(a), B(a), B(b), C(b), C_1(b), A(c), C(c), C_1(c),\\
&r(a,b), s(a,b), s(b,c)\}.\\
M_4 = \{&\tau_1(a), \tau_{41}(b), \tau_{62}(c), A(a), B(a), B(b), C(b), C_1(b), A(c), C(c), C_2(c),\\
&r(a,b), s(a,b), s(b,c)\}.\\
M_5 = \{&\tau_{21}(a), \tau_1(b), \tau_{51}(c), A(a), B(a), C(a), C_1(a), A(b), B(b), C(c), C_1(c),\\
&r(a,b), s(a,b), s(b,c)\}.\\
M_6 = \{&\tau_{21}(a), \tau_1(b), \tau_{52}(c), A(a), B(a), C(a), C_1(a), A(b), B(b), C(c), C_2(c),\\
&r(a,b), s(a,b), s(b,c)\}.\\
M_7 = \{&\tau_{21}(a), \tau_{21}(b), \tau_{61}(c), A(a), B(a), C(a), C_1(a), A(b), B(b), C(b), C_1(b),\\
&A(c), C(c), C_1(c), r(a,b), s(a,b), s(b,c)\}.\\
M_8 = \{&\tau_{21}(a), \tau_{21}(b), \tau_{62}(c), A(a), B(a), C(a), C_1(a), A(b), B(b), C(b), C_1(b),\\
&A(c), C(c), C_2(c), r(a,b), s(a,b), s(b,c)\}.
\end{aligned}
$$

As seen from the answer sets M_i, the types of the individuals here coincide with the $\mathbb{T}$-assignments t_i from Example 7. From the answer sets above we get that q_1 has only a as an answer since the atom $A(a)$ is found in all the answer sets, where q_2 has no answers since as no atom $C_1(z)$ for some constant z is found in all the answer sets.

Observe that to answer any instance queries, all we need is to consider all answer sets of the program $\mathcal{P}_\mathcal{A} \cup \mathcal{P}_\mathbb{T}$, i.e. no additional reasoning on top of the answer sets is needed. For answering any given instance query one needs to take the intersection of all the answer sets, and then gather the atoms whose predicate matches the concept/role of the instance query.

Our rewriting is suitable for the task of ABox materialization, which consists on materializing the instances of concept and role names in the ontology. The fact that we compute the answer sets only once and we can reuse them to decide entailment of all possible assertions, makes our approach practicable for this task. A simple algorithm like the following would suffice:

(1) Compute answer sets of $\mathcal{P}_\mathcal{A} \cup \mathcal{P}_\mathbb{T}$.

(2) Compute the intersection of the answer sets obtained in (1).

(3) Add the atoms obtained from (2) into the ABox.

Since steps (1) and (2) are anyway performed for answering any instance query, the only additional step one needs to perform for ABox materialization is step (3).

4.2 Reachability Queries

In this section we show how we can answer an interesting class of queries beyond IQs, by employing a simple algorithm that makes use of our model representation in $\mathbb{T}$. We focus here on reachability queries (RQs) a restricted class of the well-known RPQs (see e.g., [BOS15, CDLV03, CDLV02]). RQs are monotone, hence as already established they can be answered with our model compilation.

Definition 24. *A* reachability query (RQ) *q takes the form*

$$q(x) = \exists y\ r^*(x, y), C(y)$$

where r is a (possibly inverse) role, and C is of the form $A_1 \sqcap \cdots \sqcap A_n$ with $n \geq 1$ and $A_i \in \mathsf{N_C}$ for $1 \leq i \leq n$. We call x the answer variable *of q. An* ontology-mediated reachability query *(OMRQ) is a pair $(\mathcal{T}, q)$ of a TBox $\mathcal{T}$ and an RQ q.*

Let $\mathcal{I}$ be an interpretation, let $e_1, e_2 \in \Delta^{\mathcal{I}}$, and let r be a role. We say that e_1 r-reaches *e_2 if there is a sequence $d_1, \ldots, d_n$ of objects from $\Delta^{\mathcal{I}}$ such that $d_1 = e_1$, $d_n = e_2$, $n \geq 1$, and for each $1 \leq i < n$, $(d_i, d_{i+1}) \in r^{\mathcal{I}}$.*

We write $\mathcal{I} \models q(a)$, if $a^{\mathcal{I}}$ r-reaches some $d \in \Delta^{\mathcal{I}}$ with $d \in C^{\mathcal{I}}$. We call $a \in \mathsf{N_I}$ an answer *to $(\mathcal{T}, q)$ over $\mathcal{A}$, and write $(\mathcal{T}, \mathcal{A}) \models q(a)$ if $\mathcal{I} \models q(a)$ for all $\mathcal{I} \models (\mathcal{T}, \mathcal{A})$.*

Although the reachability queries defined above might seem simple, it is not hard to see their usefulness. Consider the following example.

Example 9. *Assume we have an ontology describing items in an inventory system. It may contain, e.g., the following axioms:*

$$\begin{aligned} \mathit{Phone5} &\sqsubseteq \exists \mathit{hasProcessor}.\mathit{AtmZ} \\ \mathit{PU25} &\sqsubseteq \exists \mathit{hasProcessor}.\mathit{AtmX} \\ \mathit{Watch3} &\sqsubseteq \exists \mathit{hasPart}.\mathit{PU25} \\ \mathit{Tab2} &\sqsubseteq \exists \mathit{hasPart}.\mathit{PU25} \\ \mathit{AtmX} &\sqsubseteq \mathit{AtomProcessor} \\ \mathit{AtmZ} &\sqsubseteq \mathit{AtomProcessor} \\ \mathit{hasProcessor} &\sqsubseteq \mathit{hasPart} \end{aligned}$$

Assume a dataset containing instances of concepts like Phone5, Tab2, etc. If the prices of processors of type Atom increase, we may want to find all items that contain one. Therefore it would be valuable to be able support RQs like:

$$\exists y\ \mathit{hasPart}^*(x, y), \mathit{AtomProcessor}(y)$$

Algorithm 4.1: Retrieve types that reach C through r

input : $r, C, \mathbb{T} = (\mathbf{L}, \mathbf{S})$
output: set $reach_{[r,C]}$ of types from $\mathbb{T}^G$

Let $reach_{[r,C]} := \{\tau | \tau \in \mathbb{T}^G \text{ and } C \in \tau\}$
Let $reach'_{[r,C]} := reach_{[r,C]}$
repeat
  for each $\tau \in \mathbb{T}^G$ *and* s, B *with* $s \sqsubseteq^*_{\mathcal{T}} r$ *such that* $\mathbf{S}(\tau, s, B) \cap \mathbb{T}^G \neq \emptyset$ **do**
    if $\tau' \in reach_{[r,C]}$
    for each $(\tau, (s, B), \tau') \in \mathbf{S}$ *with* $\tau' \in \mathbb{T}^G$ **then**
      $reach_{[r,C]} := reach_{[r,C]} \cup \{\tau\}$
    end
  end
until $reach'_{[r,C]} = reach_{[r,C]}$;
return $reach_{[r,C]}$

which can navigate the hasPart relation to different levels of depth and retrieve all individuals that can reach a certain individual belonging to a certain class.

We note that in $\mathcal{ALCHI}$, RQs can be reduced to instance queries by modifying the TBox (see [BCOv14]), but then the 'expensive' TBox reasoning step would depend on the query.

We now provide a technique that given a RQ q decides whether $\mathcal{I} \models q(a)$ for all models in $mods(\mathcal{A}, \mathbb{T})$. The core component of our technique is Algorithm 4.1, which runs on $\mathbb{T} = (\mathbf{L}, \mathbf{S})$. It takes as an input a role r and a conjunction of concept names $C = A_1 \sqcap \cdots \sqcap A_n$, and collects the set $reach_{[r,C]}$ of all the good types τ from $\mathbb{T}$ such that each d with $type^{\mathcal{I}}(d) = \tau$ r-reaches some $d' \in C^{\mathcal{I}}$, for every $\mathcal{I} \in mods(\mathcal{A}, \mathbb{T})$. For convenience, we write $C \in \tau$ to denote $\{A_1, \ldots, A_n\} \subseteq \tau$. Intuitively, our algorithm collects all the types τ such that, in every model, an individual that realizes τ can reach some element that satisfies the desired conjunction of concepts C.

Proposition 1. *Let $\tau \in \mathbb{T}^G$. Then $\tau \in reach_{[r,C]}$ iff for every $\mathcal{I} \in mods(\mathcal{A}, \mathbb{T})$ and every $d \in \Delta^{\mathcal{I}}$, $type^{\mathcal{I}}(d) = \tau$ implies that d r-reaches in $\mathcal{I}$ some $d' \in C^{\mathcal{I}}$.*

Proof. ($\Rightarrow$) We prove the contrapositive of the if statement, i.e. if $\tau \notin reach_{[r,C]}$ s.t. $\tau \in \mathbb{T}^G$ then there exists an $\mathcal{I} \in mods(\mathcal{A}, \mathbb{T})$ and a $d \in \Delta^{\mathcal{I}}$, where $type^{\mathcal{I}}(d) = \tau$ such that d does not r-reach any $d' \in C^{\mathcal{I}}$. Each $\mathcal{I} \in mods(\mathcal{A}, \mathbb{T})$ is a $(\mathcal{A}, t, \mathbb{T})$-interpretation, and by the construction rules in Definition 21 at each step of construction we choose successor types for each element in the domain from the $\mathbf{S}$. To this end let $\mathcal{I}$ be an arbitrary interpretation in $mods(\mathcal{A}, \mathbb{T})$ s.t. there exists an element in $d \in \Delta^{\mathcal{I}}$ where $type^{\mathcal{I}}(d) = \tau$. By construction d is a sequence of the form $as_1B_1\tau_1 \cdots s_nB_n\tau$ where $s \sqsubseteq^*_{\mathcal{T}}$. We construct, another model $\mathcal{I}'$ from $\mathcal{I}$ by:

- Let $\Delta^{\mathcal{I}'} = \Delta^{\mathcal{I}} \setminus d^*$, where d^* are all the successors of d, i.e. elements that are of the following form $as_1B_1\tau_1 \cdots s_nB_n\tau \cdots$.
- Let $r^{\mathcal{I}} = r^{\mathcal{I}} \setminus r_s^{\mathcal{I}}$ where $r_s^{\mathcal{I}}$ the set of all role membership assertion such that it's domain or range is one of the removed elements.
- Close the interpretation $\mathcal{I}'$ under the construction rules in Definition 21 such that each of the immediate successors $as_1B_1\tau_1 \cdots s_nB_n\tau sB\tau_s$ of d fulfill one of the following conditions:

  (c1) $\tau_s \notin reach_{[r,C]}$, or

  (c2) $s \not\sqsubseteq^*_{\mathcal{T}} r$

Note that we require only for immediate successors of d to fulfill (c1) or (c2). We know that each successor of d $d' = as_1B_1\tau_1 \cdots s_nB_n\tau \cdots sB\tau_s$ has as a witness some entry $(\tau, (s, B), \tau_s)$ in $\mathbf{S}$. We want to show that for each such pair (s, B) we can pick a type $\tau_s \in \mathbf{S}(\tau, (s, B))$ s.t. (c1) or (c2) above is satisfied. Let (s, B) be an arbitrary pair s.t. $\mathbf{S}(\tau, (s, B)) \neq \emptyset$, we have two cases:

- $s \sqsubseteq^*_{\mathcal{T}} r$. Assume that for each good type τ_s in $\mathbf{S}(\tau, (s, B))$ we have that $\tau_s \in reach_{[r,C]}$ then by the rules of the Algorithm 4.1 we get that $\tau \in reach_{[r,C]}$ as well, which contradicts our assumption that $\tau \notin reach_{[r,C]}$. Therefore we can pick a type in $\mathbf{S}(\tau, (s, B))$ that satisfies (c1).
- $s \not\sqsubseteq^*_{\mathcal{T}} r$ in which case (c2) is satisfied.

Since τ is a good type by Claim 1 there are good successor types in S for each pair (s, B) i.e. $\{\tau_s \in \mathbb{T}^G | \tau_s \in \mathbf{S}(\tau, (s, B))\} \neq \emptyset$ therefore $\mathcal{I}'$ is an $(\mathcal{A}, t.\mathbb{T})$ interpretation, i.e. $\mathcal{I}' \in mods(\mathcal{A}, \mathbb{T})$. Moreover, since for each immediate successor d' of d in $\mathcal{I}'$ we have that either $type(d') \notin reach_{[r,C]}$ or $(d, d') \notin r^{\mathcal{I}'}$, we get that d does not r-reach an individual $d'' \in \Delta^{\mathcal{I}'}$ such that $d'' \in C^{\mathcal{I}}$.

($\Leftarrow$) If $\tau \in reach_{[r,C]}$ where $\tau \in \mathbb{T}^G$ then for every $\mathcal{I} \in mods(\mathcal{A}, \mathbb{T})$ and every $d \in \Delta^{\mathcal{I}}$, $type^{\mathcal{I}}(d) = \tau$ implies that d r-reaches in $\mathcal{I}$ some $d' \in C^{\mathcal{I}}$. Let $\tau \in reach_{[r,C]}$ s.t. $\tau \in \mathbb{T}^G$, $\mathcal{I}$ an arbitrary interpretation in $mods(\mathcal{A}, \mathbb{T})$ and an arbitrary element $d \in \Delta^{\mathcal{I}}$ s.t. $type^{\mathcal{I}}(d) = \tau$. We want to prove that d r-reaches some element $d' \in \Delta^{\mathcal{I}}$ where $d' \in C^{\mathcal{I}}$. By the Algorithm 4.1 we have that a good type τ r-reaches a type τ_s if:

(i) $C \in \tau$, or

(ii) for some pair (s, B) s.t. $s \sqsubseteq^*_{\mathcal{T}} r$ and each good type $\tau_s \in \mathbf{S}(\tau, s, B)$ we have $\tau_s \in reach_{[r,C]}$.

Since $\tau \in reach_{[r,C]}$ and $type^{\mathcal{I}}(d) = \tau$ we are guaranteed that either $C \in type^{\mathcal{I}}(d)$ or there exists a chain of r successors between d and some other element $d' \in \Delta^{\mathcal{I}}$ s.t. $C \in type^{\mathcal{I}}(d')$ i.e. $d' \in C^{\mathcal{I}}$, hence we get that d r-reaches some element that contains C. □

Example 10. *Let $q(x) = \exists y\ s^*(x,y), C_1(y)$ be a reachability query that we would like to answer against the knowledge base in our running example. For convenience let's recall the resulting $\mathbf{L}_{fin}$ and $\mathbf{S}_{fin}$ tables from the Example 4.*

$\mathbf{L}_{fin}$	
$\tau_1 = \{A, B\}$	$\tau_1 = \{A, B\}$
$\tau_2 = \{A, B, C\}$	$\tau_{21} = \{A, B, C, C_1\}$
$\tau_2 = \{A, B, C\}$	$\tau_{22} = \{A, B, C, C_2, \bot\}$
$\tau_3 = \{B\}$	$\tau_3 = \{B\}$
$\tau_4 = \{B, C\}$	$\tau_{41} = \{B, C, C_1\}$
$\tau_4 = \{B, C\}$	$\tau_{42} = \{B, C, C_2, \bot\}$
$\tau_5 = \{C\}$	$\tau_{51} = \{C, C_1\}$
$\tau_5 = \{C\}$	$\tau_{52} = \{C, C_2\}$
$\tau_6 = \{A, C\}$	$\tau_{61} = \{A, C, C_1\}$
$\tau_6 = \{A, C\}$	$\tau_{62} = \{A, C, C_2\}$
$\tau_7 = \{D\}$	$\tau_7 = \{D\}$

$\mathbf{S}_{fin}$		
τ_{52}	(r, C)	τ_{41}
τ_{62}	(r, C)	τ_{41}
τ_{52}	(s, D)	τ_7
τ_{62}	(s, D)	τ_7

After the computation of the Algorithm 4.1 the $reach_{[s,C_1]}$ would contain τ_{21}, τ_{41},t_{51},τ_{52}, τ_{61} and τ_{62}. As can be seen from the table τ_{21}, τ_{41}, τ_{51} and τ_{61} would be contained trivially since they already contain the concept C_1, where as the type τ_{52} and τ_{62} would be added to $reach_{[s,C_1]}$ since all of their (r, C) successors in the table $\mathbf{S}_{fin}$ contain the concept C_1 and we have $r \sqsubseteq^*_{\mathcal{T}} s$.

Similarly as in the case of IQs, in order to test $\mathcal{I} \models q(a)$ for all $\mathcal{I} \in mods(\mathcal{A}, \mathcal{T})$, we consider ABoxes completed with certain information from $reach_{[r,C]}$, using a fresh concept name R_{rC} and asserting the membership of each individual that has a type that is found in the set $reach_{[r,C]}$. Additionally, we require that such ABoxes are a superset of the ABoxes that contain all entailed assertions, i.e. that contain $\mathcal{A}^t$.

Definition 25. *Let t be a $\mathbb{T}$-assignment and $q(x) = \exists y\ r^*(x,y), C(y)$ with $C = A_1 \sqcap \cdots \sqcap A_n$ be an RQ. Then $\mathcal{A}^{t,q}$ is the smallest $\mathcal{A}^t \subseteq \mathcal{A}^{t,q}$ such that:*

(q1) $R_{rC}(a) \in \mathcal{A}^{t,q}$ if $t(a) \in reach_{[r,C]}$.

(q2) $R_{rC}(a) \in \mathcal{A}^{t,q}$ if $R(a,b) \in \mathcal{A}^t$ and $R_{rC}(b) \in \mathcal{A}^{t,q}$.

ABoxes $\mathcal{A}^t$ suffice for answering instance queries, and the assertions added in $\mathcal{A}^{t,q}$ correctly capture the entailment of q in all the models in $mods_t(\mathcal{A}, \mathbb{T})$. Therefore to check if a query q is entailed by $(\mathcal{T}, A)$ we check if it is entailed by each model in $mods(\mathcal{A}, \mathbb{T})$.

Lemma 5. *Let $q = \exists x\ r^*(a, x), C(x)$ be an RQ then $\mathcal{I} \models q(a)$ for all $\mathcal{I} \in mods(\mathcal{A}, \mathbb{T})$ iff $R_{rC}(a) \in \mathcal{A}^{t,q}$ for each $\mathbb{T}$-assignment t.*

Proof. Let $q(a) = \exists x\ r^*(a, x), C(x)$ be an RQ.

($\Rightarrow$) We want to prove if $R_{rC}(a) \in \mathcal{A}^{t,q}$ for each $\mathbb{T}$-assignment t then for each $\mathcal{I} \in mods(\mathcal{A}, \mathbb{T})$ we have that $\mathcal{I} \models q(a)$. Let $\mathcal{I}$ be an arbitrary interpretation in $mods(\mathcal{A}, \mathbb{T})$, by Definition 21 we have that $\mathcal{I}$ is a $(\mathcal{A}, t, \mathbb{T})$-interpretation, hence for each $a \in \mathsf{N_I}$ we have that $type^{\mathcal{I}}(a) = t'(a)$ for some type assignment function t'. By our assumption we have that $R_{rc} \in \mathcal{A}^{t',q}$, there are two cases by the Definition 25 how R_{rC} was included in $\mathcal{A}^{t',q}$:

(c1) $t'(a) \in reach_{[r,C]}$. Since we have that $type(a) = t'(a)$ from the Proposition 1 we get that a r-reaches in $\mathcal{I}$ some element $d \in \Delta^{\mathcal{I}}$ such that $d \in C^{\mathcal{I}}$, therefore we have that $\mathcal{I} \models q(a)$.

(c2) $R(a, b) \in \mathcal{A}^{t'}$ and $R_{rC}(b) \in \mathcal{A}^{t',q}$. Since $R_{rC}(b) \in \mathcal{A}^{t',q}$ by the argument above *(c1)* we get that $\mathcal{I} \models q(b)$. By our assumption $R(a, b) \in \mathcal{A}^{t'}$, from Lemma 4 we have that $\mathcal{I} \models R(a, b)$, then from the Definition 24 of RQs we get that $\mathcal{I} \models q(a)$.

($\Leftarrow$) Next we prove that if $\mathcal{I} \models q(a)$ for all $\mathcal{I} \in mods(\mathcal{A}, \mathbb{T})$ then $R_{rC}(a) \in \mathcal{A}^{t,q}$ for each $\mathbb{T}$-assignment t. We will prove it in its contrapositive form, that is if there exists a $\mathbb{T}$-assignment such that $R_{rC}(a) \notin \mathcal{A}^{t,q}$ then there exists an interpretation $\mathcal{I} \in mods(\mathcal{A}, \mathbb{T})$ such that $\mathcal{I} \not\models q(a)$. Let t be a $\mathbb{T}$-assignment such that $R_{rC}(a) \notin \mathcal{A}^{t,q}$, from the Claim 2 we get that there exists an interpretation $\mathcal{I} \in mods_t(A, \mathbb{T})$. By construction of $\mathcal{I}$ we get that $type^{\mathcal{I}}(a) = t(a)$ for any $\mathcal{I} \in mods_t(\mathcal{A}, \mathbb{T})$. Let's assume that for any such $\mathcal{I} \in mods_t(\mathcal{A}, \mathbb{T})$ we have $\mathcal{I} \models q(a)$. But the by the Definition 24 of RQs we have that either:

- $a \in C^{\mathcal{I}}$ in which case $C \in t(a)$ in which case by the Algorithm we get that $t(a) \in reach_{[r,C]}$ and by the case *(q1)* of $\mathcal{A}^{t,q}$ definition we get that $R_{rC}(a) \in \mathcal{A}^{t,q}$ which contradicts our initial assumption.
- a r-reaches some $d \in \Delta^{\mathcal{I}}$ with $d \in C^{\mathcal{I}}$. Since by assumption this holds for any $\mathcal{I} \in mods_t(\mathcal{A}, \mathbb{T})$ we get that for any possible successor τ_s of $t(a)$ it is the case that $\tau_s \in reach_{[r,C]}$, then by the rules of the Algorithm 4.1 we get that $t(a) \in reach_{[r,C]}$ and by the case *(q1)* of $\mathcal{A}^{t,q}$ definition we get that $R_{rC}(a) \in \mathcal{A}^{t,q}$ which contradicts our initial assumption.

Hence we have proved that there exists an $\mathcal{I} \not\models q(a)$. □

We now provide a *rewriting of OMRQs* into ASP programs. The rewriting includes the facts in $\mathcal{P}_{\mathcal{A}}$, all the rules in program $\mathcal{P}_{\mathbb{T}}$, and additionally the rules of the program $\mathcal{P}_{rq}$ shown in the Figure 4.4.

$$R_{rC}(x) \leftarrow type_{\tau}(x) \qquad \text{for each } \tau \in reach_{[r,C]} \tag{4.16}$$
$$R_{rC}(x) \leftarrow r(x,y), R_{rC}(y) \tag{4.17}$$

Figure 4.4: $\mathcal{P}_q$ program for $q(x) = \exists y\ r^*(x,y), C(y)$

For a given RQ $q(x) = \exists y\ r^*(x,y), C(y)$, the rules (4.16) and (4.17) of the program $\mathcal{P}_{rq}$ shown in the figure 4.4 generate the assertions *(q1) – (q2)* in the definition of $\mathcal{A}^{t,q}$.

The answer sets of $\mathcal{P}_{\mathcal{A}} \cup \mathcal{P}_{\mathbb{T}} \cup \mathcal{P}_{rq}$ coincide with the $\mathcal{A}^{t,q}$. From this and Lemma 5 we get the following theorem:

Theorem 4. *Let $\mathcal{A}$ be an ABox covered by $\mathbb{P}$. Given RQ $q(x) = \exists y\ r^*(x,y), C(y)$, we have $(\mathcal{T}, \mathcal{A}) \models q(a)$ iff $R_{rC}(a) \in M$ for all answer sets M of $\mathcal{P}_{\mathcal{A}} \cup \mathcal{P}_{\mathbb{T}} \cup \mathcal{P}_{rq}$.*

Example 11. *Let $\mathcal{A}, \mathcal{T}, \mathbb{T}$ and $reach_{[s,C_1]}$ be taken from our running example. Programs $\mathcal{P}_{\mathcal{A}}$ and $\mathcal{P}_{\mathbb{T}}$ are identical to the previous case since, whereas for $\mathcal{P}_{rq}$ we get the following rules:*

$$\begin{aligned}
\mathcal{P}_q = \{\ & R_{sC_1}(X) \leftarrow \tau_{21}(X).\\
& R_{sC_1}(X) \leftarrow \tau_{41}(X).\\
& R_{sC_1}(X) \leftarrow \tau_{51}(X).\\
& R_{sC_1}(X) \leftarrow \tau_{52}(X).\\
& R_{sC_1}(X) \leftarrow \tau_{61}(X).\\
& R_{sC_1}(X) \leftarrow \tau_{62}(X).\\
& R_{sC_1}(X) \leftarrow s(X,Y), R_{sC_1}(Y).\ \}
\end{aligned}$$

The ASP program $\mathcal{P}_{\mathbb{T}} \cup \mathcal{P}_q \cup \mathcal{P}_{\mathcal{A}}$ has the same number of answer sets:

$$\begin{aligned}
M_1 = \{&\tau_1(a), \tau_3(b), \tau_{51}(c), A(a), B(a), B(b), C(c), C_1(c),\\
&r(a,b), s(a,b), s(b,c), R_{sC_1}(c)\}.\\
M_2 = \{&\tau_1(a), \tau_3(b), \tau_{52}(c), A(a), B(a), B(b), C(c), C_2(c),\\
&r(a,b), s(a,b), s(b,c), R_{sC_1}(c)\}.\\
M_3 = \{&\tau_1(a), \tau_{41}(b), \tau_{61}(c), A(a), B(a), B(b), C(b), C_1(b), A(c), C(c), C_1(c),\\
&r(a,b), s(a,b), s(b,c), R_{sC_1}(b), R_{sC_1}(c)\}.\\
M_4 = \{&\tau_1(a), \tau_{41}(b), \tau_{62}(c), A(a), B(a), B(b), C(b), C_1(b), A(c), C(c), C_2(c),\\
&r(a,b), s(a,b), s(b,c), R_{sC_1}(b), R_{sC_1}(c)\}.\\
M_5 = \{&\tau_{21}(a), \tau_1(b), \tau_{51}(c), A(a), B(a), C(a), C_1(a), A(b), B(b), C(c), C_1(c),\\
&r(a,b), s(a,b), s(b,c), R_{sC_1}(a), R_{sC_1}(c)\}.
\end{aligned}$$

$$M_6 = \{\tau_{21}(a), \tau_1(b), \tau_{52}(c), A(a), B(a), C(a), C_1(a), A(b), B(b), C(c), C_2(c), r(a,b), s(a,b), s(b,c), R_{sC_1}(a), R_{sC_1}(c)\}.$$
$$M_7 = \{\tau_{21}(a), \tau_{21}(b), \tau_{61}(c), A(a), B(a), C(a), C_1(a), A(b), B(b), C(b), C_1(b), A(c), C(c), C_1(c), r(a,b), s(a,b), s(b,c), R_{sC_1}(a), R_{sC_1}(b), R_{sC_1}(c)\}.$$
$$M_8 = \{\tau_{21}(a), \tau_{21}(b), \tau_{62}(c), A(a), B(a), C(a), C_1(a), A(b), B(b), C(b), C_1(b), A(c), C(c), C_2(c), r(a,b), s(a,b), s(b,c), R_{sC_1}(a), R_{sC_1}(b), R_{sC_1}(c)\}.$$

By observing the above answer sets we see that the only answer to the query $q(x) = \exists y\ s^*(x,y), C_1(y)$ *is* c *since the atom* $R_{sC_1}(c)$ *is found in all the answer sets.*

4.3 Semi-full Conjunctive Queries with Reachability Atoms

In this section, we show how we can obtain an ASP rewriting for an interesting query language that combines conjunctive queries with reachability atoms, subject to some restrictions on the existential quantification of variables. More specifically, in the proposed query language we allow only the second term appearing in some reachability atom $r^*(x,y)$ to be existentially quantified, and disallow joins over existential variables. Regardless of this restriction, the query language offers quite some flexibility by allowing one to express multiple RQs (as defined in the previous section), and join them arbitrarily with other atoms over answer variables. This allows for multiple search conditions about the anonymous part to be expressed in the same query. We call such query language *semi-full conjunctive query with reachability atoms* (s-CRQ).

Definition 26 (Semi-full Conjunctive Query with Reachability Atoms)**.** *A s-CRQ has the following form:*

$$q(\vec{x}) = \exists \vec{y}\ \phi(\vec{x}, \vec{y})$$

where $\phi(\vec{x}, \vec{y})$ *is a conjunction of atoms that take one of the following forms:*

(a1) $r(x,y)$ *where* $x,\ y \in \vec{x}$ *and* $r \in \mathsf{N_R}$*, or*

(a2) $r^*(x,y)$ *where* $x \in \vec{x}$*,* $r \in \overline{\mathsf{N_R}}$*, and if* $y \in \vec{y}$ *then it can occur only in other atoms of the form* $A(y)$*, or*

(a3) $A(x)$ *where* $A \in \mathsf{N_C}$*, and if* $x \notin \vec{x}$*, then it also occurs in some atom of the form* $r^*(y,x)$*.*

As it can be observed from the definition of s-CRQs above, the only variables existentially quantified are the ones occurring inside the reachability atoms or unary atoms sharing the existential variable with some reachability atom, while all other variables are universally

quantified. From *(a3)* it follows that all atoms of the form $A(x)$ where x is existentially quantified should share x with some reachability atom $r^*(x, y)$, and from *(a2)* we get that they share x with exactly one atom of the form $r^*(x, y)$. Based on this observation, we can see s-CRQ as a conjunction of RQs and a universally quantified CQ, sharing the answer variables. Moreover, we distinguish two types of reachability atoms $r^*(x, y)$, the first where variables are universally quantified, and the second where y is existentially quantified.

