TU WIEN Informatics

Reasoning in Very Expressive Description Logics with Varying Information Completeness

DISSERTATION

zur Erlangung des akademischen Grades

Doktorin der Technischen Wissenschaften

eingereicht von

DI Sanja Lukumbuzya, B.Sc
Matrikelnummer 01227514

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Assistant Prof. Dr.techn. Mantas Šimkus
Zweitbetreuung: Associate Prof. Dr.in techn. Magdalena Ortiz

Diese Dissertation haben begutachtet:

Frank Wolter

Filip Murlak

Wien, 3. Juni 2024

Sanja Lukumbuzya

TU WIEN Informatics

Reasoning in Very Expressive Description Logics with Varying Information Completeness

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktorin der Technischen Wissenschaften

by

DI Sanja Lukumbuzya, B.Sc
Registration Number 01227514

to the Faculty of Informatics

at the TU Wien

Advisor: Assistant Prof. Dr.techn. Mantas Šimkus
Second advisor: Associate Prof. Dr.in techn. Magdalena Ortiz

The dissertation has been reviewed by:

Frank Wolter

Filip Murlak

Vienna, June 3, 2024

Sanja Lukumbuzya

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

Erklärung zur Verfassung der Arbeit

DI Sanja Lukumbuzya, B.Sc

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, 3. Juni 2024

Sanja Lukumbuzya

Acknowledgements

It takes a village to complete a PhD, and I am eternally grateful to mine.

I have nothing but words of praise for my advisors, Mantas Šimkus and Magdalena Ortiz. Thank you for being excellent teachers and role models. Without your gentle mentorship filled with encouragement, patience, and understanding, completing this PhD would not have been possible. I do not think I can appropriately express my gratitude for everything you have done for me, but I am going to try.

Thank you for passing your excitement for research and knowledge on to me.
Thank you for your guidance.
Thank you for the countless hours of discussions — both the fruitful ones and the ones where we got lost on tangents.
Thank you for continuing to believe in me even when I did not believe in myself.
Most of all, thank you for always having my back.

Academia needs more people like you.

I would also like to extend my sincere thanks to Frank Wolter and Filip Murlak for investing their time and effort into reviewing this thesis and for doing it on a tight schedule. Thank you!

Furthermore, I wish to thank Meghyn Bienvenu for being an amazing host during my research stay — I look back on the time I spent in Bordeaux fondly and I hope we will have more opportunities for collaboration in the future. Many thanks also to Diego Calvanese and Marco Montali for all the discussions in the spring of 2021 that landed my name on the list of people working in Business Process Management.

On this journey, I have come across many wonderful people who have contributed to the materialization of this thesis in their own way. In particular, I am grateful to:

- My first DK friends and my go-to crowd for everything, Rafael Kiesel and Anna Rapberger, for dancing to Severina's *Gas, Gas* and for the amazing ideas that never made it out of the group chat.
- Adrián Rebola Pardo for lending his ear to my wailing.

- Ana Costa for our (in)compatible office schedules, Carl Ludwig coffee dates, as well as for providing a much-needed distraction during the final preparations for my defense.
- Soeren Terziadis (né Nickel) for approaching me in that Formal Language Theory course and all that followed.
- My dear *amios*, Federica Di Stefano, Giovanni Buraglio, and Pamina Georgiou for giving me a reason not to work from home as well as for all the yard time spent together.
- Anouk Oudshoorn for making long nights at the office feel short and for turning everything into a karaoke session.
- Tommaso Mannelli Mazzoli and Ida Gjergji for tolerating said karaoke sessions.
- Emily Yu and Nils Froleyks for sharing the highs and the lows of parenting with me.
- Kees van Berkel for music therapy.
- Davide Longo for his efforts to make our lunch breaks more elegant.
- Quentin Manière for *Hanabi* and wine, and, most of all, for being oh so patient with me.
- Maya Olszewski for flying across Europe just to sleep in my bunk bed in Bordeaux.
- Augusto Blaas Corrêa for his shopping assistance and photography skills.
- Our heroines without capes, Beatrix Buhl, Eva Nedoma, and Juliane Auerböck, for never rolling their eyes at my "quick questions".
- Anna Prianichnikova for making the LogiCS DK feel like a second family.
- The *Italian Camouflage* playlist curators.
- Monday Cake Meeting crew for letting me eat cake.
- Carl Ludwig crew for providing coffee and emotional support to go with it.
- All other fantastic people at TU Wien and beyond that I did not mention by name but whose presence made a positive mark on my PhD experience.

A special thank you goes also to all my friends outside of work for their unwavering support despite not knowing what it is exactly that I do. In particular, I would like to thank my oldest friends, Milica Petrović (née Milošević) and Teodora Živković, for way too many things to mention here, but most of all for sticking around all these years. I am also grateful to Nađa Jelica for showing me that being neighbors is a matter of mindset,

and to my spontaneous mom crew, Marija Pajić, Harriet Szanto, and Manu Parisse, for being brave enough to drop the spreadsheets.