Given an s-CRQ query q, we define a separation of q into the sets $RQ_e(q)$, $RQ_u(q)$, and $FCQ(q)$, in the following manner:

$$\begin{aligned} RQ_e(q) =& \{\ \exists y\ r^*(x, y), C(y) \mid r^*(x, y) \in q, y \in \vec{y}, \\ & \qquad C(y) \text{ represents the conjunction of unary atoms sharing } y\ \} \\ RQ_u(q) =& \{\ r^*(x, y) \mid r^*(x, y) \in q, y \in \vec{x}\ \} \\ FCQ(q) =& \{\ A(x) \mid A(x) \in q, x \in \vec{x}\ \} \cup \{\ r(x, y) \mid r(x, y) \in q, x, y \in \vec{x}\ \} \end{aligned}$$

The set $RQ_e(q)$ contains the reachability queries with existentially quantified variables as per Definition 24, $RQ_u(q)$ contains the reachability queries without existentially quantified variables which can be seen as universally quantified reachability atoms, whereas $FCQ(q)$ contains all the rest of the atoms in q. Note that we preserve the variable names during this process, s.t. by taking the union these three sets we can obtain the original query.

Next we define an ABox $\mathcal{A}^{t,crq}$ for each $\mathbb{T}$-assignment. These ABoxes are essential for testing entailment of ground s-CRQs, i.e., whether given a tuple of constants $\vec{c}$ and a s-CRQ q, $\mathcal{I} \models q(\vec{c})$ for each $\mathcal{I} \in mods_t(\mathcal{A}, \mathbb{T})$. ABoxes $\mathcal{A}^{t,crq}$ contain each ABox $\mathcal{A}^{t,rq}$ for each RQ with existential variables, moreover we complete them with certain information $R_r(a, b)$ which captures entailments of reachability atoms $r^*(x, y) \in RQ_u(q)$, and R_{rC} which capture entailments of RQs in $RQ_e(q)$.

Definition 27. *Let t be a $\mathbb{T}$-assignment and $q(x) = \exists \vec{y}\ \phi(\vec{x}, \vec{y})$ be an s-CRQ. Let $RQ_e(q)$, $RQ_u(q)$ and $FCQ(q)$ be the respective sets from the separation of q. Then $\mathcal{A}^{t,crq}$ is the smallest $\mathcal{A}^{t,crq}$ such that:*

- $\mathcal{A}^{t,rq} \subseteq \mathcal{A}^{t,crq}$ *for each* $rq \in RQ_e$.
- $R_s(a, a) \in \mathcal{A}^{t,crq}$ *for each* $s^*(x, y) \in RQ_u$ *and* $a \in \mathsf{N_I}(\mathcal{A})$, *s.t.* $s \sqsubseteq^*_{\mathcal{T}} r$.
- $R_s(a, b) \in \mathcal{A}^{t,crq}$ *if* $s(a, b) \in \mathcal{A}$ *s.t.* $s^*(x, y) \in RQ_u$ *and* $s \sqsubseteq^*_{\mathcal{T}} r$.
- $R_s(a, c) \in \mathcal{A}^{t,crq}$ *if* $R_s(a, b)$ *and* $R_s(a, b) \in \mathcal{A}^{t,crq}$.

From the given s-CRQ query q we obtain an universally quantified (full) CQ q_{fcq} in the following manner:

Definition 28. *Let q be a s-CRQ query and $RQ_e(q)$, $RQ_u(q)$, $FCQ(q)$ the respective sets from its separation. We define an universally quantified query q_{fcq} that contains:*

- *all the atoms in* $FCQ(q)$*, and*
- *an atom* $R_{rC}(x)$ *for each* $\exists y\ r^*(x,y), C(y) \in RQ_e(q)$*, and*
- *an atom* $R_r(x,y)$ *for each* $r^*(x,y) \in RQ_u(q)$*.*

Finally for checking if a s-CRQ query $q(\vec{c})$ is entailed in all the models $\mathcal{I} \in mods_t(\mathcal{A}, \mathbb{T})$, we only need to check if $q_{fcq}(\vec{c})$ is entailed by $\mathcal{A}^{t,crq}$.

Lemma 6. *Let* $q(\vec{c}) = \exists \vec{y}\ \phi(\vec{c}, \vec{y})$ *be a s-CRQ query then* $\mathcal{I} \models q(\vec{c})$ *for all* $\mathcal{I} \in mods(\mathcal{A}, \mathbb{T})$ *iff* $q_{fcq}(\vec{c}) \subseteq \mathcal{A}^{t,crq}$ *for each* $\mathbb{T}$*-assignment* t*.*

Proof Sketch. The proof of this lemma follows from Lemma 4 and Lemma 5. By these lemmas we can show that for each atom $\alpha(\vec{c})$ in q_{fcq} that $\alpha(\vec{c}) \in \mathcal{A}^{t,crq}$ iff the corresponding subquery of $\alpha(\vec{c})$ in q is modeled by all the models in $mods(\mathcal{A}, \mathbb{T})$. Then one can show that $q_{fcq}(\vec{c})$ is entailed iff $q(\vec{c})$ is entailed by showing that all atoms of $q_{fcq}(\vec{c})$ are found in $\mathcal{A}^{t,crq}$ iff all their corresponding subqueries in q are entailed. □

In Figure 4.5 an ASP rewriting $\mathcal{P}_{crq}$ of an s-CRQ q is given. Note that we use the plain q_{fcq} query of the given s-CRQ to guide the rewriting. The rules 4.18 and 4.19 encode each RQ subquery present in q, and as such are identical to the rules 4.16 and 4.17 of the program $\mathcal{P}_q$ for RQs shown in Figure 4.4. On the other hand, the rules 4.20 and 4.21 encode the reachability in ABox for each of the atoms in $RQ_u(q)$. Lastly, the rule 4.22 encodes the answers to the query q_{fcq}, by putting all universally quantified atoms in the original query together with the atoms of the form R_rC and R_r that carry the inferences of corresponding reachability queries.

The answer sets of the program $\mathcal{P}_\mathcal{A} \cup \mathcal{P}_\mathbb{T} \cup \mathcal{P}_{crq}$ are in one-to-one correspondence with the $\mathcal{A}^{t,crq}$ ABoxes. By this and Lemma 6, the following theorem holds:

Theorem 5. *Let* $\mathcal{A}$ *be an ABox covered by* $\mathbb{P}$*. Given a s-CRQ* $q(\vec{c}) = \exists \vec{y}\ \phi(\vec{c}, \vec{y})$*, we have* $(\mathcal{T}, \mathcal{A}) \models q(\vec{c})$ *iff* $ans(\vec{c}) \in M$ *for all answer sets* M *of* $\mathcal{P}_\mathcal{A} \cup \mathcal{P}_\mathbb{T} \cup \mathcal{P}_{crq}$*.*

4.3.1 Uplifting the Technique to Conjunctive Queries

Our computed type table $\mathbb{T}$ structure can be used to answer conjunctive queries by adopting the work of [EOv12a]. Moreover, our computation of $\mathbb{T}$ is largely motivated by the technique shown in [EOv12a], which utilizes the concept of *knots* for compiling the knowledge. Knots are trees of depth at most one, where the nodes represent types that coincide with the types we use and branches represent roles. Their technique works in three steps, in the first they define a set of knots $\mathbf{K}$, in the second step they use $\mathbf{K}$ to rewrite the given query into a union of CQs (UCQ), and in the last step they obtain a set of ABox completions $\mathbf{A}$ over which they answer the rewritten query.

While the set of ABox completions $\mathbf{A}$ is similar with the union of all $\mathcal{A}^t$ for each $\mathbb{T}$-assignment function t, the set of knots $\mathbf{K}$ can be obtained easily from $\mathbf{S}$ by generating

For each $\exists y\ r^*(x,y), C(y) \in RQ_e(q)$, add:

$$R_{rC}(x) \leftarrow type_\tau(x). \qquad \text{for each } \tau \in reach_{[r,C]} \tag{4.18}$$

$$R_{rC}(x) \leftarrow r(x,y), R_{rC}(y). \tag{4.19}$$

For each $r^*(x,y) \in RQ_u(q)$, add:

$$R_r(x,y) \leftarrow r(x,y). \tag{4.20}$$

$$R_r(x,y) \leftarrow R_r(x,y), r(x,y). \tag{4.21}$$

Add ASP encoding of q_{fcq}

$$\begin{aligned} ans(x_1,\ldots,x_n) \leftarrow & R_{r_iC_i}(x_i),\cdots, && \text{each } R_{r_iC_i}(x_i) \in RQ_e \\ & ,R_{r_i}(x_i,x_{i+1}),\cdots, && \text{each } R_{r_i}(x_i,x_{i+1}) \in RQ_u \\ & ,A(x_i),\cdots,r(x_j,x_{j+1}),\cdots. && \text{each } A_{x_i}, r(x_j,x_{j+1}) \text{ in } q_{fcq} \end{aligned} \tag{4.22}$$

Figure 4.5: $\mathcal{P}_{crq}$ program for a given s-CRQ $q(\vec{x}) = \exists \vec{y}\ \phi(\vec{x},\vec{y})$

all possible trees of depth one rooted at the parent type τ, and for each (r,B) s.t. $\mathbf{S}(\tau,r,B) \neq \emptyset$ adding an r-branch to one of the good (r,B) successor types in $\mathbf{S}(\tau,r,B)$. Such set of knots $\mathbf{K}$ obtained from our model representation $\mathbb{T}$ represent all relevant knots for answering CQs with the original algorithm presented in [EOv12a] as long as the ABox is covered by the set of profiles $\mathbb{P}$ used for computing $\mathbb{T}$. The key difference between our approach and the knot approach is how the model compilation is obtained. In our approach we do this in a goal oriented way, starting from a set of profiles $\mathbb{P}$ and a TBox, thus computing only relevant representation of models for the ABoxes covered by $\mathbb{P}$ which as we showed in the previous chapter are of tameable size. In contrast the algorithm in [EOv12a] assumes the presence in $\mathbf{K}$ of all possible knots relevant for any query q and ABox $\mathcal{A}$, and as such is best case exponential, hence infeasible for implementation.

While following the knot technique to adopt CQ answering in our case is possible, it is not clear how well would such an algorithm behave in practice, we believe it may need careful engineering for making it viable candidate for implementation.

4.4 Evaluation

We have implemented the algorithms for IQ and RQ answering in the prototype Mod4Q, and show in this section evidence that these algorithms are feasible to implement. We recall here that the algorithms presented in this chapter rely on the computed structure $\mathbb{T}$ for pruning the search space during reasoning, more precisely they consider typically small number of types in $\mathbb{T}$ during evaluation compared to exponentially many in theory. In previous chapter we gave empirical evidence that the number of the profiles covering ontologies with real large ABoxes we tested was typically small. In this section we present

the results obtained for the implemented algorithms for IQ answering and RQ answering as a final evidence that our overall technique for answering queries in expressive DLs fulfills the goals of the thesis.

We recall here that our technique works in two steps, an offline and online step. For a detailed explanation please check the Figure 3.2 in the Subsection 3.3.2 of previous chapter. Here we focus mostly on the online steps, i.e. the time the specific algorithm reasons with $\mathbb{T}$ including getting the encoding of the program $\mathcal{P}_q$ and evaluating it with Clingo.

We ran the implemented algorithms over all the ontologies described in Subsection 3.3.2 of the previous chapter, and report the results divided into two groups:

Ontologies with large ABox: in the first group we show the results obtained for the large ontologies considered such as NPD, MyITS and IMdb, which were of particular relevance due to their large ABox sizes.

Ontologies with complex TBox: in the second group we report the results from the selected ontologies in Oxford Ontology Repository for expressive DLs as explained in Subsection 3.3.2. Although they have small ABoxes, they are interesting due to their complex TBoxes.

Our experiments showed that:

- rewriting into ASP is feasible and the generated ASP programs are simple and perform well when evaluated with Clingo.
- query answering with our ASP rewritings scales to very large ABoxes.

Querying large ontologies with ABoxes The results we obtained testing the IQ and RQ algorithm against the first group of ontologies are in Table 4.1. Here we report also the time for producing $\mathcal{P}_\mathcal{A}$ which includes the extraction of $\mathbb{P}$ from the ABox. We did this since the ontologies in this group have large ABoxes and we noticed that the time for obtaining the ABox encoding played a role in the overall results. However, we note that this is an offline step and can be computed on demand when new assertions are added to the ABox. In addition we report the time for obtaining $\mathcal{P}_\mathbb{T}$ which includes the type table computation of $\mathbb{T}$.

Next we give a more detailed explanation of the results showed in the Table 4.1 for each of the families of queries:

Instance queries: Using the program $(\mathcal{P}_\mathbb{T} \cup \mathcal{P}_\mathcal{A})$ and Clingo, we tested for the instances of all concept names in each ontology. This resulted in 370 (MyITS), 333 (NPD) and 84 (IMDb) IQs. The average answer time over all of them is reported in the table. For comparison, we ran the same instance queries using HermiT [GHM$^+$14]; it took 4.5 hours

Ontology	$\mathcal{P}_{\mathcal{A}}$	$\mathcal{P}_{\mathbb{T}}$	RQs (avg)	IQs (avg)
MyITS50	4.66s	1.21s	2.55s	1.67s
MyITS150	7.12s	1.19s	5.08s	4.53s
MyITS250	10.17s	1.29s	10.21s	8.87s
NPD	89s	2.02s	8.89s	8.47s
IMDb	1034s	2.42s	56.48	55.13s

Table 4.1: Querying ontologies with large ABoxes

for MyITS50, versus 10 minutes accumulated time with our prototype. For all other ontologies, HermiT either timed out after 6 hours, or crashed due to memory exhaustion.

Reachability queries: We generated all RQs paring a role name r and concept name A for which either (a) r occurs in a role assertions in an ABox, and A in a good type matching the profile of an individual in the range of r; or (b) there was an (r, B) entry in the $\mathbf{S}$ with A in its end type. Since there are many possible queries, we decided not to test out the pairs for which the above conditions are not fulfilled, which are trivially empty, or their answers coincide with the instances of A. This procedure resulted in a total of 139 (MyITS), 121 (NPD), and 51 (IMDb) RQs. Answering RQs was on average slightly slower than answering IQs.

Querying ontologies with complex TBoxes We followed the same methodology as the one explained above for obtaining reachability queries and instance queries for the 70 ontologies of the Oxford Ontology Repository for which the computation of type table $\mathbb{T}$ was successful. Considering the large number of concept names in the signature of these ontologies; in the order of tens of thousands, we restricted the number of generated queries up to 100 queries per query family per ontology. We ended up with 5494 IQs and 962 RQs, i.e. not for all ontologies we could generate RQs. That is because some of the ontologies were inconsistent, hence we could not find good successor types for them. We noticed that for all the ontologies for which we could generate IQs and RQs, RQs were slightly slower than IQs, in total the minimum time for answering both IQs and RQs was 4 milliseconds, the maximum was 485 millisecond for IQs and 2.1 seconds for RQs, where as the average time was 59 milliseconds for IQs and 164 milliseconds for RQs.

Although the ontologies in the second group weren't perfect for testing the performance of our IQ and RQ algorithms due to their small sized ABoxes, together with the tests for the ontologies with large ABoxes, they provide evidence that our approach yields a promising candidate for implementation in real world scenarios.

4.5 Discussion and Related Work

Answering ontology mediated queries (OMQs) has been a very active field of research over the last decade. Two main research lines have emerged with a large gap in between. On the one hand, for the so-called *lightweight* Description Logics (DLs), algorithms have been

developed, improved, and implemented in reasoners, from DL-Lite (e.g., [CDL+05, RA10, RKZ13]), to $\mathcal{EL}$ [SMKR14, PMH10], and other *Horn DLs* [EOv+12b, ORS11]. Most works use *query rewriting approaches*, with CQs being the query language of choice. As we have already noted obtaining the rewritten query may be costly, but it is independent from a concrete dataset, which prompts it as suitable for evaluation over any ABox using existing engines for the target language. On the other hand, for expressive DLs containing $\mathcal{ALC}$, most of the research on OMQ answering has had theory-oriented goals, like understanding decidability and worst-case complexity [Lut08]. Many algorithms employ tools that are not amenable to implementation, like *automata* [CDL08, CEO14], or reduction to standard reasoning that cause an exponential blow up [GHS08, GLHS08, OCE08]. Rewritings have been proposed (e.g.,[BtCLW14, AOS16, EOv12a]) but they appear impracticable, and to our knowledge, they have not led to implementation attempts. A rewriting into Datalog for $\mathcal{SHIQ}$ was implemented a decade ago in the KAON2 reasoner, but only for instance queries. A published extension to CQs did not yield a data-independent rewriting, and to our knowledge was never implemented [HMS04]. State-of-the-art reasoners for expressive DLs can handle very large ontologies (e.g., Pellet [SPG+07], HermiT [GHM+14], Konclude [SLG14]), but they usually aim at deciding if some model exists, and as we have discussed, they don't seem very useful for query answering.

From our observation, the usual notion of data independence is too strong and can become an obstacle towards practicable algorithms, hence we have devised a technique that makes use of the 'structure' of data. We believe that this is an important direction, and as we showed in this chapter the structure of data can be utilized to obtain algorithms for different types of queries. Moreover, for some selected query languages, like instance queries and reachability queries we showed that our approach scales well for most of the expressive ontologies we tested against. As a future step we seek to uplift our approach to nested reachability queries and adopt the algorithm of [EOv12a] for full CQ answering with the practicability of the technique in mind.

CHAPTER 5

Practicable Reasoning in Hybrid Languages

ASP and ontology languages like DLs are two of the most important families of formalisms in knowledge representation and reasoning, which have largely orthogonal features due to very different assumptions regarding the *completeness* of information. Thus reasoning techniques and algorithms that are deployed in ASP are significantly different from the ones used in DLs. Combining ASP, which makes the *closed-world assumption* (CWA), with DLs, which make the *open-world assumption* (OWA), into expressive *hybrid languages* that would enjoy the positive features of ASP and DLs has received significant attention in the last decade (see, e.g., [Ros05, Ros06, EIL$^+$08, MR10, KAH11]). However, the development of efficient reasoning algorithms and implementations for reasoning in hybrid languages is lacking. This applies also to the progress on understanding the relationship between different hybrid languages, and their relationship with more standard languages like plain ASP.

In this chapter, we present a new hybrid language called *Clopen Knowledge Bases* (CKBs), which generalizes and improves the prominent r-hybrid language [Ros05]. We also present a technique that builds on our approach developed in the previous chapters to obtain an algorithm that is a viable candidate for implementation in practice. We show that reasoning with hybrid KBs that involve an expressive DL can be reduced to standard ASP reasoning. Implementing such an approach efficiently is very challenging because standard ASP tools were not designed for reasoning about ontologies, while existing efficient DL reasoners have very limited capabilities to reason about rules.

This chapter is organized as follows: we start in Section 5.2 by introducing CKBs and define for them a stable model semantics, inspired by the semantics given by Rosati to r-hybrid KBs. In a nutshell, the major difference between the latter formalism and CKBs is that CKBs allow to use CWA predicates in the ontology. This allows for more

convenient knowledge representation, but also causes technical challenges. Then, in Section 5.3, we provide a general decidability result for checking entailment of ground atoms and consistency testing in CKBs, where the ontology part is expressed in the *guarded negation fragment of FOL (GNFO)* [BCS15]. This is a very expressive fragment that subsumes the more prominent *guarded fragment* of FOL, as well as many expressive DLs. We give a NExpTime$^{\text{2ExpTime}}$ upper bound for inference from GNFO-based CKBs (we note that satisfiability of GNFO formulas is 2ExpTime-hard). Then, in Section 5.4 we study the reasoning in CKBs where the ontology is expressed in the very expressive DL $\mathcal{ALCHOI}$, which extends $\mathcal{ALCHI}$ with *nominals*. We show that the (combined) complexity of reasoning in such CKBs is not higher than in standard (non-ground) ASP. If we assume bounded predicate arities in rules, the basic reasoning problems are ExpTime-complete, which coincides with the complexity of standard problems in plain $\mathcal{ALCHOI}$.

In Section 5.5 we study the relationship between CKBs and other existing hybrid languages with an aim towards obtaining practicable algorithms. To this end, we define a restricted class of *separable* CKBs, and show that they can be transformed in polynomial time into the so-called *dl-programs* [EIL$^+$08]. Separable CKBs still generalize r-hybrid KBs, thus we establish a connection between r-hybrid KBs and dl-programs that is interesting in its own right. The dl-programs resulting from this transformation effectively implement a naive algorithm for reasoning in CKBs. In particular, such a dl-program non-deterministically guesses a (relatively large) set of ground atoms, and then uses an external query (a *dl-atom*) to update the DL ontology that is checked for consistency by an external DL reasoner. However, our experiments with the dlvhex suite (an implementation of dl-programs; see [Red17]) show that this approach is not suitable for a practical implementation of CKBs. We address the above mentioned inefficiency by developing *translations* from separable CKBs into standard ASP programs, thus enabling the reuse of existing ASP solvers. Roughly, the idea is to compile the necessary knowledge about the ontology into a set of ASP rules that are almost identical to the ASP programs shown in Section 4.1. Together with the original rules of the CKB, they form an ASP program whose stable models are in close correspondence with the stable models of the input CKB. We define two translations. The first *data-independent* one establishes a connection to ASP, showing that ASP is as expressive as separable CKBs. The other *data representation-dependent* translation is geared towards implementation, which reduces non-deterministic choices by exploiting the representation of the data via profiles and the type table $\mathbb{T}$ computed as shown in Chapter 3. We have implemented the translation into plain ASP for separable CKBs with φ expressed in $\mathcal{ALCH}$ in our prototype Mod4Q, and compared it with the direct translation of separable CKBs into dl-programs using dlvhex. We tested the ASP translations of four programs with a real world ontology on real world data sets of different sizes and present encouraging results in Section 5.6. Our approach yielded a translation procedure for efficient reasoning over instances that the direct translation utilizing dlvhex could not handle.

5.1 Basic Definitions

We recall here that hybrid languages are composed of two components, the rule and the ontological component. We often refer to the ontological component as the theory φ, and we talk about *logics* which are, in general, sets of theories. Our results in particular are for specific logics that are fragments of standard FOL. We introduce here the notions of (relational) interpretations, as usual in FOL.

Interpretations and models.

Similarly as in the definition of ASP programs, we assume a countably infinite set $\mathsf{N_V}$ of variables, $\mathsf{N_I}$ of constants, and $\mathsf{N_D}$ of predicates. An *interpretation* is a pair $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ that consists of a non-empty set $\Delta^{\mathcal{I}}$ (called *domain*), and a *valuation function* $\cdot^{\mathcal{I}}$ that maps (i) each constant $c \in \mathsf{N_I}$ to an element $c^{\mathcal{I}} \in \Delta^{\mathcal{I}}$, and (ii) each predicate symbol r to a set $r^{\mathcal{I}} \subseteq (\Delta^{\mathcal{I}})^n$, where n is the arity of r.

Furthermore, we assume a countably infinite set $\mathbf{T}$ of *theories*. Each theory $\varphi \in \mathbf{T}$ is associated with a set $\mathit{mods}(\varphi)$ of interpretations. Each $\mathcal{I} \in \mathit{mods}(\varphi)$ is called a *model* of φ. We assume that $\top \in \mathbf{T}$, and we let $\mathit{mods}(\top)$ be the set of all interpretations. A *logic* is simply a set of theories $\mathcal{L} \subseteq \mathbf{T}$. As concrete logics we will consider the DLs $\mathcal{ALCH}$, and $\mathcal{ALCHOI}$ which extends the $\mathcal{ALCHI}$ DL (defined in Section 2.1.2) with nominal concept constructor. The syntax for the nominals is given in the following way: $\{a\}$ where a is an individual, whereas the valuation function $\cdot^{\mathcal{I}}$ for $\mathcal{ALCHI}$ is extended with $\{a\}^{\mathcal{I}} = \{a^{\mathcal{I}}\}$.

Note that we use $\mathcal{I}$ for ordinary (DL) interpretations, and I, J for Herbrand interpretations, and thus in what follows we refrain from explicitly specifying the types of interpretations. An Herbrand interpretation I induces an ordinary interpretation$\tilde{I} = (\Delta^{\tilde{I}}, \cdot^{\tilde{I}})$, where:

(i) $\Delta^{\tilde{I}} = \mathsf{N_I}$, and

(i) $r^{\tilde{I}} = \{\vec{u} \mid r(\vec{u}) = I\}$ for all $r \in \mathsf{N_D}$.

5.2 Clopen Knowledge Bases

We next formally define our hybrid language.

Syntax. A *Clopen Knowledge Base (CKB)* is a triple $\mathcal{H} = (\mathcal{P}, \varphi, \Sigma)$, where $\mathcal{P}$ is a program with a finite set of rules, $\varphi \in \mathbf{T}$ is a theory, and $\Sigma \subseteq \mathsf{N_D}$. The predicate symbols in Σ are called the *open predicates* (in $\mathcal{H}$); while the remaining predicates $\mathsf{N_D} \setminus \Sigma$ are called the *closed predicates* of $\mathcal{H}$. The CKB $\mathcal{H}$ is called *safe* if the following holds for every rule $\rho \in \mathcal{P}$: each variable that appears in ρ also appears in some atom $r(\vec{u}) \in \mathsf{body}^+(\rho)$ with $r \notin \Sigma$. Unless stated otherwise, all considered CKBs are safe.

Semantics. We give the stable model semantics of Clopen inspired by the semantics of r-hybrid KBs. Recall that stable model semantics were first defined by [GL88] through the notion of the reduct (see Chapter 2). We define a variation of this reduct that takes open predicates into account.

Given a program $\mathcal{P}$, an Herbrand interpretation I, and $\Sigma \subseteq \mathsf{N_D}$, the *reduct* $\mathcal{P}^{I,\Sigma}$ *of* $\mathcal{P}$ *w.r.t.* I *and* Σ is the ground positive program obtained from $\mathsf{ground}(\mathcal{P})$ in two steps:

(1) First, delete every rule ρ that contains
   (a) $r(\vec{u}) \in \mathsf{body}^+(r)$ with $r \in \Sigma$ and $r(\vec{u}) \notin I$,
   (b) $r(\vec{u}) \in \mathsf{head}(r)$ with $r \in \Sigma$ and $r(\vec{u}) \in I$, or
   (c) $r(\vec{u}) \in \mathsf{body}^-(r)$ with $r(\vec{u}) \in I$.

(2) In the remaining rules, delete all negated atoms, and all ordinary atoms $r(\vec{u})$ with $r \in \Sigma$.

An Herbrand interpretation I is a *stable model* of a CKB $\mathcal{H} = (\mathcal{P}, \varphi, \Sigma)$ if the following two conditions are satisfied:

- $\{r(\vec{u}) \mid r(\vec{u}) \in I, r \notin \Sigma\}$ is a minimal model of $\mathcal{P}^{I,\Sigma}$, and
- $\tilde{I}$ is model of φ.

Reasoning problems. As usual in hybrid languages (see, e.g., [Ros05]), the basic reasoning task for CKBs is *entailment of ground atoms*. That is, given a CKB $\mathcal{H} = (\mathcal{P}, \varphi, \Sigma)$ and a ground atom $R(\vec{u})$, the problem is to decide whether $R(\vec{u}) \in I$ holds for all stable models I of $\mathcal{H}$. This problem can be reduced to checking the non-existence of a stable model for the CKB $\mathcal{H}' = (\mathcal{P} \cup \{\leftarrow R(\vec{u})\}, \varphi, \Sigma)$. Thus in the rest of the chapter we focus on checking the *stable model existence* for a given CKB. Note that in general a CKB may have infinitely many stable models.

Example 12. *The CKB* $\mathcal{H} = (\mathcal{P}, \varphi, \Sigma)$ *contains information on the local transport network (provided by the city's transport authority and assumed to be complete) and on hotels and relevant locations (extracted form the web and not necessarily complete). We have* $\mathcal{P} = \mathcal{P}_1 \cup \mathcal{P}_2 \cup \mathcal{P}_3$*, where* $\mathcal{P}_1$ *and* $\mathcal{P}_2$ *contain facts. The network, which is depicted by solid lines at the bottom of Figure 5.1, is described in* $\mathcal{P}_1$*. Facts of the form* $\mathsf{RouteTable}(\ell, s, s') \leftarrow$ *store that on the line* ℓ*, station* s *is followed by station* s'*. The constants* t_1 *and* t_2 *represent tram lines, while* ℓ_1 *represents a metro line; we have corresponding facts* $\mathsf{MetroLine}(\ell_1)$*,* $\mathsf{TramLine}(t_1)$*,* $\mathsf{TramLine}(t_2)$*.* $\mathcal{P}_2$ *contains facts related to locations, including the following (for convenience,* CloseTo *is depicted with dotted lines).*

$$\mathsf{CloseTo}(c_1, s_1) \leftarrow \quad \mathsf{Hotel}(h_1) \leftarrow \quad \mathsf{TramConn}(h_1) \leftarrow$$
$$\mathsf{CloseTo}(h_2, s_4) \leftarrow \quad \mathsf{Hotel}(h_2) \leftarrow$$

$$
\begin{aligned}
\mathcal{P}_3 = \{\ & \mathsf{MetroStation}(Y_1) \leftarrow \mathsf{RouteTable}(X, Y_1, Y_2), \mathsf{MetroLine}(X) \\
& \mathsf{TramStation}(Y_2) \leftarrow \mathsf{RouteTable}(X, Y_1, Y_2), \mathsf{TramLine}(X) \\
& \mathsf{ReachOnLine}(X, Y_1, Y_2) \leftarrow \mathsf{RouteTable}(X, Y_1, Y_2) \\
& \mathsf{ReachOnLine}(X, Y_1, Y_3) \leftarrow \mathsf{ReachOnLine}(X, Y_1, Y_2), \mathsf{RouteTable}(X, Y_2, Y_3) \\
& \mathsf{TramOnly}(X) \leftarrow \mathsf{TramConn}(X), \mathit{not}\ \mathsf{MetroConn}(X) \\
& \mathsf{Q}(X) \leftarrow \mathsf{Hotel}(X), \mathsf{CloseTo}(X, Y), \mathsf{ReachOnLine}(Z, Y, Y'), \mathsf{CloseTo}(c_1, Y') \\
& \mathsf{Q}'(X) \leftarrow \mathsf{Q}(X), \mathit{not}\ \mathsf{TramOnly}(X) \quad \}
\end{aligned}
$$

$$
\begin{aligned}
\varphi = \{\ & \forall x.(\mathsf{MetroStation}(x) \vee \mathsf{TramStation}(x) \leftrightarrow \mathsf{Station}(x)), \\
& \forall x.(\mathsf{TramConn}(x) \leftrightarrow \exists y\, \mathsf{CloseTo}(x, y) \wedge \mathsf{TramStation}(y)), \\
& \forall x.(\mathsf{MetroConn}(x) \leftrightarrow \exists y\, \mathsf{CloseTo}(x, y) \wedge \mathsf{MetroStation}(y)), \\
& \forall x.(\mathsf{URailConn}(x) \leftrightarrow \exists y\, \mathsf{CloseTo}(x, y) \wedge \mathsf{Station}(y)) \quad \}
\end{aligned}
$$

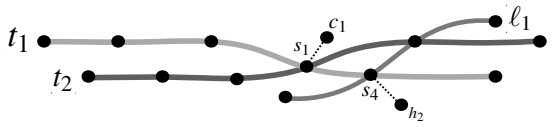


Figure 5.1: Example CKB

*The (self-explanatory) rules in $\mathcal{P}_3$ and the theory φ are shown in Figure 5.1 (*URailConn *stands for urban rail connection). If h is a hotel with direct connection to the point of interest c_1, then $\mathsf{Q}(h)$ holds for it. In this case, it holds for both h_1 and h_2 (note that we do not know which station h_1 is close to). We can use negation as failure to further exclude hotels for which a tram connection is explicitly mentioned, but no metro connection, hence we can assume that it is only reachable by tram, like h_1. For this reason, Q' only holds for h_2. The predicates that describe the network, and those that occur in the heads of the rules in $\mathcal{P}_3$ are closed. The remaining ones are open, i.e. $\Sigma = \{$*Hotel, CloseTo, Station, TramConn, MetroConn, URailConn*$\}$.*

In the spirit of r-hybrid KBs, the FOL theory of a CKB can be seen as a set of integrity *constraints* on the inferences made using the rules of the CKB. Since we are not in classical logic, and in particular because double negation elimination is not valid, "moving" a fact from the program to its theory need not preserve the stable models.