None of this would have been possible without the support of my immediate and extended family. I am grateful to my parents, Milena and Dejan Pavlović, for believing in me from day one and for allowing me to chase my dreams. I appreciate everything you have done and continue to do for me. I am also grateful to my sister and role model, Mila Pavlović, for her quiet but constant support as well as for providing me with her *Lami Vibing* playlist on Spotify that carried me to the finish line. And to the whole Lukumbuzya-Pomaroli family, I owe a million thanks for welcoming me with open arms and always making me feel like I am one of their own. I am very lucky to have you guys.

I cannot even begin to express my gratitude to my husband, Michael, for being my rock throughout this entire journey. Your love, support, and understanding have been both the wind beneath my sails and my anchor when I needed grounding. Thank you for taking care of our young family and for teaching me what a true partnership looks like. Your sacrifices and encouragement have made this achievement possible, and I am profoundly grateful for everything you have done. This accomplishment is as much yours as it is mine.

Finally, my sweet Oliver, thank you for pulling the emergency brakes and forcing me to slow down. In the past few years, you have taught me more about myself than I ever thought was possible. I hope I am making you proud.

This research was supported by the Austrian Science Fund (FWF) projects P30873 and W1255. I would like to thank the funding body for the financial support.

Kurzfassung

Seit ihrer Entstehung Ende der 2000er Jahre hat das Datenverwaltungsparadigma namens *Ontologie-basierte Datenzugriff* (*OBDA*) viel Aufmerksamkeit von der wissenschaftlichen Gemeinschaft auf sich gezogen. Das Ziel von OBDA ist es, nicht sachkundigen Nutzern die einfache Abfrage mehrerer heterogener und potenziell unvollständiger Datenquellen zu ermöglichen. Bei diesem Ansatz wird die Struktur der zugrunde liegenden Daten vor den Nutzer verborgen. Stattdessen wird Ihnen eine *Ontologie* präsentiert, die einen konzeptionellen Überblick des Anwendungsbereichs definiert und Hintergrundwissen darüber in Form einer logischen Theorie bereitstellt. Die Nutzer können dann das Vokabular der Ontologie verwenden um Abfragen zu formulieren, die, basierend auf Daten die aus verschiedenen Quellen integriert wurden, beantwortet und mit Fakten, welche mithilfe des verfügbaren Wissens abgeleitet werden können, ergänzt werden. Solche Abfragen werden als *Ontologie-vermittelte Abfragen (OMQs)* bezeichnet, und ihre Beantwortung ist eine der zentralen Aufgaben von OBDA.

Beschreibungslogiken (DLs) sind zweifellos eine der beliebtesten Familien von Formalismen, die für die Beschreibung von Ontologien verwendet werden. In DLs modellieren wir das relevante Anwendungsgebiet mit *Individuen* (=*Konstanten*), sowie einstelligen und zweistelligen Prädikatsymbolen, die *Konzeptnamen* und *Rollenamen* genannt werden. DL-Wissensbasen bestehen aus einer *TBox*, die *terminologische Axiome* enthält, welche Beziehungen zwischen Konzepten und Rollen angeben, und einer *ABox*, die *Fakten* enthält, die die Beteiligung bestimmter Individuen an Konzepten/Rollen definieren. Unterschiedliche DLs unterscheiden sich in Bezug auf verfügbare Konstruktoren zum Aufbau komplexer Konzepte/Rollen sowie die Formen der terminologischen Axiome, die sie zulassen. Unabhängig davon können die meisten DLs als entscheidbare Fragmente der Prädikatenlogik erachtet werden und verwenden daher die *Annahme der offenen Welt (OWA)*. Intuitiv besagt die OWA, dass alles, was nicht verboten ist, möglich ist, und sie ist im Allgemeinen angemessen, wenn Daten als unvollständig betrachtet werden. Wenn jedoch ein bestimmter Teil der Daten vollständig ist, ist die Verwendung von *Annahme der geschlossenen Welt (CWA)* angebrachter, welche besagt, dass das, was nicht als wahr bekannt ist, falsch sein muss. Echtweltanwendungen erfordern oft die Interaktion unvollständiger und vollständiger Teile der Daten. Als Konsequenz wurden verschiedene Ansätze zur Kombination von OWA und CWA in DLs vorgeschlagen.