Example 13. *We let* $\Sigma = \{\mathsf{Edge}\}$ *and*

$$
\begin{aligned}
\varphi &= \{\forall xy\ \mathsf{Edge}(x, y) \rightarrow (\mathsf{Node}(x) \wedge \mathsf{Node}(y))\} \\
\mathcal{P} &= \mathsf{Node}(v_1) \leftarrow; \ldots \mathsf{Node}(v_n) \leftarrow; \\
&\quad \mathsf{Reach}(X, X) \leftarrow \mathsf{Node}(X); \\
&\quad \mathsf{Reach}(X, Z) \leftarrow \mathsf{Reach}(X, Y), \mathsf{Edge}(Y, Z), \mathsf{Node}(Z); \}
\end{aligned}
$$

Then these CKBs are not equivalent:

$$\mathcal{H}_1{=}(\mathcal{P}, \varphi \wedge \mathsf{Reach}(v_1, v_2), \Sigma)$$
$$\mathcal{H}_2 {=}(\mathcal{P} \cup \{\mathsf{Reach}(v_1, v_2) \leftarrow\}, \varphi, \Sigma)$$

Indeed, each stable model of $\mathcal{H}_1$ correspond to a directed graph G over $v_1, \ldots, v_n$ such that (v_1, v_2) is included in the reflexive transitive closure of the edge relation in G. In contrast, a stable model of $\mathcal{H}_2$ consists of an arbitrary *graph over $v_1, \ldots, v_n$, together with the reflexive transitive closure of the edge relation augmented with the tuple (v_1, v_2).*

Relationship to ASP. Assume a program $\mathcal{P}$ and an Herbrand interpretation I. We call I a *stable model* of $\mathcal{P}$ if I is a stable model of the CKB $\mathcal{H} = (\mathcal{P}, \top, \emptyset)$. It is not difficult to see that this definition yields precisely the stable models that can alternatively be computed using the standard definition of stable model semantics in ASP. Indeed, the program $\mathcal{P}^{I,\emptyset}$ boils down to the standard Gelfond-Lifschitz reduct $\mathcal{P}^I$ of $\mathcal{P}$ w.r.t. I [GL88](see Chapter 2). Observe that in a CKB $\mathcal{H} = (\mathcal{P}, \varphi, \emptyset)$, the theory φ plays the role of *integrity constraints* on the stable models of the plain program $\mathcal{P}$, i.e. I is a stable model of $\mathcal{H}$ iff I is a stable model of $\mathcal{P}$ such that $\tilde{I} \in \mathit{mods}(\varphi)$.

Relationship to r-hybrid KBs. Our CKBs are a close relative of the r-hybrid KBs of Rosati [Ros05]. The safety restriction here is inspired by the safety condition in r-hybrid KBs, and so is our definition of the semantics via a generalization of the Gelfond-Lifschitz reduct that additionally reduces the program according to the truth value of atoms over open predicates. Intuitively, r-hybrid KBs are a special kind of CKBs in which the rule component can refer to both open and closed predicates, but the theory component can use open predicates only. More formally, an r-hybrid KB is defined as $\mathcal{H}{=}(\varphi, \mathcal{P})$, where φ is a theory in FOL and $\mathcal{P}$ is an ASP program. It corresponds to the CKB $\mathcal{H}'{=}(\mathcal{P}, \varphi, \Sigma)$, where Σ is the set of predicates symbols appearing in φ. One can verify that the stable models of $\mathcal{H}'$ are exactly the so-called *NM-models* of $\mathcal{H}$.

In generic CKBs $\mathcal{H} = (\mathcal{P}, \varphi, \Sigma)$, the set Σ need not contain all the predicate symbols that appear in φ. That is, closed predicates may occur in φ, and the extensions of these predicates in (the relevant) models of φ must be justified by program rules. This feature causes technical challenges, but is very useful for declarative specification of problems: in our approach, predicates under the OWA and the CWA can be used both in the program and in the theory of a hybrid KB (see Example 12 for an illustration).

Active domain predicate. For convenience, we assume the availability of a unary "built-in" predicate adom that, intuitively, stores the constants that appear in a given program. More precisely, for any program $\mathcal{P}$ and each n-ary relation symbol r with $r \neq \mathsf{adom}$ that appears in $\mathcal{P}$, we assume that (i) $\mathcal{P}$ contains the rule $\mathsf{adom}(X_j) \leftarrow r(X_1, \ldots, X_n)$ for every $1 \leq j \leq n$, and (ii) adom is allowed to occur only in bodies of the remaining rules.

5.3 Decidable CKBs

We now turn to identifying useful settings in which the existence of a stable model for a CKB $\mathcal{H} = (\mathcal{P}, \varphi, \Sigma)$ is decidable. This naturally requires φ to belong to a logic $\mathcal{L}$ in which satisfiability is decidable (i.e., the set $\{\varphi \in \mathcal{L} \mid \mathit{mods}(\varphi) \neq \emptyset\}$ should be recursive). However, this alone is not enough, since we will in general be interested in models of φ where a selected set of predicates have a concrete extension that is given as input. We will see that this calls for logics with a rather flexible support for equality reasoning.

Towards providing a quite general decidability result for checking stable model existence in CKBs, we first define a simple program that allows us to freely "guess" the extensions of open predicates of a given CKB $\mathcal{H}$. These extensions are restricted to constants that appear in $\mathcal{H}$.

Definition 29 (Program *Choose*($\mathcal{H}$))**.** *Assume a CKB $\mathcal{H} = (\mathcal{P}, \varphi, \Sigma)$. For every n-ary relation symbol $r \in \Sigma$, let $\overline{r}$ be a fresh n-ary relation symbol that does not appear in $\mathcal{H}$. We let Choose($\mathcal{H}$) be the set that contains*

$$r(Y_1, \ldots, Y_n) \vee \overline{r}(Y_1, \ldots, Y_n) \leftarrow \mathsf{adom}(Y_1), \ldots, \mathsf{adom}(Y_n)$$

for each n-ary relation symbol $r \in \Sigma$ that appears in $\mathcal{P}$.

A stable model I of $\mathcal{P} \cup \mathit{Choose}(\mathcal{H})$ can be seen as a (partially complete) candidate for a stable model of a CKB $\mathcal{H} = (\mathcal{P}, \varphi, \Sigma)$. The following proposition, whose proof relies on the imposed CKB safety requirement, tells us when such an I witnesses the existence of a stable model of $\mathcal{H}$.

Proposition 2. *A CKB $\mathcal{H} = (\mathcal{P}, \varphi, \Sigma)$ has a stable model iff $\mathcal{P} \cup \mathit{Choose}(\mathcal{H})$ has some stable model I for which there exists some $\mathcal{I} \in \mathit{mods}(\varphi)$ with the following properties:*

(C1) $(c_1^{\mathcal{I}}, \ldots, c_n^{\mathcal{I}}) \in r^{\mathcal{I}}$ *for all* $r(c_1, \ldots, c_n) \in I$,

(C2) $(c_1^{\mathcal{I}}, \ldots, c_n^{\mathcal{I}}) \notin r^{\mathcal{I}}$ *for all* $\overline{r}(c_1, \ldots, c_n) \in I$, *and*

(C3) if $(e_1, \ldots, e_n) \in r^{\mathcal{I}}$ *and* $r \notin \Sigma$, *then there exists* $r(c_1, \ldots, c_n) \in I$ *with* $c_1^{\mathcal{I}} = e_1, \ldots, c_n^{\mathcal{I}} = e_n$.

From Proposition 2, we obtain decidability of stable model existence for $\mathcal{H} = (\mathcal{P}, \varphi, \Sigma)$ whenever we can list the stable models of $\mathcal{P} \cup \mathit{Choose}(\mathcal{H})$ and test, for each of them, the existence of a model $\mathcal{I}$ of the theory φ satisfying conditions (C1–C3). Moreover, if the logic $\mathcal{L}$ in question is strong enough to express, for a fixed candidate I, conditions (C1–C3) as part of a theory in $\mathcal{L}$, then decidability of the underlying satisfiability problem suffices. This applies, in particular, to the *guarded negation fragment (GNFO)*, which is among the most expressive FOL fragments for which decidability has been established [BCS15].

We use $\varphi[\vec{x}]$ to indicate that a FOL formula φ has $\vec{x}$ as free variables. The fragment GNFO contains all formulas that can be built using the following grammar:

$$\varphi ::= r(v_1, \ldots, v_n) \mid v = u \mid \exists x\ \varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \alpha \wedge \neg\varphi[\vec{x}],$$

where $u, v, v_1, \ldots, v_n$ are terms, and α is an atom or an equality statement such that all variables of $\vec{x}$ also occur in α. Intuitively, in GNFO a subformula can be negated only if its free variables are "guarded" by an atom or an equality statement. Observe also that a subformula with a single free variable x can always be guarded by an equality statement $x = x$. GNFO is flexible and natural for domain modelling; for instance, the theory φ in Example 12 is in GNFO.

Theorem 6. *Checking the stable model existence in CKBs $\mathcal{H} = (\mathcal{P}, \varphi, \Sigma)$, where φ is in GNFO, is decidable, moreover the problem belongs to the class* NExpTime$^{\text{2ExpTime}}$*, and is 2*ExpTime*-hard.*

Proof. Assume a CKB $\mathcal{H} = (\mathcal{P}, \varphi, \Sigma)$ with φ in GNFO. Let Σ_c be the set of predicates that occur in $\mathcal{P}$ but not in Σ. For every n-ary predicate symbol $r \in \Sigma_c$, assume a tuple $\vec{x}_r = (x_r^1, \ldots, x_r^n)$ of variables. Assume a stable model I of $\mathcal{P} \cup \mathit{Choose}(\mathcal{H})$. For such I, let $\psi(I)$ be the following formula:

$$\psi(I) = \bigwedge_{r(\vec{c}) \in I} r(\vec{c}) \wedge \bigwedge_{\overline{r}(\vec{c}) \in I} \neg r(\vec{c}) \wedge \bigwedge_{r \in \Sigma_c} \forall x_r^1 \ldots \forall x_r^n \Big(r(x_r^1, \ldots, x_r^n) \rightarrow \bigvee_{r(c_1,\ldots,c_n) \in I} \big(\bigwedge_{1 \leq i \leq n} (x_r^i = c_i) \big) \Big)$$

One can check that the formula $\varphi \wedge \psi(I)$ is in GNFO. Note that the three conjuncts mimic the conditions (C1)–(C3); the third one relies on the availability of equality, and is essentially the same formula used in [BBtCP16] for reasoning about *visible* and *invisible* tables in databases. The following holds: $\psi(I)$ is satisfiable iff there exists $\mathcal{I} \in \mathit{mods}(\varphi)$ that satisfies the conditions (C1-C3) of Proposition 2. Overall, this means that $\mathcal{H}$ has a stable model iff $\mathcal{P} \cup \mathit{Choose}(\mathcal{H})$ has some stable model I such that $\varphi \wedge \psi(I)$ is satisfiable. Keeping in mind that satisfiability in GNFO is 2ExpTime-complete, this equivalence yields the NExpTime$^{\text{2ExpTime}}$ upper bound. Indeed, we can decide the existence of a stable model for $\mathcal{H}$ by non-deterministically guessing a candidate stable model I of $\mathcal{P} \cup \mathit{Choose}(\mathcal{H})$, whose size is at most exponential in the size of $\mathcal{H}$, and then checking that (i) I is a minimal model of $\mathcal{P}^{I,\emptyset}$, and (ii) that the formula $\psi(I)$ is satisfiable. The lower bound is carried over trivially from GNFO. □

5.4 CKBs and Description Logics

GNFO is very expressive and thus also computationally very expensive. Moreover the focus of this thesis is on ontologies expressed in description logics, hence in this section we

move on the study of DL-based CKBs, and show that such CKBs are (to a large extent) computationally not more expensive than plain ASP. We show an interesting result for $\mathcal{ALCHOI}$ an expressive DL, in which if we assume bounded predicate arities in the rules, the complexity of basic reasoning problems in CKB coincide with the complexity of standard problems in plain $\mathcal{ALCHOI}$. We note that satisfiability of $\mathcal{ALCHOI}$ TBoxes is ExpTime-complete [BCM+03].

Example 14. *The theory φ in Example 12 can be written in the syntax of $\mathcal{ALCHOI}$ as follows (we use the axiom $C \equiv D$ as a shortcut for the two inclusions $C \sqsubseteq D$, $D \sqsubseteq C$);*

$$\begin{aligned}\mathcal{T} = \{ \quad & \mathsf{MetroStation} \sqcup \mathsf{TramStation} \equiv \mathsf{Station}, \\ & \mathsf{TramConn} \equiv \exists \mathsf{CloseTo}.\mathsf{TramStation}, \\ & \mathsf{MetroConn} \equiv \exists \mathsf{CloseTo}.\mathsf{MetroStation}, \\ & \mathsf{URailConn} \equiv \exists \mathsf{CloseTo}.\mathsf{Station} \quad \}\end{aligned}$$

The following theorem is proven using (well) known complexity results from DLs and ASP, in combination with an encoding of condition (C3) of Proposition 2 by means of nominals, similarly to the encoding of DBoxes in [FIS11].

Theorem 7. *Deciding stable model existence in CKBs $\mathcal{H} = (\mathcal{P}, \mathcal{T}, \Sigma)$, where $\mathcal{T}$ is an $\mathcal{ALCHOI}$ TBox, is* NExpTime$^{\text{NP}}$*-complete. If $\mathcal{P}$ is not disjunctive, the problem is* NExpTime*-complete. The problem is* ExpTime*-complete, if (i) $\mathcal{P}$ is both non-disjunctive and positive, or (ii) the arity of predicate symbols in $\mathcal{P}$ is assumed to be bounded by a constant.*

Proof. Assume a CKB $\mathcal{H} = (\mathcal{P}, \mathcal{T}, \Sigma)$, where $\mathcal{T}$ is an $\mathcal{ALCHOI}$ TBox. Assume a stable model I of $\mathcal{P} \cup \mathit{Choose}(\mathcal{H})$. For any such I, let $\mathsf{TBox}(\mathcal{H}, I)$ be the $\mathcal{ALCHOI}$ TBox that contains the following inclusions (in some axioms below we use $\{d_1, \ldots, d_n\}$ instead of $\{d_1\} \sqcup \ldots \sqcup \{d_n\}$):

- $\{c\} \sqsubseteq A$, for all $A(c) \in I$ with $A \in \mathsf{N_C}$, and $\{c\} \sqsubseteq \neg A$ for all $\overline{A}(c) \in I$ with $A \in \mathsf{N_C}$,
- $\{c\} \sqsubseteq \exists r.\{d\}$ for all $r(c,d) \in I$ with $r \in \mathsf{N_R}$, and $\{c\} \sqsubseteq \forall r.\neg\{d\}$ for all $\overline{r}(c,d) \in I$ with $r \in \mathsf{N_R}$,
- $A \sqsubseteq \bigsqcup_{A(c) \in I} \{c\}$, for all concept names $A \notin \Sigma$,
- $\exists r \sqsubseteq \{d \mid \exists d' : r(d,d') \in I\}$, for all role names $r \notin \Sigma$,
- $\{c\} \sqsubseteq \forall r.\{d \mid \exists e : r(e,d) \in I\}$ for all role names $r \notin \Sigma$ and all constants c that appear in I.

The construction of $\mathsf{TBox}(\mathcal{H}, I)$ is inspired by a similar encoding in [FIS11] where an expressive DL with the so-called *DBoxes* is a translated into a standard DL with nominals.

Due to Proposition 2, and due to the construction of the above TBoxes, $\mathcal{H}$ has a stable model iff $\mathcal{P} \cup \mathit{Choose}(\mathcal{H})$ has a stable model I such that $\mathcal{T} \cup \mathsf{TBox}(\mathcal{H}, I)$ is consistent. In other words, the consistency of $\mathcal{H}$ can be decided by traversing the stable models I of $\mathcal{P} \cup \mathit{Choose}(\mathcal{H})$, for each such I building $\mathcal{T} \cup \mathsf{TBox}(\mathcal{H}, I)$ and checking its satisfiability. We note that $\mathsf{TBox}(\mathcal{H}, I)$ is always of polynomial size in the size of $\mathcal{H}$, and consequently checking the consistency of $\mathsf{TBox}(\mathcal{H}, I)$ is feasible in single exponential time. From this observation, and the complexity of standard ASP under the syntactic restriction mentioned in the theorem, the completeness results follow. □

5.5 Translations and Implementation

We focus here on DL-based CKBs as described in the previous section, and provide translations from such CKBs to other formalisms, in particular to dl-programs and to plain ASP. The translations are given for a large fragment of CKBs, which we call *separable CKBs*, and which in fact generalizes r-hybrid KBs. To define the fragment we need the notion of a *positive occurrence* and a *negative occurrence* of a concept or role name α in a concept C. These notions are defined inductively as follows.

- Each concept name A occurs positively in A.
- Each role name s with $r \sqsubseteq^*_{\mathcal{T}} s$ occurs positively in $\exists r.C$, for any concept C.
- Each role name s with $r \sqsubseteq^*_{\mathcal{T}} s$ occurs negatively in $\forall r.C$, for any concept C.
- If a concept name A occurs positively (resp., negatively) in C, then A occurs positively (resp., negatively) in $C \sqcap D$, $C \sqcup D$, $\forall r.C$ and $\exists r.C$, for any concept D and role r.
- If a concept or role name α occurs positively (resp., negatively) in C, then α occurs negatively (resp., positively) in $\neg C$.

Definition 30 (Separability)**.** *A CKB $\mathcal{H} = (\mathcal{P}, \mathcal{T}, \Sigma)$ is* separable *if the concept $\bigsqcap_{C \sqsubseteq D \in \mathcal{T}} (\neg C \sqcup D)$ does not have a positive occurrence of concept or role name α with $\alpha \notin \Sigma$.*

Example 15. *Take the CKB $\mathcal{H} = (\mathcal{P}, \mathcal{T}, \Sigma)$ with $\mathcal{P} = \{Q(X, Y, Z) \leftarrow t(X, Y), p(Y, Z)\}$, $\mathcal{T} = \{\exists r.(\exists p.A) \sqsubseteq B\}$, and $\Sigma = \{r, A, B\}$. Then $\mathcal{H}$ is separable because p occurs only negatively in $\neg(\exists r.(\exists p.A)) \sqcup B$.*

Intuitively, in a separable CKB $\mathcal{H} = (\mathcal{P}, \mathcal{T}, \Sigma)$ the inclusions in $\mathcal{T}$ can be used to infer the extensions of open predicates from the extensions of closed predicates and other predicates, but these axioms simply cannot assert membership of a domain element (resp., pair of elements) in a closed concept name (resp., role name). More concretely, for separable CKBs one can show a version of Proposition 2 where the condition (C3) is omitted (the rest of the proposition remains the same). The omission of condition (C3)

is a major change: recall that we relied heavily on the equality predicate in GNFO, and on nominals supported in $\mathcal{ALCHOI}$ in order to cope with (C3). We note that separable CKBs capture r-hybrid KBs $\mathcal{H} = (\mathcal{T}, \mathcal{P})$ with $\mathcal{T}$ an $\mathcal{ALCHOI}$ TBox. Such KBs, as mentioned, correspond to CKBs $\mathcal{H} = (\mathcal{P}, \mathcal{T}, \Sigma)$, where Σ is the set of predicates symbols that appear in $\mathcal{T}$, and which trivially satisfy the separability condition. We remark that the pair $(\mathcal{T}, \mathcal{P})$ with $\mathcal{T}, \mathcal{P}$ from Example 15 is not a safe r-hybrid KB, because the variable Z does not appear in a rule atom with a predicate symbol that does not occur in $\mathcal{T}$.

5.5.1 Translation into DL-programs

In this subsection we provide a translation from separable CKBs to dl-programs. First we recall here the syntax and semantics of dl-programs [EIL+08]. Roughly speaking, dl-programs extend plain ASP with *dl-atoms*, which are special atoms that correspond to queries over an external DL KB. For the encoding based on Proposition 2, we need only a relatively small fragment of dl-programs. In particular, here dl-atoms are only allowed to test consistency of DL knowledge bases, and can only use the operators $\uplus$ and $\cup$ to ("temporarily") update it. More formally, a *dl-atom* α is an expression of the form

$$DL[S_1 op_1 R_1, \ldots, S_n op_n R_n; \bot](t), \tag{5.1}$$

where t is a term, $\{op_1, \ldots, op_n\} \subseteq \{\uplus, \cup\}$, and each pair S_i, R_i with $1 \leq i \leq n$ is such that (i) $S_i \in \mathsf{N_C}$, and $R_i \in \mathsf{N_D}$ is unary, or (ii) $S_i \in \mathsf{N_R}$, and $R_i \in \mathsf{N_D}$ is a binary. The notion of *dl-rules* ρ is defined exactly as the notion of ordinary rules, except that here dl-atoms can occur in the place of non-negated ordinary atoms. Each dl-rule ρ must satisfy the next condition: every variable of ρ must appear in a non-negated ordinary atom in the body of ρ. A *dl-program* is a pair $\Pi = (\mathcal{T}, \mathcal{P})$ with $\mathcal{T}$ an $\mathcal{ALCHOI}$ TBox, and $\mathcal{P}$ a set of dl-rules. Here concept and role names that occur in $\mathcal{T}$ are allowed to occur in $\mathcal{P}$ only in dl-atoms. We let $\mathsf{ground}(\mathcal{P})$ be the set of ground dl-rules that can be obtained from the rules in $\mathcal{P}$ by replacing variables with constants from $\mathcal{P}$.

When building TBoxes next, we use $P(t)$, $P(t, v)$, $\neg P(t)$ and $\neg P(t, v)$ as abbreviations for inclusion axioms $\{t\} \sqsubseteq P$, $\{t\} \sqsubseteq \exists P.\{v\}$, $\{t\} \sqsubseteq \neg P$, and $\{t\} \sqsubseteq \forall P.\neg\{v\}$, respectively. For a TBox $\mathcal{T}$, an Herbrand interpretation I, and a ground dl-atom α of form (5.1), we write $I \models_{\mathcal{T}} \alpha$ if the TBox $\mathcal{T} \cup \mathcal{T}_1 \cup \cdots \cup \mathcal{T}_n$ is inconsistent, where each $\mathcal{T}_i$, $1 \leq i \leq n$, is defined as follows:

- $\mathcal{T}_i = \{S_i(t) \mid R_i(t) \in I\}$ if $op_i = \uplus$ and $S_i \in \mathsf{N_C}$,
- $\mathcal{T}_i = \{S_i(t, v) \mid R_i(t, v) \in I\}$ if $op_i = \uplus$ and $S_i \in \mathsf{N_R}$,
- $\mathcal{T}_i = \{\neg S_i(t) \mid R_i(t) \in I\}$ if $op_i = \cup$ and $S_i \in \mathsf{N_C}$, and
- $\mathcal{T}_i = \{\neg S_i(t, v) \mid R_i(t, v) \in I\}$ if $op_i = \cup$ and $S_i \in \mathsf{N_R}$.

Assume a dl-program $\Pi = (\mathcal{T}, \mathcal{P})$ and an Herbrand interpretation I. We let Π^I be the (plain) ground positive program that is obtained from $\mathsf{ground}(\mathcal{P})$ by

- deleting every rule with a dl-atom L such that $I \not\models_{\mathcal{T}} L$,
- deleting every rule with a literal *not* $R(\vec{u})$ with $R(\vec{u}) \in I$,
- deleting all dl-atoms, and all negative literals in the remaining rules.

If I is a minimal model of Π^I, then I is called a *(weak) answer set* of Π.

We can now show how a separable CKB $\mathcal{H}$ can be translated into a dl-program $\Pi_{\mathcal{H}}$ while preserving the existence of a stable model. From a separable CKB $\mathcal{H} = (\mathcal{P}, \mathcal{T}, \Sigma)$ we build a dl-program $\Pi_{\mathcal{H}} = (\mathcal{T}', \mathcal{P}')$ as follows. For every concept name A (resp., role name r) that appears in $\mathcal{T}$, let A' be a fresh concept name (resp., let r' be a fresh role name). Then the TBox $\mathcal{T}'$ is obtained from $\mathcal{T}$ simply by replacing every concept and role name S by S'. For the construction of $\mathcal{P}'$, let $S_1, \ldots, S_n$ be an arbitrary enumeration of the concept and role names that appear both in $\mathcal{P}$ and Σ. Then the set $\mathcal{P}'$ of dl-rules is defined as follows:

$$\mathcal{P}' = \mathcal{P} \cup \mathit{Choose}(\mathcal{H}) \cup \{\leftarrow DL[\lambda; \bot](X)\},$$

where $\lambda = S_1' \uplus S_1, \ldots, S_n' \uplus S_n, S_1' \uplus \overline{S}_1, \ldots, S_n' \uplus \overline{S}_n$.

Intuitively, given a stable model I of $\mathcal{P} \cup \mathit{Choose}(\mathcal{H})$, the expression λ above allows us to check the conditions (C1) and (C2) of Proposition 2 (see the construction of $\mathsf{TBox}(\mathcal{H}, I)$ as used in the proof of Theorem 7). The constraint $\leftarrow DL[\lambda; \bot](X)$ is then used to discard I in case the built TBox is inconsistent. Thus from this encoding we get the following result.

Theorem 8. *A separable CKB $\mathcal{H}$ has a stable model iff the dl-program $\Pi_{\mathcal{H}}$ has an answer set.*

5.5.2 Translation into Plain ASP

We describe here our translations from separable CKBs $\mathcal{H} = (\mathcal{P}, \mathcal{T}, \Sigma)$ into standard ASP. Intuitively, instead of using a richer language than plain ASP to perform the search for $\mathcal{I} \in \mathit{mods}(\varphi)$ with properties (C1) and (C2) described in Proposition 2 (as we did above with dl-programs), we perform reasoning about the TBox of an input KB *during* the translation so that afterwards the TBox can effectively be forgotten. Unlike our translation into dl-programs, this translation is not polynomial and may take single exponential time in the size of the input. However, our experiments show that in practice the latter performs much better than the former. The below translations are inspired by existing translation from expressive DLs into disjunctive Datalog [HMS07, EOv12a, BtCLW14]. In fact, we provide a pair of translations: a generic modular translation that is independent from the concrete facts in the input KB, and a restricted translation that does take into account the representation of the data as defined in Chapter 3 which was implemented in our proof of concept Mod4Q. Recall that Mod4Q in its current state supports $\mathcal{ALCH}$, therefore the translations presented in this section are restricted to $\mathcal{ALCH}$ instead of $\mathcal{ALCHOI}$. However, as already mentioned in Section 3.3.1 accommodating inverses ($\mathcal{ALCHI}$) into Mod4Q is not hard, whereas nominals would require significant changes in

the type table computation algorithm implemented in Mod4Q (presented in Chapter 3), therefore the feasibility of their adoption would need further considerations.

We assume here the $\mathcal{ALCH}$ TBoxes are in the *normal form* as given in Definition 9, and recall it here for convenience:

$$\begin{array}{llll} A_1 \sqcap \ldots \sqcap A_n \sqsubseteq B & A \sqsubseteq B_1 \sqcup \ldots \sqcup B_m & A \sqsubseteq \exists r.B & \text{(I)} \\ \exists r.A \sqsubseteq B & A \sqsubseteq \forall r.B & r \sqsubseteq s & \text{(II)} \end{array}$$

where A, B, A_i, B_i are concept names, $\top$ or $\bot$, and r, s are role names.

Definition 31 (Communication rules $Comm(\mathcal{H})$)**.** *For a separable CKB $\mathcal{H} = (\mathcal{P}, \mathcal{T}, \Sigma)$, let $Comm(\mathcal{H})$ denote the set of the following rules:*

$$\begin{array}{rll} s(X,Y) \leftarrow r(X,Y) & \textit{for each} & r \sqsubseteq s \in \mathcal{T} \\ B(X) \leftarrow r(X,Y), A(Y) & \textit{for each} & \exists r.A \sqsubseteq B \in \mathcal{T} \\ B(Y) \leftarrow A(X), r(X,Y) & \textit{for each} & A \sqsubseteq \forall r.B \in \mathcal{T} \end{array}$$

The program $Comm(\mathcal{H})$ simply contains the direct translation of inclusions listed in (II). To deal with the remaining inclusions (i.e. the ones listed in (I)), we employ *types*.

We recall here our definition of types. Types are simply a set of concept names including $\{\top, \bot\}$. Moreover we say that a type τ is *consistent* w.r.t. a TBox $\mathcal{T}$ if there exists a model $\mathcal{I}$ of $\mathcal{T}$ and an element $e \in \Delta^{\mathcal{I}}$ such that $e \in (\bigsqcap_{C \in \tau} C)^{\mathcal{I}}$. We use $types(\mathcal{T})$ to denote the set of consistent types τ w.r.t. $\mathcal{T}$.

Data-independent translation. Assume a separable CKB $\mathcal{H} = (\mathcal{P}, \mathcal{T}, \Sigma)$. For each $\tau \in types(\mathcal{T})$, let Type_τ be a fresh unary predicate symbol. We let $\mathsf{ASP}(\mathcal{H})$ be the extension of $\mathcal{P} \cup Choose(\mathcal{H}) \cup Comm(\mathcal{H})$ with the following rules:

(i) the rule $\bigvee_{\tau \in types(\mathcal{T})} \mathsf{Type}_\tau(X) \leftarrow \mathsf{adom}(X)$

(ii) for each type $\tau \in types(\mathcal{T})$, the following constraints

$$\begin{array}{rl} A(X) \leftarrow type_\tau(X) & \text{for each } A \in \tau \cap \mathsf{N_C} \\ \leftarrow type_\tau(X), A(X) & \text{for each } A \in \mathsf{N_C} \setminus \tau \end{array}$$

The program $\mathsf{ASP}(\mathcal{H})$ above built from a CKB $\mathcal{H}$ yields a tool to decide consistency of $\mathcal{H}$. In fact, the rules additional to the original program $\mathcal{P}$ depend only on $\mathcal{T}$ and Σ, and thus the translation is data-independent. Note that the set $types(\mathcal{T})$ can be computed in single exponential time in the size of $\mathcal{T}$, and for this a standard DL reasoner can be used. Indeed, a type τ is consistent w.r.t. $\mathcal{T}$ iff $\mathcal{T} \cup \{A(c) \mid A \in \tau\} \cup \{\neg A(c) \mid \in \mathsf{N_C} \setminus \tau\}$ has a model, for a fresh constant c.

Theorem 9. *The CKB $\mathcal{H} = (\mathcal{P}, \mathcal{T}, \Sigma)$ has a stable model iff $\mathsf{ASP}(\mathcal{H})$ has a stable model. In fact, for any set F of facts, $\mathcal{H} = (\mathcal{P} \cup F, \mathcal{T}, \Sigma)$ has a stable model iff $\mathsf{ASP}(\mathcal{H}) \cup F$ has a stable model.*

Data representation-dependent translation. Since $|types(\mathcal{T})|$ is often exponential in the size of $\mathcal{T}$, the program $\mathsf{ASP}(\mathcal{H})$ can be prohibitively large to be used in practice. We next present an optimized way to obtain a desired ASP program, by sacrificing data independence.

Assume a separable CKB $\mathcal{H} = (\mathcal{P}, \mathcal{T}, \Sigma)$, and a set of profiles $\mathbb{P}$ that cover the ABox obtained from the constants in $\mathcal{P}$. We obtain the translation in the following steps:

(1) For each profile $p \in \mathbb{P}$ and each concept name B s.t. $B \notin p$, if B appears in one of the following:

   a) in a non-fact rule of $\mathcal{P}$,

   b) in some $\exists r.A \sqsubseteq B \in \mathcal{T}$ or $A \sqsubseteq \forall r.B \in \mathcal{T}$ such that r appears in a non-fact rule of $\mathcal{P}$

   then add B to $Guess^{\mathcal{T}}(p)$.

(2) Compute the base types for each p with the updated $Guess^{\mathcal{T}}(p)$ from the previous step.

(3) Run the type table compilation algorithm from Chapter 3 over the base types obtained in previous step.

(4) We let $\mathsf{ASP}^{\mathsf{dd}}(\mathcal{H})$ be the extension of $\mathcal{P} \cup Comm(\mathcal{H})$ with the following rules:

   (i) for all roles $r \in \Sigma$ that appear in a non-fact rule in $\mathcal{P}$, the rule $r(X,Y) \vee \overline{r}(X,Y) \leftarrow \mathsf{adom}(X), \mathsf{adom}(Y)$, where $\overline{r}$ is a fresh relation symbol

   (ii) for each constant c of $\mathcal{H}$, the rule $\bigvee_{\tau \in GT_{typetab}(p)} type_\tau(c) \leftarrow$ where p is the profile of c, and $GT_{typetab}(p)$ are the set of good types computed by the type table algorithm in step (2).

   (iii) for each constant c of $\mathcal{H}$ and type $\tau \in \tau \in GT_{\mathbb{T}}(p)$ where p is the profile of c, the following constraints

$$\begin{aligned} A(c) \leftarrow & type_\tau(c) && \text{for each } A \in \tau \cap \mathsf{N_C} \\ \leftarrow & type_\tau(c), A(c) && \text{for each } A \in \mathsf{N_C} \setminus \tau \end{aligned}$$

The translation allows us to decide stable model existence:

Theorem 10. *The CKB $\mathcal{H} = (\mathcal{P}, \mathcal{T}, \Sigma)$ has a stable model iff $\mathsf{ASP}^{\mathsf{dd}}(\mathcal{H})$ has a stable model.*

5.6 Evaluation

We present here some experiments that demonstrate the advantages of translating a separable CKB $\mathcal{H}$ into a plain program $\mathsf{ASP}^{\mathsf{dd}}(\mathcal{H})$. We have evaluated the data representation based translation into ASP, using the type table computation implemented in Mod4Q by extending the set of guesses for each profiles for each of the non-ground rules as given in the steps of the translation in previous section.

We compared the performance of our implementation with the direct encoding to dl-programs, as presented on Section 5.5.1. The latter is implemented in dlvhex, which also uses Clingo as ASP solver. We recall that all the experiments were run on a modest hardware, a PC with Intel Core i7 CPU and 16GB RAM running 64bit Linux-Mint 17.

For benchmarking, we used MyITS data, which were taken from OpenStreetMap[1] data, and transformed into Datalog facts following [EPS+15] as explained in Subsection 3.3.2 of Chapter 3. For convenience we recall here the key features of the MyITS benchmarking data. The data contains facts about 19517 geographical points in the map treated as constants. Concept assertions were extracted from tags in the mapping data, for points of interest like Hotel, Restaurant, Shop, Hospital, MetroStation etc. There are also facts about relations between these points and other constants representing objects of interest such as metro lines, types of cuisine, dishes etc. Among relations interpreted under CWA, i.e. not in Σ, we extracted next, relating pairs of points whose distance is below a certain threshold set in meters. By considering different thresholds, ranging from 50 to 250 meters, we obtained sets of facts of different sizes. Other relations interpreted under CWA that were extracted to describe the Vienna metro network are locatedAlong and nextStation. The former relates a metro station to the corresponding metro line, and the latter relates pairs of consecutive stations on the same line. The extracted relations that also occur in $\mathcal{T}$ include roles like hasCuisine and serves, which relate a Restaurant to a Cuisine or a Dish, respectively. The TBox of our separable CKB was extended from the DL-Lite$_R$ as it appeared in MyITS Project [EKS13] to $\mathcal{ALCH}$, for more details refer to Subsection 3.3.2.

We considered 4 separable CKBs with the same TBox $\mathcal{T}$, but different programs $\mathcal{P}$ given in Figure 5.2. Each program captures the potential information need of a tourist searching for a hotel. Programs $\mathcal{P}_1$–$\mathcal{P}_4$ ask for a reachable Hotel from the main station "*Hauptbahnhof*". Additionally $\mathcal{P}_1$–$\mathcal{P}_3$ ask for Hotels that are next to some LocRestaurant (a concept inferred from the ontology). $\mathcal{P}_4$ asks for Hotels that are in a quiet neighbourhood, by negating the computed relation LoudNeighbourhood. Note that $\mathcal{P}_1$ requires that the station close to the Hotel should be reachable without line changes, while $\mathcal{P}_2$ allows for at most one line change, whereas $\mathcal{P}_3$–$\mathcal{P}_4$ allow for any number of changes as long as a station is reachable (achieved via recursion). Note that the relation nextStation is symmetrical.

For each of the mentioned programs, we included the datasets of different sizes shown in Table 5.1, which have up to roughly a million facts.

[1] https://www.openstreetmap.org

	next50	*next100*	*next150*	*next200*	*next250*
Fact count	145014	263075	479283	743935	1053335
$\mathcal{P}_1$	19.6s	30.1s	44.6s	60.2s	87.6s
$\mathcal{P}_2$	19.6s	31.8s	52.7s	64.0s	95.4s
$\mathcal{P}_3$	19.6s	32.8s	56.1s	64.7s	98.2s
$\mathcal{P}_4$	23.8s	32.9s	49.8s	65.9s	87.3s

Table 5.1: Running times in seconds for programs $\mathcal{P}_1$–$\mathcal{P}_4$ for different *next* relations

Our approach behaved well, as can be seen from the running times shown in Table 5.1. The dl-program encoding for dlvhex did not scale for any of the example programs provided, and failed to return answers because of memory exhaustion even for the smallest dataset shown in Table 5.1. We tried to test it against a smaller yet useful set of facts with approx 13000 Datalog facts, and it still reached the time out of 600s that was set.

5.7 Discussion and Related Work

We have presented CKBs, a powerful generalization of Rosati's r-hybrid, and provided decidability and complexity results for them. As shown in Example 12, the ability to use CWA predicates in the theory of a CKB adds significant power. This power is not readily available even in *hybrid MKNF*, a very rich formalism that captures r-hybrid and $\mathcal{DL}$+log KBs [MR10]. Roughly speaking, to capture CKBs we would need to extend hybrid MKNF to support modal operator **K** inside FOL theories. Another way to see a difference is using *data complexity*. Due to results on DLs with DBoxes (see [FIS11]), satisfiability is already NP-hard in data complexity for CKBs based on basic *DL-Lite* TBoxes in combination with non-disjunctive positive rules. The same setting in hybrid MKNF is tractable.

In the direction of getting an algorithm for evaluating CKB programs, we have provided translations for a rich fragment of CKBs (separable-CKBs); a data-representation based translation into plain ASP, and a direct translation into dl-programs, and implemented both translations. For the direct translation into dl-programs we used dlvhex, whereas for the data-representation based translation into ASP we utilized the computed type table $\mathbb{T}$ for a given set of profiles for effectively guessing the relevant types for each of the constants in the program. We compared both translations using clingo as the ASP solver. From our experiments we observed that the translation following our data representation approach yields an efficient reasoner for evaluating CKBs and the first reasoner for r-hybrid KBs thereof. Moreover, the reasoner allows to solve instances that cannot be handled by a direct translation of r-hybrid into dl-programs, which can be seen as a (slightly optimized) implementation of Rosati's original algorithm for reasoning in r-hybrid KBs [Ros05].

There are few other works on implementing reasoning over combinations of DL ontologies and rules. For expressive (non-Horn) DLs that go beyond the *DL-Lite* and $\mathcal{EL}$ families,

$\mathcal{P}_1$: Find hotels that are close to stations reachable from Hauptbahnhof (main Station) with no metro line changes, and have local restaurants close by.

$$\begin{aligned}\mathsf{reachableWithNoChanges}(X,Y) \leftarrow &\mathsf{locatedAlong}(X,Z), \mathsf{locatedAlong}(Y,Z),\\ &X = \text{“}\mathit{Hauptbahnhof}\text{”}\end{aligned}$$

$$\begin{aligned}\mathsf{q}_1(X) \leftarrow &\mathsf{Hotel}(X), \mathsf{next}(X,Y), \mathsf{reachableWithNoChanges}(Z,Y),\\ &\mathsf{next}(X,V), \mathsf{LocRestaurant}(V)\end{aligned}$$

$\mathcal{P}_2$: Find hotels that are close to stations reachable from Hauptbahnhof (main Station) with up to one metro line change, and have local restaurants close by.

$$\mathsf{reachableWithNoChanges}(X,Y) \leftarrow \mathsf{locatedAlong}(X,Z), \mathsf{locatedAlong}(Y,Z)$$

$$\begin{aligned}\mathsf{reachableWithOneChange}(X,Z) \leftarrow &\mathsf{reachableWithNoChanges}(X,Y), \mathsf{locatedAlong}(X,V),\\ &\mathsf{reachableWithNoChanges}(Y,Z), \mathsf{locatedAlong}(Z,W),\\ &X = \text{“}\mathit{Hauptbahnhof}\text{”}\end{aligned}$$

$$\begin{aligned}\mathsf{q}_2(X) \leftarrow &\mathsf{Hotel}(X), \mathsf{next}(X,Y), \mathsf{reachableWithOneChange}(Z,Y),\\ &\mathsf{next}(X,V), \mathsf{LocRestaurant}(V)\end{aligned}$$

$\mathcal{P}_3$: Find hotels that are close to stations reachable from Hauptbahnhof (main Station) using metro lines and have local restaurants close by.

$$\mathsf{reachable}(X,Y) \leftarrow \mathsf{nextStation}(X,Y), X = \text{“}\mathit{Hauptbahnhof}\text{”}$$

$$\mathsf{reachable}(X,Z) \leftarrow \mathsf{nextStation}(X,Y), \mathsf{reachable}(Y,Z)$$

$$\begin{aligned}\mathsf{q}_3(X) \leftarrow &\mathsf{Hotel}(X), \mathsf{next}(X,Y), \mathsf{reachable}(Z,Y),\\ &\mathsf{next}(X,V), \mathsf{LocRestaurant}(V)\end{aligned}$$

$\mathcal{P}_4$: Find hotels that are close to stations reachable from Hauptbahnhof (main Station) and are in a quiet neighbourhood.

$$\mathsf{reachable}(X,Y) \leftarrow \mathsf{nextStation}(X,Y), X = \text{“}\mathit{Hauptbahnhof}\text{”}$$

$$\mathsf{reachable}(X,Z) \leftarrow \mathsf{nextStation}(X,Y), \mathsf{reachable}(Y,Z)$$

$$\begin{aligned}\mathsf{LoudNeighbourhood}(X) \leftarrow &\mathsf{Hotel}(X), \mathsf{next}(X,Y),\\ &\mathsf{reachable}(Z,Y), \mathsf{next}(X,V),\\ &\mathsf{Club}(V)\end{aligned}$$

$$\begin{aligned}\mathsf{LoudNeighbourhood}(X) \leftarrow &\mathsf{Hotel}(X), \mathsf{next}(X,Y),\\ &\mathsf{reachable}(Z,Y), \mathsf{next}(X,V),\\ &\mathsf{Bar}(V)\end{aligned}$$

$$\begin{aligned}\mathsf{q}_4(X) \leftarrow &\mathsf{Hotel}(X), \mathsf{next}(X,Y), \mathsf{reachable}(Z,Y),\\ &\mathit{not}\ \mathsf{LoudNeighbourhood}(X)\end{aligned}$$

Figure 5.2: Programs evaluated against both encodings.

dl-programs is the richest formalism that has been implemented, in particular in the dlvhex suite. The HermiT system supports reasoning in expressive DLs enriched with positive rules under DL-safety [GHM$^+$14]. For Horn DLs, Heymans et al. showed how dl-programs with external queries over Datalog-rewritable DLs can be translated into Datalog with stable negation [HEX10]. Redl recently presented a generalization of this rewriting approach to external atoms in general HEX-programs [Red17], still its applicability for reasoning with DL ontologies was demonstrated only using the lightweight logic *DL-Lite*. An implementation of reasoning in hybrid MKNF KBs (with lightweight ontologies) under the Well-Founded Semantics is also available [AKS13, IKL13]. The work in [Swi04] shows how reasoning about DL concepts, but not general TBoxes, can be implemented in ASP.

CHAPTER 6

Optimizing Reasoning in Expressive Horn DLs

In this chapter, we tackle the problem of optimizing the ontology mediated query answering for expressive Horn DLs. In contrast to their non-Horn counterparts, query answering in Horn DLs is tractable in data complexity [BO15]. This feature has made them natural candidates for implementation in reasoners that scale in practice. The best-known Horn DL is *DL-Lite* [CDL$^+$05], which is the underlying logic for the OWL2 QL profile and has found the way into the application domain. *DL-Lite* is tailored so that queries over the ontology can be transformed into standard SQL queries that already incorporate all the relevant ontological knowledge and as such can be evaluated over the plain database using standard relational database management systems. Although the low data complexity (AC^0) of *DL-Lite* has made them popular, this comes with a cost on the expressivity side; there are many domains which call for expressive features not expressible with the features of *DL-Lite*. Consider the following example:

Example 16. *In Table 6.1 an ontology for the domain of anti money laundering is shown. The purpose of this ontology is to model a monitoring mechanism that identifies persons and businesses that may be part of money laundering schemes. It achieves this by modeling the interaction between persons* (Person) *and business* (Business) *through their banking accounts* (Account). *Banking accounts may additionally be monitored* (MonitoredAccount) *(axiom (a)), that is if the account is owned by some politically exposed person* (PEP). *Moreover, each account can have at most one owner that is a* Person *(axiom (b)). The interaction between accounts is modeled through the role* exchangedFundsWith *which is stated to be symmetrical (axiom (c)) and transitive (axiom (e)). Axiom (d) states that each* PEP *is also a person* ($Person$). *Lastly, axiom (f) states that the role that represents the ownership is transitive.*

(a) $\mathsf{Account} \sqcap \exists \mathsf{hasOwner}.\mathsf{PEP} \sqsubseteq \mathsf{MonitoredAccount}$	(d) $\mathsf{PEP} \sqsubseteq \mathsf{Person}$
(b) $\mathsf{Account} \sqsubseteq \leqslant 1\, \mathsf{hasOwner}.\mathsf{Person}$	(e) $trans(\mathsf{exchangedFundsWith})$
(c) $\mathsf{exchangedFundsWith} \sqsubseteq \mathsf{exchangedFundsWith}^-$	(f) $trans(\mathsf{hasOwner})$

Table 6.1: An example anti money laundering ontology

The axiom (a) expressed in the example above uses conjunction $\sqcap$ and a *qualified existential restriction* $\exists r.B$ on the left-hand side (LHS). Both constructs are not expressible in the *DL-Lite* family, but they are found in many ontologies [BCS+16], and they are the basis of the OWL 2 EL profile ($\mathcal{EL}$ DL) popular for life science ontologies,[1] e.g., SNOMED CT, NCI, and GENE ontologies. Whereas the axioms (b), (e) and (f) are not expressible in *DL-Lite* nor in $\mathcal{EL}$.

Example 17. *Consider the following ABox coupled with the ontology from the previous example.*

$\mathsf{Account}(a_1)$.	$\mathsf{Account}(a_2)$.	$\mathsf{Account}(a_3)$.
$\mathsf{Person}(p_1)$.	$\mathsf{Person}(p_2)$.	$\mathsf{PEP}(p_3)$.
$\mathsf{Business}(b_1)$.	$\mathsf{hasOwner}(a_1, p_1)$	$\mathsf{hasOwner}(a_2, p_2)$
$\mathsf{hasOwner}(a_3, b_1)$	$\mathsf{hasOwner}(b_1, p_3)$	$\mathsf{exchangedFundsWith}(a_2, a_1)$
$\mathsf{exchangedFundsWith}(a_2, a_3)$		

Table 6.2: An example ABox and query

Then one could obtain the owners of accounts that exchanged funds with a monitored account with the following query $\mathsf{q}(y)$*.*

$$\mathsf{q}(y) \leftarrow \mathsf{Account}(x), \mathsf{hasOwner}(x, y), \mathsf{exchangedFundsWith}(x, z), \mathsf{MonitoredAccount}(z)$$

In Figure 6.1 the ABox completed with all the inferences from the TBox is given as a directed graph. The given facts are marked in grey and the inferences in red. The concept (role) names are shortened to Acc*(*Account*),* MonAcc (MonitoredAccount)*,* Pers (Person)*,* hOwn (hasOwner)*, and* exchangedFundsWith (eFunds)*. From this representation one can easily check that answers for the given query* $\mathsf{q}(y)$ *over the given ABox are* p_1 *and* p_2*.*

[1] https://www.w3.org/TR/owl2-profiles/#OWL_2_EL

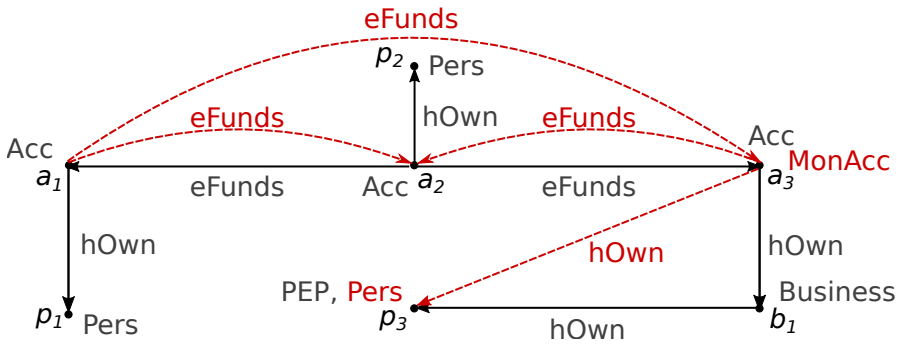


Figure 6.1: The example ABox completed with the inferences from the Ontology

Horn DLs have been advocated for: they can support the features above while remaining computationally manageable. Some advanced reasoning problems, like query emptiness and query inseparability, are more manageable for Horn DLs [BBLW16, BKR$^+$16, BLR$^+$19], and they have proved much more amenable to implementation [EOv$^+$12b, Kaz09, CGK19a]. One of the most popular Horn DLs is Horn-$\mathcal{SHIQ}$, which contains all the constructors of *DL-Lite* and $\mathcal{EL}$ and can be seen as the Horn fragment of OWL Lite. It supports all the axioms listed in the example, including *transitive roles* that, as seen in the example above, allow us to detect interactions through a chain of roles. These additional features make Horn-$\mathcal{SHIQ}$ particularly interesting to study.

Horn-$\mathcal{SHIQ}$ is relatively well understood, and there are existing reasoners for traditional reasoning problems like satisfiability and classification [Kaz09] as well as for *ontology mediated querying (OMQ)*. Unlike *DL-Lite*, it is in general not possible to reduce query answering in the presence of a Horn-$\mathcal{SHIQ}$ ontology to plain SQL query evaluation. Some alternative approaches have been proposed in order to make Horn-$\mathcal{SHIQ}$ feasible on top of existing database technologies. For example, to rewrite (exactly or approximately) an ontology into a weaker DL [LWW07, RPZ10], or to compile some of the extra expressivity into the mappings [BCS$^+$16]. Another possibility is to compile the query and the ontology into a more expressive query language than SQL, like Datalog, as done in the Clipper system [EOv$^+$12b].

Clipper is a query rewriting engine that takes as input an ontology and possibly a set of queries. After a so-called 'saturation' step that uses a *consequence-driven* calculus to add axioms implied by the ontology, it generates a Datalog rewriting of the given queries. Clipper can handle realistic ontologies and queries, despite being a relatively simple prototype. It is among the richest query answering engines for Horn DLs, and has inspired recent adaptations [LMTV19, CDK19]. However, Clipper has stark limitations, and there are many ontologies that it cannot process in reasonable time [CGK19a]. This is largely due to the ABox independence of the saturation step: some axioms that could be omitted for simpler tasks like *classification* [Kaz09], must be inferred by Clipper since they may be made relevant by the assertions in some input ABox.

To overcome this obstacle, we draw from the central idea presented in Chapter 3, which

is to mildly compromise ABox independence, making the saturation step employed in [EOv+12b] aware of the ABox structure, abstracting away concrete assertions. In our running example, suppose we have two additional concepts BusinessAccount and PrivateAccount. If we know that a combination of these will never occur together in the ABox, then we can guide the reasoning not to consider such combination to begin with. Specifically, in Section 6.1, we propose a version of the Clipper saturation that is parametrized by a set of sets of concept names, obtained from the profiles of the individuals as defined in Chapter 3. This set of concept names intuitively represent the concept combinations that may occur in the relevant ABoxes and are used to guide the inference of new axioms. Similarly as in the case of the algorithm for computing the structure $\mathbb{T}$ in Chapter 3, the technique presented here is incremental, for instance, if new ABoxes become relevant, all previous derivations remain valid; new sets of concepts can be added, and the saturation can be re-executed on top of the previous output. A proof of concept of this approach has been implemented on top of existing Clipper reasoner. We will show that despite its simplicity the approach is very effective and gave encouraging results over a range of ontologies. The results of these tests are shown in Section 6.2.

6.1 Restricting Horn-$\mathcal{SHIQ}$ Saturation

The query rewriting algorithm for Horn-$\mathcal{SHIQ}$ described in [EOv+12b] relies on building the canonical model that facilitates CQ answering. It does this in three steps:

- saturate $\mathcal{T}$ under specially tailored inference rules
- close $\mathcal{A}$ under all but existential axioms in $\mathcal{T}$
- employ a chase procedure by extending $\mathcal{A}$ with existential axioms computed in the first step

The first step is of crucial importance, and the subject of the optimization presented in this chapter. In this step the algorithm utilises the calculus shown in Table 6.3 to saturate a given TBox $\mathcal{T}$ to obtain a set of axioms that can be seen as a representation of models that is sufficient for answering queries over any ABox, since from them we can build a universal model of any ABox that is consistent with $\mathcal{T}$. The saturated TBox is then used to rewrite the query in such a way that it can be evaluated without constructing this model.

Horn-$\mathcal{ALCHIQ}^{\sqcap}$ The inference calculus in [EOv+12b] produces a Horn-$\mathcal{ALCHIQ}^{\sqcap}$ TBox. A Horn-$\mathcal{ALCHIQ}^{\sqcap}$ TBox is a Horn-$\mathcal{SHIQ}$ TBox with no transitivity axioms and additionally allowing *role conjunction* as a role constructor; here $r_1 \sqcap r_2$, where r_1, r_2 are roles, and in any interpretation $\mathcal{I}$, $(r_1 \sqcap r_2)^{\mathcal{I}} = r_1^{\mathcal{I}} \cap r_2^{\mathcal{I}}$. We let $(r_1 \sqcap r_2)^{-} = r_1^{-} \sqcap r_2^{-}$ and assume w.l.o.g. that for each role inclusion $r \sqsubseteq s$ of an Horn-$\mathcal{ALCHIQ}^{\sqcap}$ TBox $\mathcal{T}$,

Table 6.3: Inference rules. M, M', N, N', (resp., S, S') are conjunctions of atomic concepts (roles); A, B are atomic concepts.

$$\frac{M \sqsubseteq \exists S.(N \sqcap N') \quad N \sqsubseteq A}{M \sqsubseteq \exists S.(N \sqcap N' \sqcap A)}\ \mathbf{R}^{c}_{\sqsubseteq} \qquad \frac{M \sqsubseteq \exists (S \sqcap S').N \quad S \sqsubseteq r}{M \sqsubseteq \exists (S \sqcap S' \sqcap r).N}\ \mathbf{R}^{r}_{\sqsubseteq} \qquad \frac{M \sqsubseteq \exists S.(N \sqcap \bot)}{M \sqsubseteq \bot}\ \mathbf{R}_{\bot}$$

$$\frac{M \sqsubseteq \exists (S \sqcap r).N \quad A \sqsubseteq \forall r.B}{M \sqcap A \sqsubseteq \exists (S \sqcap r).(N \sqcap B)}\ \mathbf{R}_{\forall} \qquad \frac{M \sqsubseteq \exists (S \sqcap r^{-}).(N \sqcap A) \quad A \sqsubseteq \forall r.B}{M \sqsubseteq B}\ \mathbf{R}^{-}_{\forall}$$

$$\frac{M \sqsubseteq \exists (S \sqcap r).(N \sqcap B) \quad A \sqsubseteq \leqslant 1\, r.B \quad M' \sqsubseteq \exists (S' \sqcap r).(N' \sqcap B)}{M \sqcap M' \sqcap A \sqsubseteq \exists (S \sqcap S' \sqcap r).(N \sqcap N' \sqcap B)}\ \mathbf{R}_{\leq}$$

$$\frac{M \sqsubseteq \exists (S \sqcap r^{-}).(N_1 \sqcap N_2 \sqcap A) \quad A \sqsubseteq \leqslant 1\, r.B \quad N_1 \sqsubseteq \exists (S' \sqcap r).(N' \sqcap B \sqcap C)}{M \sqcap B \sqsubseteq C \quad M \sqcap B \sqsubseteq \exists (S \sqcap (S' \sqcap r)^{-}).(N_1 \sqcap N_2 \sqcap A)}\ \mathbf{R}^{-}_{\leq}$$

$s \in \overline{\mathsf{N_R}}$ and $r^- \sqsubseteq s^- \in \mathcal{T}$. We assume w.l.o.g. that there are no $r \neq s$ in $\overline{\mathsf{N_R}}$ such that $r \sqsubseteq^*_{\mathcal{T}} s$ and $s \sqsubseteq^*_{\mathcal{T}} r$. Abusing notation, we may write $r(a, b) \in \mathcal{A}$ for $r \in \overline{\mathsf{N_R}}$, meaning $r(a, b) \in \mathcal{A}$ if $r \in \mathsf{N_R}$, and $r^-(b, a) \in \mathcal{A}$ otherwise. We recall that we make the unique name assumption (UNA).

Before describing how we can improve saturation using information about the ABox structure, we discuss next in more detail the problem of ABox-independent saturation as described in [EOv$^+$12b].

6.1.1 Bottleneck of ABox-independent Saturation

The calculus of [EOv$^+$12b] shown in Table 6.3 is sound and complete for every possible ABox, and it can be implemented in a relatively simple way. However, it is computationally expensive. It is well-known that the algorithm is (unavoidably) worst-case exponential, but the problem is that this is not just a hypothetical worst-case: an unmanageable combinatorial explosion of inferred axioms may occur for realistic ontologies as well. Roughly, this is because there may be many axioms that are relevant for building the universal model of some ABox, but which are not relevant for the ABoxes we are interested in. We illustrate this through the example below:

Example 18. *Consider the following axioms, that stipulate different kinds of flags for monitored accounts:*

$$\begin{aligned} \mathsf{MonitoredAccount} &\sqsubseteq \exists \mathsf{hasFlag}.\top \\ \mathsf{IndividualAccount} &\sqsubseteq \forall \mathsf{hasFlag}.\mathsf{YellowFlag} \\ \mathsf{SmallBusinessAccount} &\sqsubseteq \forall \mathsf{hasFlag}.\mathsf{RedFlag} \\ \mathsf{BigBusinessAccount} &\sqsubseteq \forall \mathsf{hasFlag}.\mathsf{YellowFlag} \end{aligned}$$

When running the calculus on this ontology, we obtain additionally seven axioms of the form $\mathsf{MonitoredAccount} \sqcap C \sqsubseteq \exists \mathsf{hasFlag}.D$*, where* C *is some conjunction of account types such as* $\mathsf{IndividualAccount} \sqcap \mathsf{SmallBusinessAccount}$*,* $\mathsf{IndividualAccount} \sqcap \mathsf{BigBusinessAccount}$*,*

etc., and D is some conjunction of flag colors. However, if we know that an account will only have one account type, we do not need all these axioms.

Note that these account types could be made disjoint in the ontology by adding a suitable axiom, in which case the algorithm would discard some of the axioms after inferring them. Our approach does not require the account types to be declared disjoint and allows us to save the computation of these axioms in advance if we know that an account is never declared to have two types in the data. We also remark that these kind of patterns are common in ontologies, and in fact they even occur for unrelated classes of objects for which the same 'general role' is used. For example, if we also use the role hasFlag *to flag something other than accounts, such as transactions, the calculus would derive axioms for all combinations of types of transactions and types of accounts. In practice, ontologies often omit some "common sense" disjointness assertions (such as saying that transactions are not accounts, or that accounts are not persons) which are apparent and often irrelevant for modeling the knowledge, but could have a significant impact on this kind of ABox-independent saturation.*

6.1.2 Constraining the Derivation

From the example above, one can see that ABox independence may hinder the performance of the calculus, which generates axioms with combinations of concepts which in practice will never happen, since we are not querying arbitrary ABoxes. We therefore use knowledge about the structure of relevant ABoxes stored in the profiles as done in Chapter 3 to guide the calculus and avoid inferring these axioms in the first place.

Definition 32 (Propagated concepts and activators)**.** *Let a be an individual and $\mathcal{A}$ an ABox. Let $\mathcal{T}$ be a TBox and p a profile. For convenience, we recall the Definition 14 of profiles here.*

$$\mathsf{prof}^{\mathcal{A}}(a) = \{A \mid A(a) \in \mathcal{A}\} \cup \{\exists r \mid r(a,b) \in \mathcal{A}\} \cup \{\exists r^- \mid r(b,a) \in \mathcal{A}\}$$

We define the $\mathcal{T}$-propagating concepts of a profile *p as follows.*

$$prop(p, \mathcal{T}) = \{B | A \sqsubseteq \forall s.B \in \mathcal{T}, r \sqsubseteq^*_{\mathcal{T}} s, \exists r^- \in p\}$$

An activator *α is a set of concept names. We say that a set Λ of activators covers an ABox $\mathcal{A}$ w.r.t. a TBox $\mathcal{T}$ if for each individual a occurring in $\mathcal{A}$, there is some activator $\alpha \in \Lambda$ such that $\mathsf{prof}^{\mathcal{A}}(a)|_{\mathsf{N_C}} \cup prop(\mathsf{prof}^{\mathcal{A}}(a), \mathcal{T}) \subseteq \alpha$, where $\mathsf{prof}^{\mathcal{A}}(a)|_{\mathsf{N_C}}$ denotes the restriction of the profile of a to concept names only.*

Note that in covering sets of activators, we require that for each individual there is an activator that contains not only its type, but all propagating concepts to be included for each individual. We can see this as a way to over-approximate its actual type in the universal model.

Table 6.4: Optimized inference calculus $\nabla(\mathcal{T}, \Lambda)$. M, M', N, N', (resp., S, S') are conjunctions of concept names (roles); A, B are concept names. Λ is the set of activators and α, α' are activators in Λ. The calculus is obtained from [EOv$^+$12b] by adding the side conditions and the rules Λ^*, Λ^+, Λ^-.

$$(\mathbf{R}^{c}_{\sqsubseteq}) \quad \frac{M \sqsubseteq \exists S.(N \sqcap N') \quad N \sqsubseteq A}{M \sqsubseteq \exists S.(N \sqcap N' \sqcap A)} \quad : M \subseteq \alpha, \alpha \in \Lambda$$

$$(\mathbf{R}^{r}_{\sqsubseteq}) \quad \frac{M \sqsubseteq \exists (S \sqcap S').N \quad S \sqsubseteq r}{M \sqsubseteq \exists (S \sqcap S' \sqcap r).N} \quad : M \subseteq \alpha, \alpha \in \Lambda$$

$$(\mathbf{R}_{\bot}) \quad \frac{M \sqsubseteq \exists S.(N \sqcap \bot)}{M \sqsubseteq \bot} \quad : M \subseteq \alpha, \alpha \in \Lambda$$

$$(\mathbf{R}_{\forall}) \quad \frac{M \sqsubseteq \exists (S \sqcap r).N \quad A \sqsubseteq \forall r.B}{M \sqcap A \sqsubseteq \exists (S \sqcap r).(N \sqcap B)} \quad : M \cup A \subseteq \alpha, \alpha \in \Lambda$$

$$(\mathbf{R}^{-}_{\forall}) \quad \frac{M \sqsubseteq \exists (S \sqcap r^-).(N \sqcap A) \quad A \sqsubseteq \forall r.B}{M \sqsubseteq B} \quad : M \subseteq \alpha, \alpha \in \Lambda$$

$$(\mathbf{R}_{\leq}) \quad \frac{\begin{array}{l} M \sqsubseteq \exists (S \sqcap r).(N \sqcap B) \quad A \sqsubseteq \leqslant 1\, r.B \\ M' \sqsubseteq \exists (S' \sqcap r).(N' \sqcap B) \end{array}}{M \sqcap M' \sqcap A \sqsubseteq \exists (S \sqcap S' \sqcap r).(N \sqcap N' \sqcap B)} \quad : M \cup M' \cup A \subseteq \alpha, \alpha \in \Lambda$$

$$(\mathbf{R}^{-}_{\leq}) \quad \frac{\begin{array}{l} M \sqsubseteq \exists (S \sqcap r^-).(N_1 \sqcap N_2 \sqcap A) \quad A \sqsubseteq \leqslant 1\, r.B \\ N_1 \sqsubseteq \exists (S' \sqcap r).(N' \sqcap B \sqcap C) \end{array}}{M \sqcap B \sqsubseteq C \quad M \sqcap B \sqsubseteq \exists (S \sqcap (S' \sqcap r)^-).(N_1 \sqcap N_2 \sqcap A)} \quad : M \cup B \subseteq \alpha, \alpha \in \Lambda$$

$$(\Lambda^*) \quad \frac{M \sqsubseteq B \quad M \subseteq \alpha}{\Lambda = \Lambda \cup \{\alpha \cup \{B\}\}} \qquad (\Lambda^+) \quad \frac{M \sqsubseteq \exists R.N}{\Lambda = \Lambda \cup \{N\}} \qquad (\Lambda^-) \quad \frac{\alpha' \subseteq \alpha}{\Lambda = \Lambda \setminus \alpha'}$$

In Table 6.4 we present the optimized version of the calculus, which takes as input and maintains a set of activators. Each rule has a side condition that checks if the LHS of the axiom we want to derive is contained in an activator present in the maintained set of activators. There are three additional rules Λ^*, Λ^+ and Λ^- not present in [EOv$^+$12b]. The rule Λ^* is used to close the maintained activators under axioms of the form $(NF1)$, while Λ^+ is used to create fresh activators for inferred axioms of the form $(NF2)$, and Λ^- drops redundant activators.

For a TBox $\mathcal{T}$, we can always take a singleton set of activators that contains exactly the set of all concept names occurring in $\mathcal{T}$, and it will cover any ABox over the signature of $\mathcal{T}$. However, this way our optimised calculus is reduced to the original one, since the side condition will not play any role, hence we would derive the same set of axioms. For a concrete ABox we can be more accurate, and take as set of activators precisely the set of all $\alpha_a = \mathsf{prof}^{\mathcal{A}}(a)|_{\mathsf{N_C}} \cup \mathit{prop}(\mathsf{prof}^{\mathcal{A}}(a), \mathcal{T})$ where a is an individual in $\mathcal{A}$. In fact, we use such activators sets in our experiments in Section 6.2.

We define the result of applying the calculus to the TBox. The handling of transitivity here is standard: we drop transitivity axioms from the input TBox, and instead add

axioms of the form (NF4) that ensure the effect of transitivity is accounted for during saturation. The result of saturation, with standard chase techniques, is used to build so-called *premodels* that become models once the extensions of non-simple roles are updated to satisfy transitivity axioms. This is standard and not central to our results, so we do not discuss it in depth. Our optimization of the calculus, of course, does not affect the handling of transitivity during query rewriting from [EOv$^+$12b].

Definition 33. *Given an Horn-$\mathcal{SHIQ}$ ontology, let $\mathcal{T}^*$ be the result of dropping all transitivity axioms trans(r) from $\mathcal{T}$, and adding, for every $A \sqsubseteq \forall s.B \in \mathcal{T}$ and every transitive role r with $r \sqsubseteq^*_{\mathcal{T}} s$, the axioms $A \sqsubseteq \forall r.B^r$, $B^r \sqsubseteq \forall r.B^r$ and $B^r \sqsubseteq B$, where B^r is a fresh concept name.*

We denote by $\nabla(\mathcal{T}, \Lambda)$ the result of saturating $\mathcal{T}^*$ with the rules in Table 6.4 and set of initial activators Λ.

Formally $\nabla(\mathcal{T}, \Lambda)$ is a pair of a TBox and a set of activators, but we sometimes abuse notation and use $\nabla(\mathcal{T}, \Lambda)$ to denote the TBox alone.

Similarly as in [EOv$^+$12b], the saturated set of axioms contains all inferences from the ontology that are relevant for reasoning about any covered ABox; not only for checking consistency, but also for other problems like query rewriting.

Definition 34. *For an ABox $\mathcal{A}$, we denote by $\mathcal{A}_c$ the result of closing $\mathcal{A}$ under the following rules:*

- $A_1 \sqcap \ldots \sqcap A_n \sqsubseteq B \in \nabla(\mathcal{T}, \Lambda)$ *and* $\{A_1(a), \ldots A_n(a)\} \in \mathcal{A}$, *then* $B(a) \in \mathcal{A}$;
- $A \sqsubseteq \forall r.B \in \nabla(\mathcal{T}, \Lambda)$ *and* $r(a, b) \in \mathcal{A}, A(a) \in \mathcal{A}$, *then* $B(b) \in \mathcal{A}$;
- $r \sqsubseteq s \in \mathcal{T}$ *and* $r(a, b) \in \mathcal{A}$, *then* $s(a, b) \in \mathcal{A}$;
- $A \sqsubseteq \leqslant 1\, r B \in \mathcal{T}$ *and* $A(a), r(a, b), r(a, c), B(b), B(c)$, *then* $\bot(a) \in \mathcal{A}$
- $A_1 \sqcap \ldots \sqcap A_n \sqsubseteq \exists(r_1 \sqcap \ldots \sqcap r_m).(B_1 \sqcap \ldots \sqcap B_k)$, $A \sqsubseteq \leqslant 1\, r.B \in \nabla(\mathcal{T}, \Lambda)$ *such that for some i, j we have $r{=}r_i$,$B{=}B_j$ and $A(a), r(a, b) \in \mathcal{A}$, then* $\{B_1(b), \ldots, B_k(b), r_1(a, b), \ldots, r_k(a, b)\} \subseteq \mathcal{A}$.

We call $\mathcal{A}^{\nabla(\mathcal{T},\Lambda)}$ contradiction-free *if there are no assertions of the form $\bot(a)$.*

To test if a given ABox covered by Λ is consistent with $\mathcal{T}$, it is enough to check $\mathcal{A}^c$ for contradiction-freeness.

However, we do not want to only test consistency. Our motivation is OMQ, and we want support for instance and conjunctive queries for different ABoxes. We thus provide the standard guarantee one would expect in this setting: from the computed axioms and a consistent ABox, we can build a *universal model*.

Definition 35 ($\mathcal{T}$-chase, universal model)**.** *Let $\mathcal{T}$ be a Horn-$\mathcal{SHIQ}$ TBox and $\mathcal{I}$ an interpretation. We say that an axiom of the form $M \sqsubseteq \exists S.N$ is* applicable at $e \in \Delta^{\mathcal{I}}$ *if*

(a) $e \in M^{\mathcal{I}}$,
(b) there is no $e' \in \Delta^{\mathcal{I}}$ with $(e, e') \in S^{\mathcal{I}}$ and $e' \in N^{\mathcal{I}}$,
(c) there is no axiom $M' \sqsubseteq \exists S'.N' \in \mathcal{T}$ such that $e \in (M')^{\mathcal{I}}$, $S \subseteq S'$, $N \subseteq N'$, and $S \subset S'$ or $N \subset N'$.

If $M \sqsubseteq \exists S.N$ is applicable at $e \in \Delta^{\mathcal{I}}$, then we obtain an interpretation $\mathcal{I}'$ by applying $M \sqsubseteq \exists S.N$ in $\mathcal{I}$ as follows:

- $\Delta^{\mathcal{I}'} = \Delta^{\mathcal{I}} \cup \{e'\}$ with e' a new element not present in $\Delta^{\mathcal{I}}$,
- for each $A \in \mathsf{N_C}$, we have $A^{\mathcal{I}'} = A^{\mathcal{I}} \cup \{e'\}$ if $A \in N$, and $A^{\mathcal{I}'} = A^{\mathcal{I}}$ otherwise.
- for each $r \in \mathsf{N_R}$, we have $r^{\mathcal{I}'} = r^{\mathcal{I}} \cup \{(e, e')\}$ if $r \in S$, $r^{\mathcal{I}'} = r^{\mathcal{I}} \cup \{(e', e)\}$ if $r^- \in S$, $r^{\mathcal{I}'} = r^{\mathcal{I}}$ otherwise.

For a contradiction-free ABox $\mathcal{A}$, we let $\mathcal{I}_{\mathcal{A}}$ denote the interpretation whose domain are the individuals in $\mathcal{A}$, and that has $A^{\mathcal{I}_{\mathcal{A}}} = \{a \mid A(a) \in \mathcal{A}\}$ for all $A \in \mathsf{N_C}$, and $r^{\mathcal{I}_{\mathcal{A}}} = \{(a, b) \mid r(a, b) \in \mathcal{A}\}$ for all $r \in \mathsf{N_R}$.

The $\mathcal{T}$-chase of a contradiction-free ABox $\mathcal{A}$ is the possibly infinite interpretation obtained from $\mathcal{I}_{\mathcal{A}}$ by fairly applying the existential axioms in $\mathcal{T}$ (that is, where every applicable axiom is eventually applied).

We denote by $\mathcal{I}_c$ the interpretation obtained as follows:

- *let $\mathcal{J}$ denote the $\mathcal{T}$-chase of $\mathcal{A}_c$, then*
- *$\Delta^{\mathcal{I}_c} = \Delta^{\mathcal{J}}$, and*
- *$A^{\mathcal{I}_c} = \mathcal{A}^{\mathcal{J}}$ for every $A \in \mathsf{N_C}$, and*
- *for every $r \in \overline{\mathsf{N_R}}$ $\quad r^{\mathcal{I}_c} = \bigcup_{s \sqsubseteq^*_{\mathcal{T}} r} s^{\mathcal{J}}_+$*

  where $s^{\mathcal{J}}_+$ is the transitive closure of $s^{\mathcal{J}}$ if $trans(s) \in \mathcal{T}$, and $s^{\mathcal{J}}_+ = s^{\mathcal{J}}$ otherwise.

Claim 3. *Let $\mathcal{T}$ be a Horn-$\mathcal{SHIQ}$ TBox, Λ a set of activators. Let $\mathcal{T}'$, and Λ' be the resulting TBox and the set of activators obtained from application of inference calculus $\nabla(\mathcal{T}, \Lambda)$. We say that an axioms $\delta \in \mathcal{T}'$ is introduced by $\nabla(\mathcal{T}, \Lambda)$ if $\delta \notin \mathcal{T}$. Then, we claim that for each introduced axiom of the form $M \sqsubseteq \exists S.N$ there exists a pair of activators $\alpha, \alpha' \in \Lambda'$ s.t. $M \subseteq \alpha$ and $N \subseteq \alpha'$.*

Proof. Let $\delta : M \sqsubseteq \exists S.N$ be an arbitrary introduced axiom by $\nabla(\mathcal{T}, \Lambda)$, and let $\mathcal{T}_1, \ldots, \mathcal{T}_n$ $(\Lambda_1, \ldots, \Lambda_n)$ be the list of TBoxes (sets of Activators), where $\mathcal{T}_i$ was obtained from $\mathcal{T}_{i-1}$ by applying one of the rules of the calculus during the course of saturation, and $\mathcal{T}_1 = \mathcal{T}$, $\mathcal{T}_n = \mathcal{T}'$.
Then δ was introduced at some i-th computation step, by applying one of the following rules: $\mathbf{R}^c_{\sqsubseteq}$, $\mathbf{R}^r_{\sqsubseteq}$, $\mathbf{R}_{\forall}$, $\mathbf{R}^-_{\forall}$, $\mathbf{R}_{\leq}$, $\mathbf{R}^-_{\leq}$, all of which have as a precondition that there should

exist an activator $\alpha \in \Lambda_{i-1}$ s.t $M \subseteq \alpha$.
On the other hand since the calculus is exhaustive and any rule applicable is applied at some time point, there is an j-th computation step in which the rule Λ^+ is applied to the axiom $M \sqsubseteq \exists S.N$ resulting in the addition of an activator $\alpha' = N$ to Λ_j.
Finally, since an activator can be removed in subsequent k-th computation step by the rule Λ^- only in case there exists some other activator $\beta \in \Lambda_{k-1}$ s.t. $\alpha \subseteq \beta$, then we know that there exists a pair of activators α,α' in Λ that contain the sets M and N. □

Next we show that $\mathcal{I}_c$ is indeed a universal model. The following theorem is analogous to Proposition 2 in [EOv$^+$12b]. By guaranteeing that we can build a universal model of any ABox that is consistent with $\mathcal{T}$, we can use $\nabla(\mathcal{T}, \Lambda)$ as a representation of models that is sufficient for query answering. As in the work of Eiter et al., the finite $\nabla(\mathcal{T}, \Lambda)$ allows us to rewrite the query in such a way that it can be evaluated over a small and easy to compute part of the possibly infinite represented universal model, please refer to [EOv$^+$12b] for details.

Theorem 11. *Let $(\mathcal{T}, \mathcal{A})$ be a Horn-$\mathcal{SHIQ}$ KB, and Λ a set of activators, such that Λ covers $\mathcal{A}$ w.r.t. $\mathcal{T}$. The following hold:*

(a) $\mathcal{A}_c$ is contradiction-free iff $(\mathcal{T}, \mathcal{A})$ is consistent, and

(b) if $(\mathcal{T}, \mathcal{A})$ is consistent then $\mathcal{I}_c \models (\mathcal{T}, \mathcal{A})$, and

(c) $\mathcal{I}_c$ can be homomorphically embedded into any model of $(\mathcal{T}, \mathcal{A})$.

Proof. This proof follows the structure of a very similar proof presented in [EOv$^+$12b] for the original calculus.

First we show (a) $\Leftarrow$, i.e. if $(\mathcal{T}, \mathcal{A})$ is consistent, then $\mathcal{A}_c$ is contradiction free.

Observe that for any axiom $\alpha \in \nabla(\Lambda, \mathcal{T})$ we have that $\mathcal{T} \models \alpha$, i.e. α is a logical consequence of $\mathcal{T}$. Suppose that $(\mathcal{T}, \mathcal{A})$ is consistent and $\mathcal{I} \models (\mathcal{T}, \mathcal{A})$, where as $\mathcal{A}_c$ is not contradiction-free, i.e. there exists some a s.t. $\bot(a) \in \mathcal{A}_c$. Since we make use of UNA, we can assume that for each $a \in \mathsf{N_I}$, $a^{\mathcal{I}} = a$. Now, such concept membership $\bot(a) \in \mathcal{A}_c$ can come from three cases:

- $\bot(a) \in \mathcal{A}$, in which case we get that $(\mathcal{T}, \mathcal{A})$ is inconsistent. Contradiction.
- $\bot(a) \in \mathcal{A}_c$ is derived by the first rule in Definition 34, where axiom is of the form $A_1 \sqcap \ldots \sqcap A_n \sqsubseteq \bot$ and $\{A_1, \ldots, A_n\} \subseteq \{A | A(a) \in \mathcal{A}_c \text{ and } A \in \mathsf{N_C}\}$. We distinguish two cases:
  - $A_1 \sqcap \ldots \sqcap A_n \sqsubseteq \bot \in \mathcal{T}$, in which case $(\mathcal{T}, \mathcal{A})$ is inconsistent. Contradiction.
  - $A_1 \sqcap \ldots \sqcap A_n \sqsubseteq \bot \in \nabla(\mathcal{T}, \Lambda)$. By our observation we have that $\mathcal{T} \models A_1 \sqcap \ldots \sqcap A_n \sqsubseteq \bot$, from where we get that $(\mathcal{T}, \mathcal{A})$ is inconsistent. Contradiction.

- $\bot(a) \in \mathcal{A}_c$ is derived by the fourth rule in Definition 34, where axiom is of the form $A \sqsubseteq \leqslant 1\, r B \in \mathcal{T}$ and $A(a), r(a,b), r(a,c), B(b), B(c)$, in which case we get that $(\mathcal{T}, \mathcal{A})$ is inconsistent. Contradiction.

Next we show (b). Assume $(\mathcal{T}, \mathcal{A})$ is consistent. Then there exists an interpretation $\mathcal{I} \models (\mathcal{T}, \mathcal{A})$, from above we get that $\mathcal{A}_c$ is contradiction free. Then according to Definition 35 it's $\nabla(\mathcal{T}, \Lambda)$-chase is defined, and $\mathcal{I}_c$, $\mathcal{I}_{\mathcal{A}_c}$ denote the interpretations of the $\nabla(\mathcal{T}, \Lambda)$-chase and $\mathcal{A}_c$.

To show that $\mathcal{I}_c \models (\mathcal{T}, \mathcal{A})$, we need to define some book keeping in the course of building $\mathcal{I}_c$. By construction we have $\Delta^{\mathcal{I}_{\mathcal{A}_c}}$ is constrained to the individuals in the ABox, and we add a suffix to each fresh individual introduced by the chase procedure in the following way, if d is a successor of e (according to Definition 35) then $d = e \cdot n$, where n is an integer. To show that $\mathcal{I}_c \models (\mathcal{T}, \mathcal{A})$ we first prove $\mathcal{I}_c \models \mathcal{A}$ then $\mathcal{I}_c \models \mathcal{T}$.

Showing that $\mathcal{I}_c \models \mathcal{A}$ is trivial. We know that $\mathcal{I}_c$ is an extension of $\mathcal{I}_{\mathcal{A}_c}$, and $\mathcal{A} \subseteq \mathcal{A}_c$ from where we get that $\mathcal{I}_{\mathcal{A}_c}$ is a model of $\mathcal{A}$ as well.

In order to show that $\mathcal{I}_c \models \mathcal{T}$, we show a stronger result that $\mathcal{I}_c \models \nabla(\mathcal{T}, \Lambda)$, by arguing that $\mathcal{I}_c$ satisfies each axiom found in $\nabla(\mathcal{T}, \Lambda)$, which can take one of the following forms:

($M \sqsubseteq B \in \nabla(\mathcal{T}, \Lambda)$) Note that $\mathcal{T} \subseteq \nabla(\mathcal{T}, \Lambda)$. Assume an arbitrary domain element $e \in M^{\mathcal{I}_c}$. We have two cases:

(i) $e \in \mathsf{N_I}$, in this case since $\mathcal{I}_{\mathcal{A}_c} \models \mathcal{A}_c$ we know that $e \in B^{\mathcal{I}_{\mathcal{A}_c}}$, and since $\mathcal{I}_c$ is an extension of $\mathcal{I}_{\mathcal{A}_c}$ we get $e \in B^{\mathcal{I}_c}$ as well.

(ii) e is introduced by the chase procedure, i.e $e = w \cdot n$ is a successor of some individual w by application of some axiom $M' \sqsubseteq \exists S.N \in \nabla(\mathcal{T}, \Lambda)$. By the chase procedure we know that e satisfies exactly the concepts in N, i.e. $e \in N^{\mathcal{I}_c}$. By assumption we have that $e \in M^{\mathcal{I}_c}$ and $M \subseteq N$. It remains to see that $B \in M$.
Let's assume that $e \notin B^{\mathcal{I}_c}$, we have two cases:

- $M' \sqsubseteq \exists S.N \in \mathcal{T}$, in which case M' and N are concept names, by which we get that $N = M = B$, therefore $e \in B^{\mathcal{I}_c}$.
- $M' \sqsubseteq \exists S.N \in \nabla(\mathcal{T}, \Lambda)$, $M \sqsubseteq B \in \nabla(\mathcal{T}, \Lambda)$, $M \subseteq N$, and provided that an activator $\alpha \in \Lambda^{\nabla}$ s.t. $M' \subseteq \alpha$ exists then by application of $(\mathbf{R}^c_{\sqsubseteq})$ we get that $M' \sqsubseteq \exists S.N \sqcap B \in \nabla(\mathcal{T}, \Lambda)$, which makes the axiom $M' \sqsubseteq \exists S.N \in \nabla(\mathcal{T}, \Lambda)$ inapplicable due to Definition 35. Then it follows that $N \cup B = N$, i.e. $e \in B^{\mathcal{I}_c}$.
  It remains to show that an activator $\alpha \in \Lambda^{\nabla}$ s.t. $M' \subseteq \alpha$ exists. Since $M' \sqsubseteq \exists S.N$ is one of the introduced axioms then by Claim 3 we get that an activator $\alpha \in \Lambda^{\nabla}$ s.t. $M' \subseteq \alpha$ indeed exists.

($M \sqsubseteq \exists S.N \in \nabla(\mathcal{T}, \Lambda)$) To prove that $\mathcal{I}_c$ satisfies each existential axiom in $\nabla(\mathcal{T}, \Lambda)$ it suffices to prove that $\mathcal{I}_c$ satisfies each applicable axioms at some arbitrary element $e \in \Delta^{\mathcal{I}_c}$. To this end, let $M \sqsubseteq \exists S.N$ be an arbitrary axiom applicable at e. Now, suppose $e \in M^{\mathcal{I}_c}$ and $e \notin (\exists S.N)^{\mathcal{I}_c}$, but this leads to contradiction to the fairness condition of the chase procedure, which guarantees that such an axiom will be applied.

($A \sqsubseteq \forall r.B \in \nabla(\mathcal{T}, \Lambda)$) Let's assume an arbitrary domain element $e \in \Delta^{\mathcal{I}_c}$ and an axiom $A \sqsubseteq \forall r.B \in \nabla(\mathcal{T}, \Lambda)$ s.t. there exists some element e' where $(e, e') \in r^{\mathcal{I}_c}$ and $e' \notin B^{\mathcal{I}_c}$. Due to construction of $\mathcal{I}_c$ there are three different cases where does $r^{\mathcal{I}_c}(e, e')$ come from:

(i) $e, e' \in \mathsf{N_I}$, hence $r(e, e') \in \mathcal{I}_{\mathcal{A}_c}$, and since $\mathcal{I}_{\mathcal{A}_c} \models \mathcal{A}_c$, we reach a contradiction, i.e. $e' \in B^{\mathcal{I}_{\mathcal{A}_c}}$, hence $e' \in B^{\mathcal{I}_c}$.

(ii) $e' = e \cdot n$, i.e. e' is introduced by the chase procedure by applying some axiom $M \sqsubseteq \exists S.N \in \nabla(\mathcal{T}, \Lambda)$. We know that by construction of chase we have $e \in M^{\mathcal{I}_c}$, and $e' \in N^{\mathcal{I}_c}$. By assumption we have that $e \in A^{\mathcal{I}_c}$, then provided that an activator $\alpha \in \Lambda^{\nabla}$ s.t. $M \cup A \subseteq \alpha$ exists, ($\mathbf{R}_{\forall}$) would be applicable, hence $M \sqcap A \sqsubseteq \exists S.N \sqcap B$ will be generated, and due to maximality of $M \sqsubseteq \exists S.N$, we have that $N \sqcap B = N$.
It remains to show that an activator $\alpha \in \Lambda^{\nabla}$ such that $M \cup A \subseteq \alpha$ exists. We distinguish two cases on how e was introduced:

* $e \in \mathsf{N_I}$. We recall that $\mathcal{I}_c$ is an extension of $\mathcal{I}_{\mathcal{A}_c}$ by application of the chase procedure. Now, since Λ covers $\mathcal{A}$ w.r.t $\mathcal{T}^c$, we can prove that there exists an activator $\alpha \in \Lambda^{\nabla}$ that covers e s.t. $\{C | C^{\mathcal{I}}_{\mathcal{A}_c}(e)\} \subseteq \alpha$.
Let C be an arbitrary concept name s.t. $e \in C^{\mathcal{I}_c}$, and α an activator from Λ^{∇} that covers e. Since $\mathcal{I}_{\mathcal{A}_c}$ is the least model of $\mathcal{A}_c$, then the membership $e \in C^{\mathcal{I}_c}$ must be supported by one of the following cases:

(a) $C(e) \in \mathcal{A}$. Since α covers e then by the definition of coverage we have that $C \in \alpha$. we will get that $A \in \alpha$ as well.

(b) $B \sqsubseteq \forall r.C \in \mathcal{T}$ s.t. $s(a, e) \in \mathcal{A}$ and $r \sqsubseteq s$. Since α covers e then by definition of coverage we have that $C \in \alpha$.

(c) $B \sqsubseteq \forall r^{-}.C \in \mathcal{T}$ s.t. $s(e, a) \in \mathcal{A}$ and $r^{-} \sqsubseteq s$. Since α covers e we get that $C \in \alpha$.

(d) $M \sqsubseteq C \in \nabla(\mathcal{T}, \Lambda)$ s.t. $e \in M^{\mathcal{I}_c}$. Assuming $M \subseteq \alpha$, due to the rule Λ^* we get that $C \in \alpha$ as well. Considering that the rule Λ^* is applied exhaustively, and since we already proved that any other consequence coming from (a)-(c) is captured by α since it covers e, the assumption that $M \subseteq \alpha$ is correct.

** e is introduced by the chase procedure by applying an axiom of the form $M' \sqsubseteq \exists S.M$, and since by assumption we have that $e \in A^{\mathcal{I}_c}$, i.e. $A \in M'$ then by the rule Λ^+ we get that an activator will be create for each RHS

of each existential axiom in the set $\nabla(\mathcal{T}, \Lambda)$. Therefore there exists an activator α s.t. $M \cup A \subseteq \alpha$.

(iii) $e = e' \cdot n$, i.e. e is introduced by the chase procedure by applying some axiom $M \sqsubseteq \exists S.N \in \nabla(\mathcal{T}, \Lambda)$. We know that by construction $e \in N^{\mathcal{I}_c}$, $r^- \in S$. By assumption we get that $A \in N$. Then by the inference rule $(\mathbf{R}^-_\forall)$, provided that there exists an activator α such that $M \subseteq \alpha$, we get the following axiom $M \sqsubseteq B \in \nabla(\mathcal{T}, \Lambda)$. Since we shown that axioms of the form $M \sqsubseteq B$ satisfy $\mathcal{I}_c$, by construction of we get that $e' \in B^{\mathcal{I}_c}$.
It remains to show that an activator $\alpha \in \Lambda^\nabla$ exists such that $M \subseteq \alpha$. The proof for the existence of such activator is symmetric to the previous case.

($S \sqsubseteq r \in \nabla(\mathcal{T}, \Lambda)$) Assume a role inclusion $S \sqsubseteq r \in \nabla(\mathcal{T}, \Lambda)$ and a pair $(e, e') \in S^{\mathcal{I}_c}$. Due to the definition of $\mathcal{I}_c$, we have 3 possible cases:

(i) $e, e' \in \mathsf{N_I}$. Then $(e, e') \in r^{\mathcal{I}_c}$ because $\mathcal{I}_{\mathcal{A}_c}$ is a model of $\mathcal{A}_c$ by assumption.

(ii) $e' = e \cdot n$ for some integer n, where e' was introduced by applying some axiom $M \sqsubseteq \exists S'.N \in \nabla(\mathcal{T}, \Lambda)$ with $S \subseteq S'$. We know from the inference rule $(\mathbf{R}^r_\sqsubseteq)$ that $M \sqsubseteq \exists S' \sqcap r.N \in \nabla(\mathcal{T}, \Lambda)$ provided that an activator $\alpha \in \Lambda^\nabla$. Due to maximality of $M \sqsubseteq \exists S'.N$, we must have $S' \sqcap r = S'$, which implies $(e, e') \in r^{\mathcal{I}_c}$.
It remains to show that an activator $\alpha \in \Lambda^\nabla$ s.t. $M \subseteq \alpha$ exists. The argument is analogous to the case $(A \sqsubseteq \forall r.B)$(ii).

(iii) $e = e' \cdot n$ for some integer n, where e was introduced by applying some axiom $M \sqsubseteq \exists S'.N \in \nabla(\mathcal{T}, \Lambda)$ with $S^- \subseteq S'$. Note that $S^- \sqsubseteq r^- \in \mathcal{T}$ (see preliminaries). We know from the inference rule $(\mathbf{R}^r_\sqsubseteq)$ that $M \sqsubseteq \exists S' \sqcap r^-.N \in \nabla(\mathcal{T}, \Lambda)$. Again, due to maximality of $M \sqsubseteq \exists S'.N$, we must have $S' \sqcap r^- = S'$, which implies $(e', e) \in (r^-)^{\mathcal{I}_c}$ and $(e, e') \in r^{\mathcal{I}_c}$.
The proof for the existence of an activator $\alpha \in \Lambda^\nabla$ s.t. $M \subseteq \alpha$ is analogous to the case $(A \sqsubseteq \forall r.B)$(ii).

($A \sqsubseteq \leqslant 1\, r.B \in \nabla(\mathcal{T}, \Lambda)$) Assume an axiom $A \sqsubseteq \leqslant 1\, r.B \in \mathcal{T}$ and a domain element $e \in A^{\mathcal{I}_c}$. Suppose there is $e_1, e_2 \in \Delta^{\mathcal{I}_c}$ such that $e_1 \neq e_2$, $\{(e, e_1), (e, e_2)\} \subseteq r^{\mathcal{I}_c}$ and $\{e_1, e_2\} \subseteq B^{\mathcal{I}_c}$. We have the following possible cases:

(i) $\{e_1, e_2\} \subseteq \mathsf{N_I}$. Then by the construction of $\mathcal{I}_c$ we must have $e \in \mathsf{N_I}$. Also, $B(e_1)$ and $B(e_2)$ in $\mathcal{A}_c$. Then by the fourth completion rule of $\mathcal{A}_c$ (Definition 34) we get that $\bot(e) \in \mathcal{A}_c$ as well. Contradiction.

(ii) $e_1, e \in \mathsf{N_I}$ and e_2 is of the form $e_2 = e \cdot n$ for some integer. Assume e_2 was introduced by applying an applicable axiom $M \sqsubseteq \exists S.N \in \nabla(\mathcal{T}, \Lambda)$ at e. Note we have $e \in M^{\mathcal{I}_c}$. By the rule of the last type in the Definition 34, we have that $e_1 \in N^{\mathcal{I}_{\mathcal{A}_c}}$ and $(e, e_1) \in S^{\mathcal{I}_{\mathcal{A}_c}}$. This shows that $M \sqsubseteq \exists S.N \in \nabla(\mathcal{T}, \Lambda)$ was never applicable at e. Contradiction.

(iii) $e_2, e \in \mathsf{N_I}$ and e_1 is of the form $e_1 = e \cdot n$ for some integer. Symmetric to the above.

(iv) e_1, e_2 are of the form $e_1 = e \cdot n$ and $e_2 = e \cdot n'$. Suppose e_1, e_2 where introduced by applying axioms $M \sqsubseteq \exists S.N \in \nabla(\mathcal{T}, \Lambda)$ and $M' \sqsubseteq \exists S'.N' \in \nabla(\mathcal{T}, \Lambda)$ at e. Then by the construction of $\mathcal{I}_c$ we have $r \in S$, $r \in S'$, $B \in N$ and $B \in N'$. Then by the inference rule ($\mathbf{R}_{\leq}$), we have $M \sqcap M' \sqcap A \sqsubseteq \exists S \sqcap S'.N \sqcap N' \in \nabla(\mathcal{T}, \Lambda)$, provided that an activator $\alpha \in \Lambda^{\nabla}$ s.t. $M \cup M' \cup A \subseteq \alpha$ exists. Since $e \in (M \sqcap M' \sqcap A)^{\mathcal{I}_c}$, we have a violation of applicability of $M \sqsubseteq \exists S.N \in \nabla(\mathcal{T}, \Lambda)$ and $M' \sqsubseteq \exists S.N \in \nabla(\mathcal{T}, \Lambda)$ at e, i.e. they are not maximal.
It remains to show that there exists an activator $\alpha \in \Lambda^{\nabla}$ s.t. $M \cup M' \cup A \subseteq \alpha$. By assumption we have that $e \in M^{\mathcal{I}_c} \sqcap M'^{\mathcal{I}_c} \sqcap A^{\mathcal{I}_c}$, and considering that $e \notin \mathsf{N_I}$ then it must be a result of applying an introduced axiom at his parent by the chase, then by the Claim3 we get that an activator $\alpha \in \Lambda^{\nabla}$ s.t. $M \cup M' \cup A \subseteq \alpha$ exists.

(v) $e = e_1 \cdot n$ and $e_2 = e \cdot n'$ obtained by applying some axioms $M \sqsubseteq \exists S.N \in \nabla(\mathcal{T}, \Lambda)$ and $M' \sqsubseteq \exists S'.N' \in \nabla(\mathcal{T}, \Lambda)$ at e_1 and e, respectively. By the construction of $\mathcal{I}_c$, we have have $A \in N$, $r^- \in S$, $r \in S'$ and $B \in N'$. Then by the inference rule ($\mathbf{R}^-_{\leq}$), we have $M \sqcap B \sqsubseteq C \in \nabla(\mathcal{T}, \Lambda)$ for all $C \in N'$ and also $M \sqcap B \sqsubseteq \exists S \sqcap (S')^-.N \in \nabla(\mathcal{T}, \Lambda)$, provided that an activator $\alpha \in \Lambda^{\nabla}$ s.t. $M \cup B \subseteq \alpha$ exists. Since $e_1 \in (M \sqcap B)^{\mathcal{I}_c}$, we have $(S^-)^- \subset S$ by the maximality of $M \sqsubseteq \exists S.N$. Due to point (2) in this proof, we also have $e_1 \in C^{\mathcal{I}_c}$ for all $C \in N'$. This shows that $M' \sqsubseteq \exists S'.N' \in \nabla(\mathcal{T}, \Lambda)$ was not applicable at e, i.e. maximality violated.
It remains to show that there exists an activator $\alpha \in \Lambda^{\nabla}$ s.t. $M \cup B \subseteq \alpha$. By assumption we have that $e_1 \in M^{\mathcal{I}_c} \sqcap B^{\mathcal{I}_c}$, and considering that $e_1 \notin \mathsf{N_I}$ then it must be a result of applying one of the introduced axioms at his parent by the chase, then by the Claim 3 we get that an activator $\alpha \in \Lambda^{\nabla}$ s.t. $M \cup B$ exists.

($\bot^{\mathcal{I}_c} = \emptyset$) It remains to see that $\bot^{\mathcal{I}_c} = \emptyset$. First note that $\mathsf{N_I} \cap \bot^{\mathcal{I}_c} = \emptyset$ because $\mathcal{I}_{\mathcal{A}_c}$ is a model of $\mathcal{A}_c$. Thus it suffices to prove the following statement: if $e \cdot n \in \bot^{\mathcal{I}_c}$, then also $e \in \bot^{\mathcal{I}_c}$. Assume some $e \cdot n \in \bot^{\mathcal{I}_c}$. Suppose $e \cdot n$ was introduced by applying an axiom $M \sqsubseteq \exists S.N \in \nabla(\mathcal{T}, \Lambda)$. Then by the definition of $\mathcal{I}_c$, $\bot \in N$. By the inference rule ($\mathbf{R}_{\bot}$), we have $M \sqsubseteq \bot \in \nabla(\mathcal{T}, \Lambda)$. Since $e \in M^{\mathcal{I}_c}$, by point ($\sqsubseteq B \in \nabla(\mathcal{T}, \Lambda)$) in this proof we have $e \in \bot^{\mathcal{I}_c}$. Let $e' = a \cdot n$ be introduced through applying an axiom $M' \sqsubseteq \exists S'.N'$ where $e' \in \bot$, by inductive hypothesis we get that $a \in \bot^{\mathcal{I}_c}$. Contradiction.

Now we show (c), i.e. that $\mathcal{I}_c$ can be homomorphically embedded into any model $\mathcal{I}$ of $(\mathcal{T}, \mathcal{A})$. A homomorphism h from $\mathcal{I}_c$ to $\mathcal{I}$ can be inductively defined as follows:

- $h(e) = e^{\mathcal{I}}$ for all $e \in \mathsf{N_I} \cap \Delta^{\mathcal{I}_c}$. It is straightforward to see that $e_1 \in A^{\mathcal{I}_c}$ and $(e_1, e_2) \in r^{\mathcal{I}_c}$ imply $e_1 \in A^{\mathcal{I}}$ and $(e_1, e_2) \in r^{\mathcal{I}}$ for all $e_1, e_2 \in \mathsf{N_I}$, concepts A and roles r.

- Assume $e \cdot n \in \Delta^{\mathcal{I}_c}$ was introduced in $\mathcal{I}_c$ by an application of $M \sqsubseteq \exists S.N \in \nabla(\mathcal{T}, \Lambda)$. Note that $e \in M^{\mathcal{I}_c}$. It suffices to define $h(e \cdot n) = e'$ where $e' \in \Delta^{\mathcal{I}}$ is an element such that $S \subseteq \{r \mid (h(e), e') \in r^{\mathcal{I}}\}$ and $N \subseteq \{A \mid e' \in A^{\mathcal{I}}\}$. Note that such e' exists. Indeed, by the induction hypothesis, $h(e) \in M^{\mathcal{I}}$. Since $\mathcal{I}$ is a model of $\nabla(\mathcal{T}, \Lambda)$, we must have $h(e) \in (\exists S.N)^{\mathcal{I}}$.

Finally we show (a)$\Rightarrow$, i.e. that if $\mathcal{A}_c$ is contradiction-free then $(\mathcal{T}, \mathcal{A})$ is consistent. Assume $(\mathcal{T}, \mathcal{A})$ is inconsistent and suppose $\mathcal{A}_c$ is contradiction-free. Then there exists the least model $\mathcal{I}_{\mathcal{A}_c}$ of $\mathcal{A}_c$, and thus $\mathcal{I}_c$ is defined. As we shown in (a), $\mathcal{I}_c \models (\mathcal{T}, \mathcal{A})$. Contradiction. □

By this theorem, $\nabla(\mathcal{T}, \Lambda)$ can be used for query rewriting as in [EOv$^+$12b]. Note that the output of the original algorithm in [EOv$^+$12b] coincides with $\nabla(\mathcal{T}, \Lambda)$ if Λ contains only the set of all the concept names appearing in $\mathcal{T}$. In terms of computational complexity, the same worst-case upper bounds apply for the size of the saturated sets obtained with either version of the calculus (it may be single exponential in $\mathcal{T}$, and this exponential blow-up is in general unavoidable), but as we discuss in Section 6.2, in practice the version with activators is faster, builds smaller sets, and can handle more ontologies.

Incremental Computation of Activators Similarly as in the case of profiles, the computation of activators can be done incrementally. This is an important feature since as already stated in Chapter 3 the databases evolve over time, and most of them will constantly get new entries, moreover their structure will evolve to accommodate for new use cases.

Assuming an existing computation of $\nabla(\mathcal{T}, \Lambda)$, let $\mathcal{T}^C$ be the TBox and Λ^C be the set of activators resulting from the completion of the inference procedure. Now, suppose that the underlying structure of the ABox A' covered previously by Λ has changed and let Λ' be the minimal set of activators that covers it. Instead of running the optimized inference calculus from scratch, we need only to get the set of initial activators Λ^U of uncovered individuals e by Λ^C, i.e. for each such individual e we add it's activator to Λ^U:

$$\mathsf{prof}^{\mathcal{A}'}(e)|_{\mathsf{N_C}} \cup \mathit{prop}(\mathsf{prof}^{\mathcal{A}'}(e), \mathcal{T})$$

Note that we required for individuals to be covered against the computed Λ^C instead of initial Λ, since Λ^C is already computed from Λ w.r.t. $\mathcal{T}$ and considering that the calculus is monotonic for each activator α in Λ there exists an activator in Λ^C that is the superset of α. In the last step, instead of computing $\nabla(\mathcal{T}, \Lambda')$ we can incrementally compute:

$$\nabla(\mathcal{T}^C, \Lambda^C \cup \Lambda^U)$$

This way we avoid re-computation of all inferred axioms in $\mathcal{T}^C$ from previous computations that would have been derived otherwise.

6.2 Evaluation

The optimized calculus presented in Table 6.4 was implemented in the Clipper reasoner. From here on we will be referring to Clipper with C-orig and to our implementation with C-opt. We tested C-opt on a large test set of ontologies from the Oxford Ontology Repository,[2] and compared its performance to C-orig. The activators that were given as input to C-opt were obtained from the respective ABoxes found in the ontologies.

Oxford Ontology Repository contains 797 ontologies, from which, only 370 had ABoxes, and out of them 18 yielded exceptions while loading on C-orig. Of the remaining ontologies, 131 were uninteresting since their normalized TBoxes did not contain existential axioms (NF3), therefore the saturation step would trivialize since no rule would be fired. The remaining 221 ontologies were selected for comparing the performance of the two versions. All ontologies were preprocessed, axioms not expressible in Horn-$\mathcal{SHIQ}$ were dropped, while the others were normalized according to the Horn-$\mathcal{SHIQ}$ normal form given in Definition 12.

The test ontologies and the compiled C-opt can be found in the following repository[3].

Table 6.5: Distribution of ontologies by their respective ABox and TBox sizes.

		TBox Sizes				
		S	**M**	**L**	**VL**	**Total**
ABox Sizes	**S**	5.12%	6.51%	4.65%	0.47%	16.75%
	M	0%	9.3%	3.72%	0.93%	13.95%
	L	0%	5.12%	12.09%	0.47%	17.68%
	VL	0.47%	11.63%	14.88%	24.64%	51.62%
	Total	5.59%	32.56%	35.34%	26.51%	

In Table 6.5 the distribution of the 221 selected ontologies with respect to the sizes of TBox and ABox is shown. For convince we have categorized them into (S)mall, (M)edium, (L)arge and (V)ery (L)arge, with boundaries of up to 100, up to 1000, up to 10000, and above 10000 axioms/assertions. As shown, the considered set of ontologies have a fair mix of sizes, where around half of the ontologies have both an ABox and a TBox that are large or very large.

We recall here that all experiments were run on the same hardware, a PC with an Intel i7 2.4 GHz CPU with 4 cores running 64 bit LinuxMint 17, and a Java heap of 12 GB. The test consisted of running both reasoners against the ontologies. The key component

[2] http://www.cs.ox.ac.uk/isg/ontologies/UID/

[3] https://github.com/ghxiao/clipper-materials/tree/master/iswc-2019

we checked was the saturation procedure in C-ORIG and C-OPT. A time-out limit of 2 minutes for running the tests was set. This was the total time allowed for loading and normalizing the TBox, saturating it, and in the case of C-OPT, also the time used to obtain activators from the ABox.

In addition to successfully saturating all ontologies that C-ORIG succeeded on, C-OPT showed a 37.96% increase in the success rate: C-OPT succeeded in 149 out of 221 (67.71%), while C-ORIG in 108 out of 221 (49.33%), see Figure 6.2. Out of the 221 ontologies, 52 are in the *DL-Lite* or $\mathcal{EL}$ profiles. For them, C-OPT succeeded in 49 vs 48 for C-ORIG. If we take into account only ontologies in more expressive fragments, beyond *DL-Lite* and $\mathcal{EL}$, the performance improvement is even more pronounced: our C-OPT succeeded in 100 cases out of 169 ontologies (59.17%), while C-ORIG succeeds only in 60 cases (35.5%), resulting in an increase of 66.67% in the success rate.

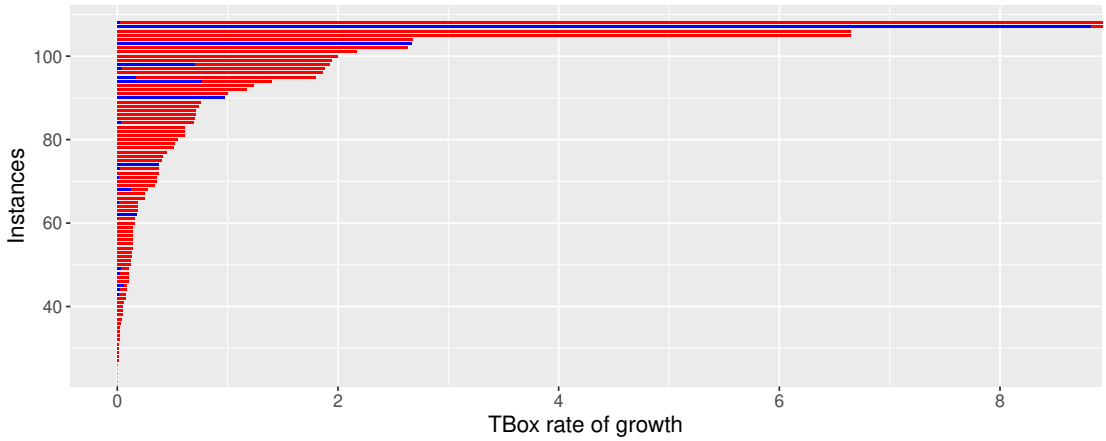
Figure 6.2: Successfull Instances.

Next we show a morefine grained analysis of tests runs along the categories shown in Figure 6.2:

- **(C1)** successful test cases for C-ORIG and C-OPT
- **(C2)** gained test cases with C-OPT
- **(C3)** failed test cases with both versions

In **(C1)**, we compare the test runs over the ontologies (108 of them) for which both versions succeeded over metrics which show the performance of TBox saturation procedure. In **(C2)**, we show the data for the same metrics as in **(C1)** for cases in which only C-OPT succeded (41 ontologies), whereas in **(C3)** we present an analysis of the cases where both versions failed (72 ontologies), and discuss the factors that we think are indicative why the C-OPT failed.

Figure 6.3: **(C1)** TBox rate of growth for both versions (C-orig =red, C-opt =blue).



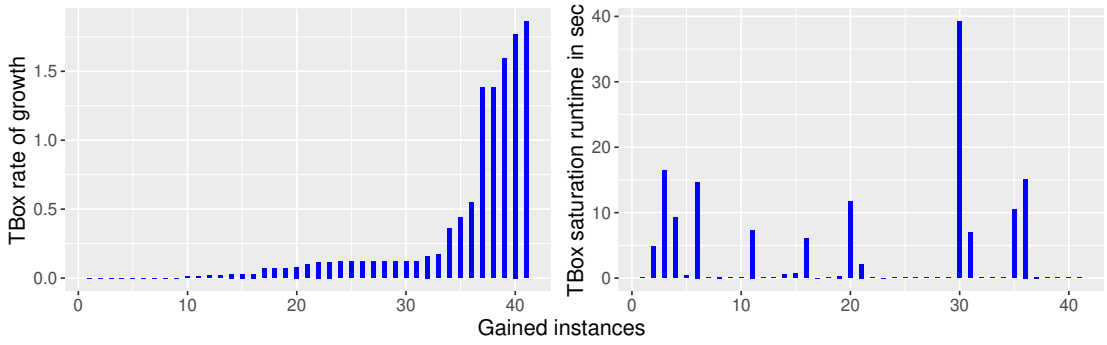
(C1) We compare the sizes of the saturated sets and the run times of both versions, on the 108 ontologies that both versions succeeded on. To better understand how much smaller the saturated set is in the optimized version, independently of the size of the original TBox, we show the *TBox rate of growth* given by the number of axioms of the form (NF1) and (NF3) in the saturated TBox, divided by their total in the initial (normalized) TBox supplied. Figure 6.3 depicts the TBox rate of growth; where blue bars that represent C-opt are plotted over the red bars representing C-orig. Note that the y-axis is simply the 108 ontologies, ordered by the rate of growth of C-orig. The x-axis is cut-off in order to avoid the distortion of the graph, as most of the cases had a growth rates below 4-folds for C-orig and even smaller for C-opt. We remark that the rate of growth of C-orig in fact reached a 20-fold growth. As shown in the figure, the rate of growth of the C-opt was nearly always smaller, often just a small fraction of C-orig, and in very few cases they were equal. For 88 ontologies the rate of growth for C-opt was zero, i.e. no new axioms were derived (see the bars without blue color). This means that all the axioms derived by C-orig were irrelevant for the ABox in the ontology, and for any ABox covered by the same profiles extracted from that ABox.

The comparison of the run times is given in Figure 6.4. TBox saturation runtime is shown in x-axis over ontologies in y-axis. Similarly as the previous graph, y-axis is ordered by the runtime of C-orig. Here as well as in Figure 6.3, the run time of C-opt (blue) is plotted over the runtime of C-orig (red). Also in this metric, C-opt outperformed C-orig in most of the cases. With very few exceptions, in which C-orig was so fast (typically under 100 milliseconds), that the overhead of handling the activators in C-opt did not pay off.

Figure 6.4: **(C1)** TBox saturation run time for both versions (C-ORIG =red, C-OPT =blue).

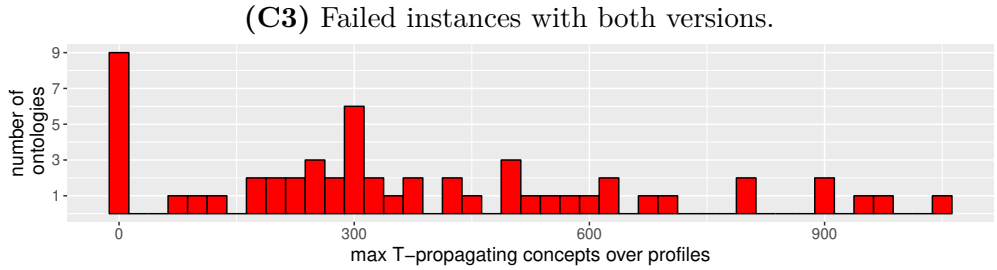
(C2) The growth rate and saturation run times of the 41 ontologies gained with C-OPT are shown in the Figure 6.5. In both graphs, the ontologies are ordered by the TBox rate of growth. The left graph shows that the rate of growth for these ontologies is in line with the growth reported in **(C1)**, remaining below double times the original size even in the worst-case. In the graph on the right one can see that the run time for most ontologies was under 10 second, which is in line with the run times in **(C1)**.

Figure 6.5: **(C2)** Gained instances with C-OPT.



(C3) We analysed the ontologies that we could not saturate, and observed that the maximal size of $\mathcal{T}$-propagating concepts over profiles plays a key role, for C-ORIG and C-OPT.

Figure 6.6: Failed instances with both versions.



From our tests we observed C-orig always failed when this number was above 20, whereas the C-opt could handle some ontologies with up to almost a hundred, but failed in all ontologies above that threshold. In Figure 6.6 we give a histogram of the maximum sizes of $\mathcal{T}$-propagating concepts over profiles extracted from the ABoxes of the failed instances. In the x-axis an ordering of the maximum number of $\mathcal{T}$-propagating concepts over any of the given profiles, whereas the y-axis shows the number of ontologies with such a number. Note that there were a few ontologies with no propagating concepts for which both versions failed. These were hard to saturate for other reasons not related to our optimization.

From **(C1)**– **(C3)**, we can conclude that the presented optimization yields improvements in three dimensions: the number of ontologies we can saturate, the run time of the inference calculus, and the size of the resulting saturated TBox.

6.3 Discussion and Related Work

In this chapter, we showed how the Horn-$\mathcal{SHIQ}$ CQ answering algorithm proposed by [EOv+12b] could be optimized. To the best of our knowledge the aforementioned algorithm remains the only one to support query answering along with all the features of Horn-$\mathcal{SHIQ}$, and its implementation in Clipper has also attracted recent adaptations [LMTV19, CDK19]. Therefore the optimization of Clipper presented in this chapter is beneficial to a broader community that build on it.

The authors of [TSCS15] also propose a CQ answering algorithm for Horn-$\mathcal{SHIQ}$ based on resolution together with implementation as a proof of concept. However, their algorithm does not support full CQs in the true sense, since they constrain the queries to not include atoms encoding transitive roles. Other authors [CGK19b] propose a technique that seems promising for Horn-$\mathcal{SROIQ}$ but constrained to deciding assertion entailments. [Kaz09] presents a consequence based driven calculus for the task of classification and briefly explains an approach used for guiding its calculus through the derivation process used in its prototype. However such an approach does not apply to the case when one does ABox reasoning as in our case.

We illustrated the problems of current ABox-independent approaches to TBox saturation, which often manifests exponential behavior, and proposed a way to overcome this.

In a nutshell, we avoid the derivation of axioms that are useless since they consider combinations of concepts that can not occur in the real data. We achieve this by constraining the axiom derivation through the use of *activators* that reflect the possible structure of the data. We implemented our approach as an optimization of the Clipper reasoner [EOv+12b], which scales well in general and can handle considerably more ontologies than the original Clipper. For future work, it would be good to pursue more refined techniques for generating initial activators as opposed to the over approximated ones we are using and see if that brings some improvement in terms of the size of the saturated TBox as well as in run-time.

CHAPTER 7

Extracting ABox Structure from OBDA Specifications

The OBDA paradigm [XCK+18] eases access to possibly heterogeneous and incomplete data sources using an *ontology*, a formal representation of the conceptualization of the domain that is written in a shareable, machine-readable language. Different sources can be linked to the same ontology, making OBDA a very effective alternative to the costly integration of data sources [XDCC19]. Integrating data through an ontology is of principal interest, considering that gathering and unifying the semantics of data across databases which by design are built to fit the information needs of specific applications is a challenging endeavour. Moreover, the user queries can be expressed over the familiar ontology vocabulary, and the knowledge in the ontology can be leveraged to infer implicit facts and obtain more query answers.

The ontology of an OBDA specification is commonly expressed in the so-called *DL-Lite* family of description logics (DLs), which as already stated through the thesis, is tailored for efficient query answering by rewriting the input query into standard SQL queries that already incorporate the relevant ontological knowledge and can be evaluated with existing mature database query engines. This central property is key to OBDA being efficiently implementable on top of current database management systems. However, many domains call for expressive features not supported in *DL-Lite*, hence real world use cases in which more expressive DLs might scale are of particular interest.

Considerable research efforts have been devoted to more expressive DLs, however data independent approaches seem to be impractical in this context. In the previous chapter we showed some results of how the structure of ABox can be effectively used for optimizing the most well know query answering algorithm for a rich Horn DL (Horn-$\mathcal{SHIQ}$). In this chapter, we want to show how the structure of families of ABoxes, paraphrased in this thesis as profiles can be extracted from the mapping layer of OBDA, which strengthens

the case for ABox structure aware algorithms, since it effectively can be extracted from the mapping layer that arguably does not evolve on the fly and is used to 'represent' a family of ABoxes that can be expected when evaluated over different databases.

For showcasing the usefulness of OBDA, let us consider a use case scenario for the anti money laundry ontology presented in the previous chapter.

Example 19. *Consider a state regulatory body, which in order to facilitate compliance with anti money laundry laws needs to gather information from financial institutions about subjects that fit certain criteria. In order to achieve this, the regulatory body shares with financial institutions the ontology from Example16. The actual storage of the accounts, owners and interactions may be rather complex, spanning different tables and databases, and is likely to differ between different (sub)organizations and financial institutions. In OBDA this is overcome by using* mappings *such as:*

$$\begin{aligned} \mathsf{sql}_1(x,y) &\rightsquigarrow \mathsf{interactedWith}(x,y), \mathsf{Account}(x), \mathsf{Account}(y) \\ \mathsf{sql}_2(x,y) &\rightsquigarrow \mathsf{hasOwner}(x,y) \\ \mathsf{sql}_3(x) &\rightsquigarrow \mathsf{PEP}(x) \end{aligned}$$

where $\mathsf{sql}_1 - \mathsf{sql}_3$ *are (possibly complex) SQL queries that specify how the data in one specific organization's database is mapped to the vocabulary of the ontology. Then the state regulatory body can supply each financial institution with a query such as the one showed in the Example 16, with which it collects the necessary data to track suspicious activities of individuals state wide.*

The mappings in an OBDA specification together with a database instance can also be seen as an 'implicit' ABox that results from evaluating them. Obtaining the profiles directly from the mapping layer of an OBDA specification enables the use of data representation based query answering algorithms such as the ones presented in this thesis to be used in a practical setting. We focus in this thesis in OBDA settings where the mappings are given in R2RML [1], which is a language for expressing the mappings from relational databases to RDF datasets.

Our approach is particularly meaningful for OBDA, where the virtual ABoxes arising from the mappings have a restricted and predictable structure. The rest of the chapter is organized as follows: in Section 7.1 we provide basic definitions local to this chapter, in Section 7.2 we give a simple algorithm for obtaining the profiles from mappings of an OBDA specification and discuss other alternatives. Finally, in Section 7.3 we report on the profiles we obtained from an analysis of three OBDA specifications with mappings expressed in R2RML.

[1] https://www.w3.org/TR/r2rml/

7.1 Preliminary Definitions

We recall the definitions of ontology-based data access and databases we use in this chapter.

A database schema $\mathcal{S}$ consists of a set of relations R and a set of functional dependencies (FDs) $\mathcal{F}$. The columns of a relation R are identified by their positions $1, \ldots, n$. For a set $\vec{i}$ of columns of R, and a tuple t of R, $t[\vec{i}]$ denotes the projection of t over $\vec{i}$. An FD F over R has the form $R : \vec{i} \to \vec{j}$, where $\vec{i}$ and $\vec{j}$ are tuples of columns in R; we call each $j \in \vec{j}$ a functional attribute in F. This FD holds in an instance $\mathcal{D}$ if the values of $\vec{i}$ determine the values of $\vec{j}$, i.e. $\vec{t_1}[\vec{i}] = \vec{t_2}[\vec{i}]$ implies $\vec{t_1}[\vec{j}] = \vec{t_2}[\vec{j}]$ for every pair of tuples $\vec{t_1}$ and $\vec{t_2}$ such that $\{R(\vec{t_1}), R(\vec{t_2})\} \subseteq \mathcal{D}$.

An *OBDA specification* is a triple $\mathcal{P} = (\mathcal{T}, \mathcal{M}, \mathcal{S})$, where $\mathcal{T}$ is a TBox (in e.g, *DL-Lite* or Horn-$\mathcal{SHIQ}$), $\mathcal{S}$ is a database schema, $\mathcal{M}$ is a mapping consisting of mapping assertions that link predicates in $\mathcal{T}$ to queries over $\mathcal{S}$. The standard W3C language for mappings is R2RML [DSC12], however here we use a more concise syntax that is common in the OBDA literature. Formally, a *mapping* $\mathcal{M}$ is a set of *mapping assertions* m that take the form

$$\text{conj}(\vec{y}) \rightsquigarrow X(\vec{f}, \vec{x})$$

consisting of a *source* part $\text{conj}(\vec{y})$, which is a conjunction of database atoms whose variables are $\vec{y}$, and a *target* part $X(\vec{f}, \vec{x})$, which is an atom whose predicate is X over terms built using function symbols $\vec{f}$ and variables $\vec{x} \subseteq \vec{y}$. In this thesis $X(\vec{f}, \vec{x})$ takes either the form $C(f(\vec{x_1}))$ for a concept name C, or the form $r(f(\vec{x_1}), g(\vec{x_2}))$ for a role name r. We say that such mapping assertion m *defines the predicate* X. We use $body(m)$ to refer to the source part $\text{conj}(\vec{y})$ of a mapping m as above, and $head(m)$ to refer to its head $X(\vec{f}, \vec{x})$.

We make the following assumptions:

(i) $\vec{a} \neq \vec{b}$ implies $f(\vec{a}) \neq f(\vec{b})$, for any f, and

(ii) $f_1 \neq f_2$ implies $f_1(\vec{a}) \neq f_2(\vec{b})$, for any $\vec{a}, \vec{b}$.

Both assumptions are in-line with the use of function symbols in OBDA systems [PLC+08], where they act as *templates* for producing a unique identifier for each input value. Assumption *(i)* is ensured in the R2RML standard using "safe separators", and although *(ii)* is not built into R2RML, it is assumed in existing OBDA tools like Ontop version 1 (implicitly in [RR15]).

In practice functions $f(\vec{x})$ take as an input a tuple of constants from the database, and generate a string that is an IRI (International Resource Identifier). Typically such functions can be seen as template strings that look like the following:

http://exampledomain/departmentid#**c**/employeeid#**d**

where **c**, **d** are place holders for the input parameters c, d of the function.

For a database instance $\mathcal{D}$ and mapping $\mathcal{M}$, the ABox $\mathcal{M}(\mathcal{D})$ is the set of atoms $X(\vec{f}, \vec{a})$, for all $\text{conj}(\vec{y}) \rightsquigarrow X(\vec{f}, \vec{x}) \in \mathcal{M}$ and all tuples **a** of constants in $\mathcal{D}$ such that $\text{conj}(\mathbf{a})$ holds in $\mathcal{D}$. An *OBDA instance* is a pair $(\mathcal{P}, \mathcal{D})$, where $\mathcal{P}$ is an OBDA specification and $\mathcal{D}$ is a database instance that satisfies the dependencies in $\mathcal{S}$ from $\mathcal{P}$. The semantics of $(\mathcal{P}, \mathcal{D})$ is given by the models of $\mathcal{T}$ and $\mathcal{M}(\mathcal{D})$.

7.2 Profiles and Activators from Mappings

In this section we show how one can obtain profiles and activators from a given OBDA specification $\mathcal{P} = (\mathcal{T}, \mathcal{M}, \mathcal{S})$, such that they cover each ABox $\mathcal{M}(\mathcal{D})$ for any legal database $\mathcal{D}$ instance for $\mathcal{S}$.

We first introduce the notion of profiles that pre-cover a given ABox.

Definition 36 (Profiles pre-Coverage of ABoxes)**.** *A set $\mathbb{P}$ of profiles* pre-covers *$\mathcal{A}$ if for each $a \in \mathsf{N_I}$ there exists a profile $p \in \mathbb{P}$ such that $\mathsf{prof}^{\mathcal{A}}(a) \subseteq p$.*

In the first step we describe how to obtain profiles that pre-cover $\mathcal{M}(\mathcal{D})$ for any legal database instance $\mathcal{D}$ for $\mathcal{S}$. Later we will see how to obtain from these pre-covering profiles a set of activators that cover all possible $\mathcal{M}(\mathcal{D})$ ABoxes, and as such can be used to optimize any Horn-DL contained in Horn-$\mathcal{SHIQ}$. We will also see how these pre-covering profiles can help towards computing a type table as in Chapter 3 by showing how one can obtain suitable profiles from the pre-covering ones.

One simple way to obtain the profiles that pre-cover relevant ABoxes would be to take as a profile all the head predicates of mappings that share the same functional symbol, roughly treating each function symbol as the same constant in the ABox. After all, if all these mappings that share some $f(\vec{x})$ in the head would be successfully evaluated for some mapping of $\vec{x}$ to the constants in $\mathcal{D}$, they could potentially all yield assertions for the same individual in ABox. This approach could potentially generate quite large profiles. As we discuss in Section 7.3, we often encounter mapping assertions that share a function in the head, but by the functional dependencies they cannot fire for the same values of $\vec{x}$. To leverage functional dependencies in obtaining a more fine-grained set of profiles, we first define conflicting mappings, i.e. mappings that in practice would never yield assertions for the same individuals when mapped over any database.

Definition 37 (Conflicting mapping assertions)**.** *Let $F = R : \vec{i} \rightarrow \vec{j}$ a functional dependency and let $j \in \vec{j}$ be one of its functional attributes. We call a pair m, m' of mapping assertions* (F, j)-conflicting *if the following hold:*

- *there are terms $f(\vec{x})$ in head(m) and $f(\vec{y})$ in head(m'), for some function symbol f, and*

- *there are atoms* $R(\vec{t}) \in body(m)$ *and* $R(\vec{t'}) \in body(m')$ *such that, for each* $i \in \vec{i}$ *we have* $\vec{t}[i] = x_i$, $\vec{t'}[i] = y_i$, *where* $x_i \in \vec{x}$, $y_i \in \vec{y}$, *and there exists a* $j \in \vec{j}$, *such that* $\vec{t}[j]$, $\vec{t'}[j]$ *are two different constants.*

Example 20. *Let* $\mathcal{M}$ *be the mapping that maps the source database to the ontology from Example 18.*

$$
\begin{aligned}
m_1 : \mathsf{transfers}(x, y) &\rightsquigarrow \mathsf{interactedWith}(f_1(x), f_1(y)), \\
&\quad \mathsf{Account}(f_1(x)), \mathsf{Account}(f_1(y)) \\
m_2 : \mathsf{account\_owners}(x, y, \text{`}politician'\text{)}) &\rightsquigarrow \mathsf{PEP}(f_2(x)) \\
m_3 : \mathsf{account\_owners}(x, y, z) &\rightsquigarrow \mathsf{hasOwner}(f_1(x), f_2(y)) \\
m_4 : \mathsf{account\_details}(x, \text{`}business', \text{`}big'\text{)}) &\rightsquigarrow \mathsf{BigBusinessAcc}(f_1(x)) \\
m_5 : \mathsf{account\_details}(x, \text{`}business', \text{`}small'\text{)}) &\rightsquigarrow \mathsf{SmallBusinessAcc}(f_1(x)) \\
m_6 : \mathsf{account\_details}(x, \text{`}private', y) &\rightsquigarrow \mathsf{IndividualAcc}(f_1(x))
\end{aligned}
$$

Now consider a functional dependency $F_1 : id \rightarrow type, size$ *defined over the relation* $\mathsf{account\_details}(id, type, size)$. *Then, according to Definition 37, pairs* m_4,m_6 *and* m_5,m_6 *are* $(F_1, type)$*-conflicting, while the pair* m_4,m_5 *is* $(F_1, size)$*-conflicting.*

Note that we define conflicts in a way that they are easy to identify, and that we can guarantee that conflicting mapping assertions cannot fire to create assertions about the same constant. Failing to identify other reasons why two mappings do not fire together does not compromise the correctness of our approach, it may simply result in larger profiles with potentially more complex neighbourhood, which may decrease the performance of the algorithm making use of the profiles.

We identify the sets of mapping assertions that can fire together, and create a profile for each of them. For a functional symbol f, we denote by $\mathcal{M}(f)$ the set of all mapping assertions in $\mathcal{M}$ such that f occurs in the head. A subset $\mathcal{M}'$ of $\mathcal{M}(f)$ is conflict-free if there are no mapping assertions m and m' in $\mathcal{M}'$ that are (F, j)-conflicting for some F and j. With $\mathbb{M}_f$ we denote the set of maximal conflict-free subsets of $\mathcal{M}(f)$. Then we can guarantee the pre-coverage of $\mathcal{M}(\mathcal{D})$ by creating a profile for each function symbol f and each $M_i \in \mathbb{M}_f$.

The problem of computing maximal conflict-free subsets can be solved by using the notions of cliques from graph theory and the hitting set problem. Recall that a clique in an undirected graph is a subset of the vertices such that every two distinct vertices are adjacent; a maximal clique is a clique that cannot be extended by adding one more vertex. For a set of sets Ω, H is a hitting set of Ω if for all $S \in \Omega, H \cap S \neq \emptyset$; a hitting set H is minimal if there exists no other hitting set H', such that $H' \subseteq H$. To compute $\mathbb{M}_f$, we first create a graph G_f where the node set is $\mathcal{M}(f)$ and the edge set is $\{(m, m') \mid m, m' \text{ are } (F, j)\text{-conflicting for some } (F, j)\}$. Next let Ω_f be the set of maximal cliques of G_f. Note that each set in Ω_f also includes the set of conflict free mapping assertions. Then every minimal hitting set of Ω_f is a maximal conflict-free subset of $\mathcal{M}(f)$. One can use any hitting set algorithm, e.g. [SC10], for this task. Note that although the problems of maximal clique and minimal hitting set are intractable in

general, there are efficient algorithms available and the sizes of instances in this case are relatively small. We also point out that minimality is not always critical. An algorithm that yields hitting sets that are small enough to be handled by the algorithms, even if not minimal, may be sufficient.

Another approach in-between the fine grained approach of computing the maximal conflict-free mappings via hitting sets and simply taking all the mappings that share the same functional symbol in the head $\mathcal{M}(f)$, would be to generate the sets of mappings by simply taking the Cartesian product of the (F, j)-conflicting mappings. One could also consider relaxing the notion of conflict, provided that such an approach gives good enough results for a given application use case.

Next we give a definition of how one can obtain a set of profiles that pre-cover any relevant ABox for a given OBDA specification.

Definition 38 (Pre-Covering profiles from an OBDA specification)**.** *Given an OBDA specification $\mathcal{P} = (\mathcal{T}, \mathcal{M}, \mathcal{S})$, let f be a function symbol occurring in $\mathcal{M}$, and let $\mathbb{M}_f$ be the set of maximal conflict-free subsets of $\mathcal{M}(f)$. Then we define $\mathbb{P}(\mathcal{P})$ as the set that for each $M_i \in \mathbb{M}_f$ contains the following profiles:*

$$\begin{aligned}\mathsf{prof}^{\mathcal{M}}(f, M_i) = &\{A \mid \textit{some } m \in M_i \textit{ has } \textit{head}(m) \textit{ of the form } A(f(x))\} \cup \\ &\{\exists r \mid \textit{some } m \in M_i \textit{ has } \textit{head}(m) \textit{ of the form } r(f(x), t)\} \cup \\ &\{\exists r^- \mid \textit{some } m \in M_i \textit{ has } \textit{head}(m) \textit{ of the form } r(t, f(x))\}\end{aligned}$$

Using the fact that, for all $\mathcal{D}$, each assertion in $\mathcal{M}(\mathcal{D})$ comes from some mapping in $\mathcal{M}$, and that only non-conflicting mappings that share a function symbol can produce assertions about a common individual, then we show the following.

Proposition 3. *For every OBDA instance $(\mathcal{P}, \mathcal{D})$, $\mathbb{P}(\mathcal{P})$ pre-covers the ABox $\mathcal{M}(\mathcal{D})$.*

Proof. Let $(\mathcal{P}, \mathcal{D})$ be an OBDA instance where $\mathcal{P} = (\mathcal{T}, \mathcal{M}, \mathcal{S})$. Let $\mathbb{P}(\mathcal{P})$ be the set of profiles obtained from $\mathcal{P}$ as per Definition 39, and $\mathcal{M}(\mathcal{D})$ the ABox obtained by evaluating $\mathcal{M}$ over $\mathcal{D}$.

We want to prove that for each individual a in $\mathcal{M}(\mathcal{D})$ there exists a profile $p \in \mathbb{P}(\mathcal{P})$ that pre-covers a as in Definition 36, i.e., $\mathsf{prof}^{\mathcal{M}(\mathcal{D})}(a) \subseteq p$. Let a be an arbitrary individual in $\mathcal{M}(\mathcal{D})$, we know that $a = f(\vec{c})$ where $\vec{c}$ is a tuple of constants from $\mathcal{D}$ and f is a function symbol appearing in the heads of a set of mapping assertions $\mathcal{M}(f) \subseteq \mathcal{M}$.

With M_a we denote the set of mappings assertions $m \in \mathcal{M}(f)$ for some mapping from variables in m to constants in $\mathcal{D}$ s.t. the $\mathit{head}(m)$ is of one of the following forms $A(a)$, $r(a, b)$, or $r(b, a)$.

In order to prove that there exists a profile $p \in \mathbb{P}(\mathcal{P})$ that pre-covers a, we first recall Definition 14 from which we have that the profile of a is:

$$\begin{aligned}\mathsf{prof}^{\mathcal{M}(D)}(a) =& \{A \mid A \in \mathsf{N_C}, A(a) \in \mathcal{M}(D)\} \cup \\ & \{\exists r \mid r \in \mathsf{N_R}, r(a,b) \in \mathcal{M}(D)\} \cup \\ & \{\exists r^- \mid r \in \mathsf{N_R}, r(b,a) \in \mathcal{M}(D)\}\end{aligned}$$

We prove the existence of a p that pre-covers a in two steps:

(a) there exists a maximal conflict-free subset M_i of $\mathbb{M}_f$ such that $M_a \subseteq M_i$; and

(b) $\mathsf{prof}^{\mathcal{M}(D)}(a) \subseteq \mathsf{prof}^{\mathcal{M}}(f, M_i)$

(a) Since, $\mathbb{M}_f$ contains all the maximal conflict-free mappings of $\mathcal{M}(f)$ and since $M_a \subseteq \mathcal{M}(f)$, next we prove that there exists some $M_i \in \mathbb{M}_f$ such that $M_a \subseteq M_i$.
In order to prove that there exists such an $M_i \in \mathbb{M}_f$ such that $M_a \subseteq M_i$, we only need to prove that there exists no pair of mappings in M_a that violate the Definition 37 of conflicting mappings. To this end, let's assume that there exist a pair of mappings $m, m' \in M_a$ such that they are (F, j)-conflicting, where $F = R : \vec{i} \to \vec{j}$ and $j \in \vec{j}$ a functional attribute of F, then according to the Definition 37 and definition of M_a we have that:

- there exists some predicate R s.t. $R(\vec{b}) \in m$ and $R(\vec{b'}) \in m'$; and
- both mappings share the same functional symbol f where $f(\vec{c}) \in \mathit{head}(m)$, $f(\vec{c}) \in \mathit{head}(m')$; and
- $\vec{b}[i] = c_i$, $\vec{b'}[i] = c_i$ for each $i \in \vec{i}$, where $c_i = \vec{c}[i]$; and
- there exists some $j \in \vec{j}$ s.t. $\vec{b}[j] = d$ and $\vec{b'}[j] = d'$

However such mappings from variables of R to constants violates the functional dependency F, and since $\mathcal{D}$ is a legal database satisfying all the data base constraints, we get a contradiction to our initial assumption that there exist a pair of conflicting mappings in M_a.

Now, since $\mathbb{M}_f$ contains all maximal-conflict free subsets of $\mathcal{M}(f)$, from $M_a \subseteq \mathcal{M}(f)$ and the fact that all mapping assertions in M_a are conflict free we get that there exists an $M_i \in \mathbb{M}_f$, such that $M_a \subseteq M_i$.

(b) We want to prove that $\mathsf{prof}^{\mathcal{M}(\mathcal{D})}(a) \subseteq \mathsf{prof}^{\mathcal{M}}((f, M_i))$. Considering that each assertion in $\mathcal{M}(\mathcal{D})$ comes from successful evaluation of mappings, i.e., mappings in M_a, we can rewrite the formula for $\mathsf{prof}^{\mathcal{M}(D)}(a)$ into:

$$\begin{aligned}\mathsf{prof}^{\mathcal{M}(D)}(a) =& \{A \mid \textit{some } m \in M_a \textit{ has head } A(f(c))\} \cup \\ & \{\exists r \mid \textit{some } m \in M_a \textit{ has head } r(f(c), f(c'))\} \cup \\ & \{\exists r^- \mid \textit{some } m \in M_a \textit{ has head } r(f(c'), f(c))\}\end{aligned}$$

Since, $M_a \subseteq M_i$ then from Definition 39 we have:

$$\begin{aligned}\mathsf{prof}^{\mathcal{M}}(f, M_i) = &\{A \mid \text{some } m \in M_i \text{ has } head(m) \text{ of the form } A(f(x))\} \cup \\ &\{\exists r \mid \text{some } m \in M_i \text{ has } head(m) \text{ of the form } r(f(x), t)\} \cup \\ &\{\exists r^- \mid \text{some } m \in M_i \text{ has } head(m) \text{ of the form } r(t, f(x))\}\end{aligned}$$

from where we conclude $\mathsf{prof}^{\mathcal{M}(D)}(a) \subseteq \mathsf{prof}^{\mathcal{M}}(f, M_i)$. □

7.2.1 Covering Activators

Activators as defined in 32 are the cornerstone of the optimization calculus shown in Table 6.4 for any Horn DL contained in Horn-$\mathcal{SHIQ}$. We show in this subsection that we can obtain the set of activators from an OBDA specification, such that they cover any possible ABox that may result from the evaluation of the OBDA specification over legal database instances. Hence, this set of activators can be used by the optimized calculus for any of the resulting ABoxes. To obtain the set of activators we use the pre-covering profiles as defined in 39 in the following manner.

Definition 39 (Covering Activators from an OBDA specification)**.** *Given an OBDA specification $\mathcal{P} = (\mathcal{T}, \mathcal{M}, \mathcal{S})$, we denote by $\Lambda(\mathcal{P})$ the set of activators that contains, for each $p \in \mathbb{P}(\mathcal{P})$:*

$$p|_{\mathsf{N_C}} \cup prop(p, \mathcal{T})$$

where $p|_{\mathsf{N_C}}$ denotes the restriction of p to concept names only, and $prop(p, \mathcal{T})$ the set of propagated concepts from the neighbourhood of p as given in Definition 32.

The following proposition shows that for a given OBDA specification $\mathcal{P} = (\mathcal{T}, \mathcal{M}, \mathcal{S})$, the set of activators $\Lambda(\mathcal{P})$ obtained as defined above cover $\mathcal{M}(\mathcal{D})$ for any database $\mathcal{D}$ adhering to $\mathcal{S}$.

Proposition 4. *For every OBDA instance $(\mathcal{P}, \mathcal{D})$, $\Lambda(\mathcal{P})$ covers the ABox $\mathcal{M}(\mathcal{D})$.*

Proof. From Definition 32 we have that $\Lambda(\mathcal{P})$ covers $\mathcal{M}(\mathcal{D})$ if for each $a \in \mathcal{M}(D)$ there exists an activator $\alpha \in \Lambda(\mathcal{P})$ s.t.:

$$\mathsf{prof}^{\mathcal{M}(D)}(a)|_{\mathsf{N_C}} \cup prop(\mathsf{prof}^{\mathcal{M}(D)}(a), \mathcal{T}) \subseteq \alpha \tag{7.1}$$

In Proposition 3 we proved that $\mathbb{P}(\mathcal{P})$ pre-covers $\mathcal{M}(\mathcal{D})$, i.e., for each $a \in \mathcal{M}(\mathcal{D})$ there exists a profile $p \in \mathbb{P}(\mathcal{P})$ s.t.:

$$\mathsf{prof}^{\mathcal{M}(D)}(a) \subseteq p \tag{7.2}$$

Then according to Definition 39 there exists an activator $\alpha \in \Lambda(\mathcal{P})$ obtained in the following manner:

$$p|_{\mathsf{N_C}} \cup prop(p, \mathcal{T}) \subseteq \alpha \tag{7.3}$$

Then from (7.1), (7.2) and (7.3) we get that α covers a. □

Example 21. *Let's consider mapping assertions from Example 20. For the functional symbol f_1, we get the graph $G_{f_1} = (E_{f_1}, V_{f_1})$, where*

$E_{f_1} = \{m_1, m_3, m_4, m_5, m_6\}$, *and*
$V_{f_1} = \{(m_4, m_5), (m_4, m_6), (m_5, m_6)\}$.

The maximal cliques are $\Omega_{f_1} = \{\{m_1\}, \{m_3\}, \{m_4, m_5, m_6\}\}$. Thus,the maximal conflict-free sets $\mathbb{M}_{f_1}$ are the minimal hitting sets of Ω_{f_1}, i.e. $\mathbb{M}_{f_1} = \{M_1, M_2, M_3\}$, where

$M_1 = \{m_1, m_3, m_4\}$
$M_2 = \{m_1, m_3, m_5\}$
$M_3 = \{m_1, m_3, m_6\}$

Similarly, the maximal conflict-free sets for f_2 is $\mathbb{M}_{f_2} = \{M_4\}$ where

$M_4 = \{m_2, m_3\}$

Then by Definition 39 we get:

$$\begin{aligned}
\mathsf{prof}^{\mathcal{M}}(f_1, M_1) &= \{\exists\mathsf{interactedWith}, \exists\mathsf{interactedWith}^-, \mathsf{Account}, \mathsf{hasOwner}, \mathsf{BigBusinessAcc}\} \\
\mathsf{prof}^{\mathcal{M}}(f_1, M_2) &= \{\exists\mathsf{interactedWith}, \exists\mathsf{interactedWith}^-, \mathsf{Account}, \mathsf{hasOwner}, \mathsf{SmallBusinessAcc}\} \\
\mathsf{prof}^{\mathcal{M}}(f_1, M_3) &= \{\exists\mathsf{interactedWith}, \exists\mathsf{interactedWith}^-, \mathsf{Account}, \mathsf{hasOwner}, \mathsf{IndividualAcc}\} \\
\mathsf{prof}^{\mathcal{M}}(f_1, M_4) &= \{\exists\mathsf{hasOwner}^-, \mathsf{Politician}\}
\end{aligned}$$

and, the following set of activators:

$$\begin{aligned}
\Lambda = \quad & \{\{\mathsf{Account}, \mathsf{BigBusinessAcc}\}, \{\mathsf{Account}, \mathsf{SmallBusinessAcc}\}, \\
& \{\mathsf{Account}, \mathsf{IndividualAcc}\}, \{\mathsf{Politician}\}\}
\end{aligned}$$

Going back to the Example 18, note that, with the Λ obtained in example above, the $\mathbf{R}_\forall$ rule of the optimised calculus would have not derived the irrelevant axioms that were discussed there.

7.2.2 Covering Profiles

For the type table computation algorithm 3.1 for $\mathcal{ALCHI}$ DL, a set of profiles that covers the ABoxes as per the Definition 14 is needed. Here we explain how for a given OBDA specification one can obtain such set of profiles, i.e, profiles that cover any ABox that may result from that specification. A simple approach is to take all the possible subsets for each profile in pre-covering $\mathbb{P}$ which cover any possible resulting ABox from the specification. We propose a slight refinement of this naive technique in which we get a set of profiles that do not formally cover the ABoxes but still are complete for computing the type tables.

To define a weaker notion of 'sutiable' profiles, we rely on the the way the base types of profiles are composed. We recall here parts from the base types Definition 16:

$$\mathsf{btyp}^{\mathcal{T}}(p) = \{\mathsf{btyp}(p, S) \mid S \subseteq \mathit{Guess}^{\mathcal{T}}(p), \bot \notin \mathsf{btyp}(p, S)\}$$

where

$$\mathsf{btyp}(p, S) = detCl^{\mathcal{T}}(p \cup S)$$

Notice that, for any given ABox $\mathcal{A}$, and any given set of profiles $\mathbb{P}$, in order for the type table computation algorithm to be complete, for the purpose of computing the good types for $\mathcal{A}$ it would suffice that for each $a \in \mathsf{N_I}(\mathcal{A})$ the base types of the $\mathsf{prof}^{\mathcal{A}}(a)$ are contained in the set of base types of some profile in $\mathbb{P}$. Clearly, in such a case all the types needed would have already been computed and ready to use. We call such profiles suitable profiles.

Definition 40 (Suitable Profiles)**.** *Given an ABox and a TBox in $\mathcal{ALCHI}$, and a set of profiles $\mathbb{P}^S$, we call $\mathbb{P}^S$ suitable if for each $a \in \mathsf{N_I}(\mathcal{A})$ we that $\mathsf{btyp}^{\mathcal{T}}(\mathsf{prof}^{\mathcal{A}}(a)) \subseteq \mathsf{btyp}^{\mathcal{T}}(\mathbb{P}^S)$.*

Therefore a set of profiles $\mathbb{P}^S$ is suitable for some OBDA specification, if for any ABox $\mathcal{M}(\mathcal{D})$ that may result from the evaluation of the mappings $\mathcal{M}$ over some database $\mathcal{D}$, and any individual $a \in \mathsf{N_I}(\mathcal{M}(\mathcal{D}))$ we have $\mathsf{btyp}^{\mathcal{T}}(\mathsf{prof}^{\mathcal{M}(\mathcal{D})}(a)) \subseteq \mathsf{btyp}^{\mathcal{T}}(\mathbb{P}^S)$. Next, we show how a set of suitable profiles from pre-covering profiles can be obtained.

Definition 41 (Suitable Profiles from Pre-Covering Profiles)**.** *Given a set of pre-covering profiles $\mathbb{P}$ of an OBDA specification $\mathcal{P}$, for each profile p we define:*

- $C_p = \{det^{\mathcal{T}}(\{A\}) \mid$ *for each* $A \in p \cap \mathsf{N_C}\}$*, i.e., the deterministic closure of each concept name in the profile under axioms of the form (NF1).*
- $R_p = \{r \mid r \in p \setminus \mathsf{N_C}$ *and* $detCl^{\mathcal{T}}(r) \neq \emptyset\}$*, i.e., the set of incoming and outgoing roles that fire some domain or range axiom.*
- $R'_p = (p \setminus R_p) \setminus \mathsf{N_C}$.

Then, for each p and the set $\mathbb{P}$ we get the following set of suitable profiles:

$$\mathbb{P}^S_p = \{conc \cup roles \cup R'_p \mid conc \subseteq C_p \text{ and } roles \subseteq R_p\}$$
$$\mathbb{P}^S = \bigcup_{p \in \mathbb{P}} \mathbb{P}^S_p$$

Note that, even if we use the suitable profiles, in some cases the number of them may be quite large.

The set of suitable profiles computed this way can of course be very large, but we suspect that in many cases an incremental computation may be useful, where one can consider a more practical approach in which for any pre-covering profile p that is evaluated to have an associated $\mathbb{P}^S_p$ with a large number of profiles, we omit the generation of those profiles and instead run the computation in incremental mode. This way the computation would be done only for few profiles of a given ABox that are not contained in the set of suitable profiles.

7.3 Validating Profile Extraction from OBDA Specifications

All ontologies used for evaluation in other chapters had no accompanying mapping specifications. With the purpose of understanding the feasibility of obtaining the profiles when ABoxes come from real OBDA specifications, we analysed three OBDA specification benchmarks:

- NPD [LRXC15] (1173 mapping assertions).
- Slegge [HKS+17] (62 mapping assertions).
- UOBM [BCS+16] (96 mapping assertions).

While UOBM is synthetic, the other two are from real-world scenarios. We analyzed the above listed OBDA specification in terms of the sizes of profiles they would generate using the algorithm provided in Section 7.2. Files with the mapping analysis can be found in the following repository [2].

In the case of NPD mappings we encountered 107 functional symbols, 37 in the case of Slegge and 8 in the case of UOBM, which in the plain approach in which we would not consider the conflicts would mean that we get 107 profiles for NPD, 37 for Slegge and 8 for UOBM. However, as previously advocated in such a case the structure of profiles could be quite complex with many incoming and outgoing roles and concept assertions.

Using the algorithm in Section 7.2 which exploits functional dependencies, we obtained a set of pre-covering profiles with over 600 profiles for NPD, 12 for UOBM, and only 4 for Slegge. In terms of the size of the largest profile, for all three ontologies we got similar sizes. The largest profile obtained for NPD had 9 concept names, Slegge had 3 concept names, and UOBM had 11 concept names. In terms of the number of profiles, NPD had significantly more profiles. This is due to larger number of conflicts in the mappings of NPD compared to Slegge and UOBM. We recall that when conflicts are present, the number of maximal conflict-free subsets is higher, hence inducing more profiles. However, from our observations during the evaluation of Mod4Q, the size of the profiles plays far greater role in the performance of type table algorithm then the number of profiles.

In all the cases above the ontologies were simple, i.e., there were no propagating concepts, hence the initial activators as presented in the previous chapter would coincide with the set of concept names in the profiles. If we recall the results we got for ontologies from the Oxford repository with similar ABoxes (namely, they contained 5 or more assertions in the ABox type of some individual). From the view point of the sizes of the activators we obtained from the mappings, an ontology paired with those set of activators should be manageable for the optimized calculus presented in previous chapter.

[2] https://github.com/ghxiao/clipper-materials/tree/master/iswc-2019

From the first look at the pre-covering profiles, we noticed that the number of suitable profiles is still manageable, however for full evaluation of the technique one needs to have OBDA specifications with expressive DLs including the data. The type table technique makes sense to be evaluated in expressive settings. Since in the case of OBDA with *DL-Lite* the profiles we would have gotten zero guesses, and furthermore the lack of non-determinism would imply that for each profile we would get only one type. Hence using ASP solvers that guess the types for each profiles in a setting where no guesses are needed would be an overkill. Therefore an end to end evaluation of the technique, i.e., from obtaining the profile up to computing the types remains subject of further tests in the future, once OBDA specifications with expressive DLs become available.

7.4 Discussion

In this chapter, we showed how we the mapping layer of the OBDA specification can be exploited for obtaining the set or profiles as defined in Chapter 3, such that the obtained profiles cover all the ABoxes that could be materialized by evaluating the mappings over any given database instance. We gave a simple algorithm for computing the profiles, which has important implications on the overall practicability of our data representation approach since it makes it more viable for deploying it in practical settings.

We corroborated the presented algorithm in Section 7.2 into existing OBDA specifications. The results of the analysis of three OBDA specifications were presented in the previous Section, with encouraging observations in the size and shape of the profiles. We hope this work will bring closer the goal of realizing OBDA with ontologies beyond *DL-Lite*. Moreover, we believe that a careful implementation of the whole technique from obtaining the profiles from the OBDA mappings to the evaluation of the translations in ASP could yield first OBDA systems for more expressive DLs in use cases for which the underlying data representation algorithms such as the ones presented in this thesis can scale.

Lastly, we note that getting OBDA specification of real systems remains a challenge, and we recognize the need for validating our approach against more instances, and in particular against instances paired with data and expressive DLs that allow for end-to-end evaluation. Another direction worth pursuing towards further refinements of the technique is to conduct extensive studies with more real-world OBDA specifications in order to identify even more fine-grained approaches for generating the maximal conflict-free sets of mappings, and studying the trade-offs between them and the more relaxed approaches in concrete application use cases.

CHAPTER 8

Summary and Conclusions

OMQ answering is a challenging problem in general, and even more so when one considers expressive ontologies. In the light of the high complexity of the problem for expressive DLs, the current state-of-the-art research is focused mainly on theory-oriented goals, and to the best of our knowledge, there has been no attempt in implementing query answering algorithms beyond instance queries for expressive DLs.

We believe that in practice, a significant obstacle for tackling the problem of query answering for expressive DLs is the standard data *independent* approach to query rewriting. Therefore we set as the primary goal of this thesis to obtain a technique that uses the knowledge about the structure of the data for guiding the query answering algorithms. We hope that the approach developed in this thesis will inspire further research in this direction, and culminate with efficient systems for query answering algorithms in expressive DLs.

We have presented our representation of the structure of the data in Chapter 3. More specifically, we represented the structure of families of ABoxes via what we call *profiles*, which play a key role in our approach. Profiles were used to compile model representations for the $\mathcal{ALCHI}$ DL, which are expressed as a structure $\mathbb{T}$ that stores all the relevant types and their relations with respective successor witnessing existential axioms. The algorithm we presented runs in goal-oriented way, adding types along the way and reusing previous computations. Hence, contrary to the existing ones, it is amenable to implementation. To show this, we implemented the type table computation algorithm in a proof of concept prototype Mod4Q. We presented in Chapter 3 evaluation results from which there are two take away messages: (i) the sizes of the profile sets for real-world ontologies with large ABoxes is expected to be small; and (ii) the model compilation computation $\mathbb{T}$ is feasible for most complex ontologies with large TBoxes. We noticed 80% success rate against a significant number of ontologies from different application domains obtained from Oxford Ontology Repository, some of which were quite large and complex. However, we note that our technique also has limitations, for roughly 20% of ontologies that force

a lot of guesses in our profiles, our computation was infeasible; therefore we believe that further investigation on refining the representations is needed in order to push the scalability of the technique to a larger number of cases.

In Chapter 4 we proved that our model representation via type table computations $\mathbb{T}$ is sufficient for answering any query preserved under homomorphisms. Furthermore, we gave algorithms for answering *instance queries*, *reachability queries*, and *semi-full conjunctive query with reachability atoms* an interesting query language that combines RQs and CQs with restrictions on the existential quantification of variables. An important feature of our approach is that the computation of the model representations is done offline. Moreover, answers for each possible instance query can be obtained directly from the answer sets of the ASP program, which makes our technique particularly suitable for the task of ABox materialization. We showed some promising results from testing the implemented algorithms for instance queries and reachability queries over the real world ontologies reported in Chapter 3, and observed reasonable answering times. We also remark that our model representation has similarities with the knot representation presented in [EOv12a], and the CQ answering algorithm presented there can be adopted for answering CQs in our approach. Therefore we envision that obtaining an algorithm that is implementable for CQs inspired by the one in [EOv12a] is the next important step to undertake.

In addition to showing that our approach in the setting of expressive DLs has the potential for implementation in practice, in Chapter 5 we took the idea one step further by showing that our approach can be also exploited in the context of expressive hybrid languages. For this purpose, we presented an expressive new hybrid language which we call *Clopen KBs* (CKBs), which generalizes and improves the prominent r-hybrid language [Ros05]. We showed that when the ontology in a CKB is expressed in $\mathcal{ALCHOI}$ and we assume bounded predicate arities in rules, the complexity of basic reasoning problems coincides with the complexity of standard problems in plain $\mathcal{ALCHOI}$. Toward the purpose of obtaining a practicable algorithm for *Clopen KBs*, we presented separable CKBs which still generalize r-hybrid. We presented two translations of separable CKBs; the first translation is a direct translation of separable CKBs to dl-programs, while the second a translation into plain ASP programs. We compared both approaches with four CKBs using real world data and a real ontology (MyITS). The translation in dl-programs run with dlvhex failed to finish within the time-out limit. On the other hand, the translation into plain ASP optimized using profiles performed well for different sizes of facts with linear growth of program evaluation times. Both translations used clingo as the ASP solver. Although limited to a small set of examples, our tests showed that there is potential for obtaining implementable algorithms in the very expressive setting of hybrid languages using our data representation-based approach.

In Chapter 6, we showed that profiles can be employed in a different setting than initially envisioned. Instead of using the profiles for computing the representation of models for expressive DLs, we showed that they can be adapted for optimizing algorithms in Horn yet expressive DLs. We optimized a current state of the art algorithm for CQ

answering in Horn-$\mathcal{SHIQ}$ [EOv$^+$12b] by using profiles for obtaining the set of *activators*-which act as a constraining mechanism inside the TBox saturation inference calculus. Activators contain information regarding the set of concepts that an individual from an ABox covered by a profile can potentially satisfy. They helped us prune away superfluous inferences, which for many ontologies were being derived with the original algorithm. An implementation of the activators together with the rules and constraint conditions of the optimized calculus was implemented in Clipper and was compared with the original Clipper against a range of ontologies. We reported significant improvements. Many ontologies that were untameable with the original implementation became manageable with the optimized Clipper, showcasing in this way the benefits of using the representation based approach instead of independent based query rewriting.

Finally, in Chapter 7, we showed that our abstract representation of families of ABoxes via profiles can be obtained from existing OBDA specifications with R2RML mappings. We gave a simple algorithm and a refinement technique for computing the profiles from a given OBDA specification and observed manageable set of profiles from the analysis of three real-world OBDA specifications. These results, together with the modularity of our approach, and its support for incremental computation, encourage further development and refinement of structure based approaches to OMQ answering.

For future research, we think that extending our type table $\mathbb{T}$ computation algorithm to more expressive DLs and developing algorithms for supporting other query languages like conjunctive queries should be undertaken next. We note here that Mod4Q is meant as a proof of concept and due to time resources, not much performance engineering has been put into it. Therefore we believe that it is equally important to continue the development of Mod4Q with performance engineering in mind, and fitting its architecture to take full advantage of incremental reasoning, a feature which due to the unavailability of data has not been tested. Moreover, we believe that our approach allows for parallelization of the computation of extended types distributed across different workers, and as such, it remains an option to think about. With proper developmental resources our approach could yield the first system that supports query answering for expressive DLs for a range of application domains. In the end, we want to emphasize that obtaining more benchmarking data for the DL community remains a challenging task. This is especially true for expressive ontologies, which commonly come without real-world data because application use cases of expressive DLs have targeted almost exclusively TBox reasoning tasks. Therefore, for further validating our approach we think that identifying more application use cases (such as MyITS) with large complex TBoxes in expressive DLs remains an important goal.

List of Figures

List of Tables

List of Algorithms

Bibliography

[AKS13] José Júlio Alferes, Matthias Knorr, and Terrance Swift. Query-driven procedures for hybrid MKNF knowledge bases. *ACM Trans. Comput. Logic*, 14(2):16:1–16:43, June 2013.

[AOS16] Shqiponja Ahmetaj, Magdalena Ortiz, and Mantas Simkus. Polynomial datalog rewritings for expressive description logics with closed predicates. In *IJCAI 2016, USA*, 2016.

[BBL05] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the EL envelope. In *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*, pages 364–369, 2005.

[BBLW16] Franz Baader, Meghyn Bienvenu, Carsten Lutz, and Frank Wolter. Query and predicate emptiness in ontology-based data access. *J. Artif. Intell. Res.*, 56:1–59, 2016.

[BBtCP16] Michael Benedikt, Pierre Bourhis, Balder ten Cate, and Gabriele Puppis. Querying visible and invisible information. In *Proc. of LICS 2016*, pages 297–306. ACM, 2016.

[BCH+14] Timea Bagosi, Diego Calvanese, Josef Hardi, Sarah Komla-Ebri, Davide Lanti, Martin Rezk, Mariano Rodriguez-Muro, Mindaugas Slusnys, and Guohui Xiao. The ontop framework for ontology based data access. In *The Semantic Web and Web Science - 8th Chinese Conference, CSWS 2014, Wuhan, China, August 8-12, 2014, Revised Selected Papers*, pages 67–77, 2014.

[BCM+03] 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, New York, NY, USA, 2003.

[BCOv14] Meghyn Bienvenu, Diego Calvanese, Magdalena Ortiz, and Mantas Šimkus. Nested regular path queries in description logics. AAAI Press, 2014.

[BCS15] Vince Bárány, Balder Ten Cate, and Luc Segoufin. Guarded negation. *J. ACM*, 62(3):22:1–22:26, June 2015.

[BCS+16] Elena Botoeva, Diego Calvanese, Valerio Santarelli, Domenico F. Savo, Alessandro Solimando, and Guohui Xiao. Beyond OWL 2 QL in OBDA: rewritings and approximations. In *AAAI*, pages 921–928. AAAI Press, 2016.

[BET11] Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski. Answer set programming at a glance. *Commun. ACM*, 54(12):92–103, 2011.

[BHLS17] Franz Baader, Ian Horrocks, Carsten Lutz, and Ulrike Sattler. *An Introduction to Description Logic*. Cambridge University Press, 2017.

[BKR+16] Elena Botoeva, Roman Kontchakov, Vladislav Ryzhikov, Frank Wolter, and Michael Zakharyaschev. Games for query inseparability of description logic knowledge bases. *Artif. Intell.*, 234:78–119, 2016.

[BLB08] Franz Baader, Carsten Lutz, and Sebastian Brandt. Pushing the EL envelope further. In *Proceedings of the Fourth OWLED Workshop on OWL: Experiences and Directions, Washington, DC, USA, 1-2 April 2008.*, 2008.

[BLR+19] Elena Botoeva, Carsten Lutz, Vladislav Ryzhikov, Frank Wolter, and Michael Zakharyaschev. Query inseparability for ALC ontologies. *CoRR*, abs/1902.00014, 2019.

[BO15] Meghyn Bienvenu and Magdalena Ortiz. Ontology-mediated query answering with data-tractable description logics. In *Reasoning Web. Web Logic Rules - 11th International Summer School 2015, Berlin, Germany, July 31 - August 4, 2015, Tutorial Lectures*, pages 218–307, 2015.

[BOS15] Meghyn Bienvenu, Magdalena Ortiz, and Mantas Simkus. Regular path queries in lightweight description logics: Complexity and algorithms. *J. Artif. Intell. Res.*, 53:315–374, 2015.

[BOS17] Labinot Bajraktari, Magdalena Ortiz, and Mantas Simkus. Clopen knowledge bases: Combining description logics and answer set programming. In *Proceedings of DL2017, Montpellier, France*, 2017.

[BOS18a] Labinot Bajraktari, Magdalena Ortiz, and Mantas Simkus. Combining rules and ontologies into clopen knowledge bases. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 1728–1735, 2018.

[BOS18b] Labinot Bajraktari, Magdalena Ortiz, and Mantas Simkus. Compiling model representations for querying large aboxes in expressive dls. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden.*, pages 1691–1698, 2018.

[BOX19] Labinot Bajraktari, Magdalena Ortiz, and Guohui Xiao. Optimizing horn-SHIQ reasoning for OBDA. In *To appear: Proceedings of the Eighteenth International Semantic Web Conference, ISWC 2019, October 26-30, 2019, Auckland, New Zealand.*, 2019.

[BtCLW14] Meghyn Bienvenu, Balder ten Cate, Carsten Lutz, and Frank Wolter. Ontology-based data access: A study through disjunctive datalog, csp, and MMSNP. *ACM Trans. Database Syst.*, 39(4):33:1–33:44, 2014.

[CDK19] David Carral, Irina Dragoste, and Markus Krötzsch. The combined approach to query answering in horn-alchoiq (extended abstract). In *Proceedings of the 32nd International Workshop on Description Logics, Oslo, Norway, June 18-21, 2019.*, 2019.

[CDL+98] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Description logic framework for information integration. In *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98), Trento, Italy, June 2-5, 1998.*, pages 2–13, 1998.

[CDL+05] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Dl-lite: Tractable description logics for ontologies. In *In AAAI/IAAI 2005, USA*, 2005.

[CDL08] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Conjunctive query containment and answering under description logic constraints. *ACM Trans. Comput. Log.*, 9(3), 2008.

[CDL+11] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Mariano Rodriguez-Muro, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. The MASTRO system for ontology-based data access. *Semantic Web*, 2(1):43–53, 2011.

[CDLV02] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Rewriting of regular expressions and regular path queries. *J. Comput. Syst. Sci.*, 64(3):443–465, 2002.

[CDLV03] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Reasoning on regular path queries. *SIGMOD Record*, 32(4):83–92, 2003.

[CEO14] Diego Calvanese, Thomas Eiter, and Magdalena Ortiz. Answering regular path queries in expressive description logics via alternating tree-automata. *Inf. Comput.*, 237:12–55, 2014.

[CGK19a] David Carral, Larry González, and Patrick Koopmann. From horn-sriq to datalog: A data-independent transformation that preserves assertion entailment. In *AAAI*, 2019.

[CGK19b] David Carral, Larry Gonzalez, and Patrick Koopmann. From horn-SRIQ to datalog:a data-independent transformation that preserves assertion entailment. In *In AAAI 2019, USA*, 2019.

[DFH11] John Domingue, Dieter Fensel, and James A. Hendler, editors. *Handbook of Semantic Web Technologies*. Springer, 2011.

[DSC12] Souripriya Das, Seema Sundara, and Richard Cyganiak. R2RML: RDB to RDF mapping language. W3C recommendation, W3C, 2012.

[EGL16] Esra Erdem, Michael Gelfond, and Nicola Leone. Applications of answer set programming. *AI Magazine*, 37(3):53–68, 2016.

[EIK09] Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. Answer set programming: A primer. In *Reasoning Web. Semantic Technologies for Information Systems, 5th International Summer School 2009, Brixen-Bressanone, Italy, August 30 - September 4, 2009, Tutorial Lectures*, pages 40–110, 2009.

[EIL+08] Thomas Eiter, Giovambattista Ianni, Thomas Lukasiewicz, Roman Schindlauer, and Hans Tompits. Combining answer set programming with description logics for the semantic web. *Artif. Intell.*, 172(12-13):p. 1495, 2008.

[EKS13] Thomas Eiter, Thomas Krennwallner, and Patrik Schneider. Lightweight spatial conjunctive query answering using keywords. In *Proc. of ESWC 2013*. Springer, 2013.

[EOv12a] Thomas Eiter, Magdalena Ortiz, and Mantas Š. Conjunctive query answering in the description logic SH using knots. *J. Comput. Syst. Sci.*, 78(1):47–85, 2012.

[EOv+12b] Thomas Eiter, Magdalena Ortiz, Mantas Šimkus, Trung-Kien Tran, and Guohui Xiao. Query rewriting for Horn-SHIQ plus rules. AAAI Press, 2012.

[EPS+15] Thomas Eiter, Jeff Z. Pan, Patrik Schneider, Mantas Šimkus, and Guohui Xiao. A rule-based framework for creating instance data from OpenStreetMap. In *Proc. of RR 2015*. Springer, 2015.

[FFS+18] Andreas A. Falkner, Gerhard Friedrich, Konstantin Schekotihin, Richard Taupe, and Erich Christian Teppan. Industrial applications of answer set programming. *KI*, 32(2-3):165–176, 2018.

[FIS11] Enrico Franconi, Yazmin Angélica Ibáñez-García, and Inanç Seylan. Query answering with DBoxes is hard. *Electr. Notes Theor. Comput. Sci.*, 278:71–84, 2011.

[Fra94] Enrico Franconi. Description logics for natural language processing. In *In: Working Notes of the AAAI Fall Symposium on "Knowledge Representation for Natural Language Processing in Implemented Systems",1994*, pages 37–44, 1994.

[GBFF91] Manfred Gehrke, Gerrit Burkert, Peter Forster, and Enrico Franconi. Natural language processing and description logics. In *In: Peltason, C., von Luck, K., Kindermann, C. (eds.) Proc. of the Terminological Logic Users Workshop,Department of Computer Science, Technische Universität Berlin (1991)*, pages 162–164, 1991.

[GHM+14] Birte Glimm, Ian Horrocks, Boris Motik, Giorgos Stoilos, and Zhe Wang. Hermit: An OWL 2 reasoner. *J. Autom. Reasoning*, 53(3):245–269, 2014.

[GHS08] Birte Glimm, Ian Horrocks, and Ulrike Sattler. Unions of conjunctive queries in SHOQ. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Eleventh International Conference, KR 2008, Sydney, Australia, September 16-19, 2008*, pages 252–262, 2008.

[GIKK15] Víctor Gutiérrez-Basulto, Yazmin Angélica Ibáñez-García, Roman Kontchakov, and Egor V. Kostylev. Queries with negation and inequalities over lightweight ontologies. *J. Web Sem.*, 35:184–202, 2015.

[GKK+11] Martin Gebser, Benjamin Kaufmann, Roland Kaminski, Max Ostrowski, Torsten Schaub, and Marius Schneider. Potassco: The potsdam answer set solving collection. *AI Comm.*, 24(2):107–124, 2011.

[GKL+14] Birte Glimm, Yevgeny Kazakov, Thorsten Liebig, Trung-Kien Tran, and Vincent Vialard. Abstraction refinement for ontology materialization. In *The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part II*, pages 180–195, 2014.

[GKT17] Birte Glimm, Yevgeny Kazakov, and Trung-Kien Tran. Ontology materialization by abstraction refinement in horn SHOIF. In *In AAAI 2017, USA*, 2017.

[GL88] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Proc. of ICLP/SLP 1988*, pages 1070–1080. MIT Press, 1988.

[GLHS08] Birte Glimm, Carsten Lutz, Ian Horrocks, and Ulrike Sattler. Conjunctive query answering for the description logic SHIQ. *J. Artif. Intell. Res.*, 31:157–204, 2008.

[Hay81] Patrick J. Hayes. The logic of frames. In Nils J. Nilsson Bonnie Lynn Webber, editor, *Readings in Artificial Intelligence*, pages 451–458. Morgan Kaufmann, 1981.

[HB11] Matthew Horridge and Sean Bechhofer. The OWL API: A java API for OWL ontologies. *Semantic Web*, 2(1):11–21, 2011.

[HEX10] Stijn Heymans, Thomas Eiter, and Guohui Xiao. Tractable reasoning with dl-programs over datalog-rewritable description logics. In *Proc. of ECAI 2010*. IOS Press, 2010.

[HKS+17] Dag Hovland, Roman Kontchakov, Martin Skjæveland, Ariid Waaler, and Michael Zakharyaschev. Ontology-based data access to slegge. In *ISWC*, 2017.

[HM05] Ullrich Hustadt and Boris Motik. Description logics and disjunctive datalog - the story so far. In *Proceedings of the 2005 International Workshop on Description Logics (DL2005), Edinburgh, Scotland, UK, July 26-28, 2005*, 2005.

[HMS04] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. A decomposition rule for decision procedures by resolution-based calculi. In *LPAR 2004, Uruguay*, 2004.

[HMS07] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Reasoning in description logics by a reduction to disjunctive datalog. *J. Autom. Reasoning*, 39(3):351–384, 2007.

[HRG96] Ian Horrocks, Alan L. Rector, and Carole A. Goble. A description logic based schema for the classification of medical data. In *Knowledge Representation Meets Databases, Proceedings of the 3rd Workshop KRDB'96, Budapest, Hungary, August 13, 1996*, 1996.

[IKL13] Vadim Ivanov, Matthias Knorr, and João Leite. A query tool for *EL* with non-monotonic rules. In *Proc. of ISWC 2013*, pages 216–231, 2013.

[KAH11] Matthias Knorr, José Júlio Alferes, and Pascal Hitzler. Local closed world reasoning with description logics under the well-founded semantics. *Artif. Intell.*, 175(9-10):1528–1554, 2011.

[Kaz09] Yevgeny Kazakov. Consequence-driven reasoning for Horn SHIQ ontologies. In *IJCAI*, pages 2040–2045, 2009.

[KRH07] Markus Krötzsch, Sebastian Rudolph, and Pascal Hitzler. Complexity boundaries for Horn description logics. In *AAAI*, pages 452–457. AAAI Press, 2007.

[KSH12] Markus Krötzsch, Frantisek Simancik, and Ian Horrocks. A description logic primer. *CoRR*, abs/1201.4089, 2012.

[Len02] Maurizio Lenzerini. Data integration: A theoretical perspective. In *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA*, pages 233–246, 2002.

[Len11] Maurizio Lenzerini. Ontology-based data management. In *Proceedings of the 20th ACM Conference on Information and Knowledge Management, CIKM 2011, Glasgow, United Kingdom, October 24-28, 2011*, pages 5–6, 2011.

[LMTV19] Nicola Leone, Marco Manna, Giorgio Terracina, and Pierfrancesco Veltri. Fast query answering over existential rules. *ACM Trans. Comput. Log.*, 20(2):12:1–12:48, 2019.

[LRXC15] Davide Lanti, Martin Rezk, Guohui Xiao, and Diego Calvanese. The NPD benchmark: Reality check for OBDA systems. In *Proc. of the 18th Int. Conf. on Extending Database Technology (EDBT 2015)*. ACM Press, 2015.

[LTW09] Carsten Lutz, David Toman, and Frank Wolter. Conjunctive query answering in the description logic EL using a relational database system. In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 2070–2075, 2009.

[Lut08] Carsten Lutz. The complexity of conjunctive query answering in expressive description logics. volume 5195 of *LNCS*, pages 179–193. Springer, 2008.

[LWW07] Carsten Lutz, Dirk Walther, and Frank Wolter. Conservative extensions in expressive description logics. In *IJCAI*, pages 453–458, 2007.

[McG99] Deborah L. McGuinness. Ontology-enhanced search for primary care medical literature. In *Proceedings of Int. Medical Informatics Association Working Group 6 – Conference on Natural Language Processing and Medical Concept Representation, IMIA 1999*, 1999.

[Min85] Marvin Minsky. A framework for representing knowledge. In *Readings in Knowledge Representation, USA.*, pages 245–262. Morgan Kaufmann, 1985.

[MNP+14] Boris Motik, Yavor Nenov, Robert Piro, Ian Horrocks, and Dan Olteanu. Parallel materialisation of datalog programs in centralised, main-memory RDF systems. In *Proceedings of the Twenty-Eighth AAAI Conference on*

Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada., pages 129–137, 2014.

[MR10] Boris Motik and Riccardo Rosati. Reconciling description logics and rules. *J. ACM*, 57(5), 2010.

[Neb90] Bernhard Nebel. Terminological reasoning is inherently intractable. *Artif. Intell.*, 43(2):235–249, 1990.

[OCE08] Magdalena Ortiz, Diego Calvanese, and Thomas Eiter. Data complexity of query answering in expressive description logics via tableaux. *J. Autom. Reasoning*, 41(1):61–98, 2008.

[ORS11] Magdalena Ortiz, Sebastian Rudolph, and Mantas Simkus. Query answering in the Horn fragments of the description logics SHOIQ and SROIQ. In *IJCAI 2011, Spain*, 2011.

[OWL09] OWL working group. OWL Web Ontology Language semantics: document overview – W3C recommendation. Technical report, World Wide Web Consortium, October 2009. Available at `http://www.w3.org/TR/owl2-overview/`.

[Pap94] Christos H. Papadimitriou. *Computational complexity.* Addison-Wesley, 1994.

[PLC+08] 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.

[PMH10] Héctor Pérez-Urbina, Boris Motik, and Ian Horrocks. Tractable query answering and rewriting under description logic constraints. *J. Applied Logic*, 8(2):186–209, 2010.

[Qui67] M. Ross Quillan. A theory and simulation of some basic capabilities. In *Behavioral Science*, pages 410–430. Morgan Kaufmann, 1967.

[RA10] Riccardo Rosati and Alessandro Almatelli. Improving query answering over dl-lite ontologies. In *KR 2010, Canada.* AAAI Press, 2010.

[RBG+97] Alan L. Rector, Sean Bechhofer, Carole A. Goble, Ian Horrocks, W. A. Nowlan, and W. D. Solomon. The GRAIL concept modelling language for medical terminology. *Artificial Intelligence in Medicine*, 9(2):139–171, 1997.

[Red17] Christoph Redl. Efficient evaluation of answer set programs with external sources based on external source inlining. In *Proc. of AAAI 2017.* AAAI Press, February 2017.

[RKZ13] Mariano Rodriguez-Muro, Roman Kontchakov, and Michael Zakharyaschev. Ontology-based data access: Ontop of databases. volume 8218 of *LNCS*, pages 558–573. Springer, 2013.

[Ros05] Riccardo Rosati. On the decidability and complexity of integrating ontologies and rules. *J. Web Sem.*, 3(1):61–73, 2005.

[Ros06] Riccardo Rosati. DL+log: Tight integration of description logics and disjunctive datalog. In *Proc. of KR 2006*, 2006.

[RPZ10] Yuan Ren, Jeff Z. Pan, and Yuting Zhao. Soundness preserving approximation for tbox reasoning. In *AAAI*, 2010.

[RR15] Mariano Rodriguez-Muro and Martín Rezk. Efficient sparql-to-sql with R2RML mappings. *J. Web Semant.*, 33:141–169, 2015.

[Rud11] Sebastian Rudolph. Foundations of description logics. In *Reasoning Web. Semantic Technologies for the Web of Data - 7th International Summer School 2011, Galway, Ireland, August 23-27, 2011, Tutorial Lectures*, pages 76–136, 2011.

[SC10] L. Shi and X. Cai. An exact fast algorithm for minimum hitting set. In *2010 Third International Joint Conference on Computational Science and Optimization*, volume 1, pages 64–67, 2010.

[Sch93] Andrea Schaerf. On the complexity of the instance checking problem in concept languages with existential quantification. *J. Intell. Inf. Syst.*, 2(3):265–278, 1993.

[Sim13] Frantisek Simancik. *Consequence-Based Reasoning for Ontology Classification*. PhD dissertation, University of Oxford, 2013.

[SKH11] František Simančík, Yevgeny Kazakov, and Ian Horrocks. Consequence-based reasoning beyond horn ontologies. In *Proc. of IJCAI 2011*, pages 1093–1098. AAAI Press, 2011.

[SLG14] Andreas Steigmiller, Thorsten Liebig, and Birte Glimm. Konclude: System description. *J. Web Sem.*, 27:78–85, 2014.

[SM15] Giorgio Stefanoni and Boris Motik. Answering conjunctive queries over EL knowledge bases with transitive and reflexive roles. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pages 1611–1617, 2015.

[SMKR14] Giorgio Stefanoni, Boris Motik, Markus Krötzsch, and Sebastian Rudolph. The complexity of answering conjunctive and navigational queries over OWL 2 EL knowledge bases. *J. Artif. Intell. Res. (JAIR)*, 51:645–705, 2014.

[SPG+07] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL reasoner. *J. Web Sem.*, 5(2):51–53, 2007.

[Swi04] Terrance Swift. Deduction in ontologies via ASP. In *Proc. of LPNMR 2004*. Springer, 2004.

[TSCS15] Despoina Trivela, Giorgos Stoilos, Alexandros Chortaras, and Giorgos B. Stamou. Query rewriting in horn-shiq. In *Proceedings of the 28th International Workshop on Description Logics, Athens,Greece, June 7-10, 2015.*, 2015.

[Var82] Moshe Y. Vardi. The complexity of relational query languages (extended abstract). In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 137–146, 1982.

[WM12] Sebastian Wandelt and Ralf Möller. Towards abox modularization of semi-expressive description logics. *Applied Ontology*, 7(2):133–167, 2012.

[XCK+18] Guohui Xiao, Diego Calvanese, Roman Kontchakov, Domenico Lembo, Antonella Poggi, Riccardo Rosati, and Michael Zakharyaschev. Ontology-based data access: A survey. In *IJCAI*, pages 5511–5519, 2018.

[XDCC19] Guohui Xiao, Linfang Ding, Benjamin Cogrel, and Diego Calvanese. Virtual knowledge graphs: An overview of systems and use cases. *Data Intelligence*, 1:201–223, 2019.