Ein besonders herausragender Ansatz beinhaltet die Erweiterung von DLs um die Möglich-

keit, anzugeben, welcher Teil der Signatur als vollständig anzusehen ist. Diese Prädikate werden als *geschlossen* bezeichnet und unterliegen der CWA. In dieser Arbeit konzentrieren wir uns auf sehr ausdrucksstarke Beschreibungslogiken mit geschlossenen Prädikaten und untersuchen ihre algorithmischen und modelltheoretischen Eigenschaften. Insbesondere präsentieren wir neue scharfe Schanken der Datenkomplexität für einige Standard-Reasoning-Aufgaben in ausdrucksstarken DLs mit geschlossenen Prädikaten, die gleichzeitig Nominalformen, Inverse und Zahlenbeschränkungen unterstützen – eine Kombination von Konstruktoren, die bekanntermaßen auch ohne geschlossene Prädikate algorithmische Probleme verursacht. Speziell zeigen wir, dass das Problem der Entscheidung über die Konsistenz von Wissensbasen, die in der DL $\mathcal{ALCHOIQ}$ formuliert sind, in der Datenkomplexität in Anwesenheit geschlossener Prädikate NP-vollständig bleibt. Dies ist ein positives Ergebnis, da es zeigt, dass geschlossene Prädikate keinen Anstieg der Komplexität von grundlegenden Reasoning-Aufgaben in ausdrucksstarken DLs verursachen. Anschließend wenden wir uns OMQs zu, deren Ontologiekomponente geschlossene Prädikate enthält und in einer für uns relevanten DL beschrieben ist, und untersuchen ihre Expressivität auf zwei unterschiedliche Arten. Die erste betrifft die *relative Expressivität* dieser OMQ-Sprachen im Vergleich zu Standard-Abfragesprachen wie Datalog und seinen Erweiterungen. Insbesondere präsentieren wir Polynomialzeitumformung einer großen Klasse von durch $\mathcal{ALCHOIQ}$-Ontologien mit geschlossenen Prädikaten repräsentierten Abfragen nach Datalog mit Negation unter der stabilen Modellsemantik. Ein wichtiges Nebenprodukt unserer Umformung ist das die Beantwortung von Abfragen für die betrachtete Klasse von OMQs co-NP-vollständig in der Datenkomplexität ist. Die zweite Richtung untersucht die expressive Leistungsfähigkeit dieser OMQ-Sprachen aus dem Blickwinkel der *deskriptiven Komplexität*, wobei die zentrale Frage ist, ob eine gegebene OMQ-Sprache ausdrucksstark genug ist, um *alle* Abfragen auszudrücken, die innerhalb einer bestimmten Zeit- oder Speicherbeschränkung berechnet werden können. Unsere Ergebnisse zeigen, dass die OMQ-Sprache, die atomare Abfragen mit Ontologien in der sehr ausdrucksstarken DL $\mathcal{ALCHOI}$ mit geschlossenen Prädikaten kombiniert, nicht alle co-NP-berechenbaren booleschen Abfragen ausdrücken kann, obwohl sie in der Datenkomplexität co-NP-vollständig ist. Nach diesem negativen Ergebnis schlagen wir eine Erweiterung der betrachteten OMQ-Sprache vor und zeigen, dass sie tatsächlich ausdrucksstark genug ist, um exakt die Klasse aller booleschen Abfragen zu erfassen, die in co-NP berechenbar sind. Schließlich stellen wir fest, dass die Deklaration einiger Prädikate als geschlossen einen interessanten Effekt auf die Modelle der Wissensbasis haben kann. Es kommt gelegentlich vor, dass die Interaktion zwischen geschlossenen Prädikaten und den numerischen Beschränkungen in der TBox dazu führt, dass bestimmte offene Prädikate nur Erweiterungen begrenzter Größe haben Wir präsentieren einen Algorithmus zur Identifizierung solcher Prädikate und geben eine allgemeine, im Schlimmstfall optimale Schranke für die Größe ihrer Erweiterungen an. Unsere Ergebnisse bieten ein vielversprechendes Werkzeug zur Unterstützung des Aufbaus hochwertiger Ontologien sowie einen neuen Ansatz, um die Entscheidbarkeitsgrenzen anspruchsvoller Datenverwaltungsaufgaben voranzutreiben.

Als nächstes wenden wir uns einem anderen Ansatz zur Berücksichtigung der teilweisen

CWA in DLs zu, der auf der Kombination von Ontologien und nicht-monotonen Regeln in sogenannten *hybriden Wissensbasen* basiert. In diesem Zusammenhang stellen wir einen neuen hybriden Formalismus namens *Resilient Logic Programs (RLPs)* vor, der erststellige Ontologien und nicht-monotone Regeln kombiniert, um Unterstützung für Systeme zu bieten, die *resilient* sind, d. h. in allen möglichen Situationen korrekt reagieren. In dieser Einstellung beschreibt die Ontologie alle möglichen Situationen, denen das System begegnen könnte. Die Regeln werden verwendet, um auf diese Situationen zu reagieren, und die Aufgabe, zu entscheiden, ob das System immer korrekt reagiert, kann auf die Überprüfung der Konsistenz solcher hybriden Wissensbasen reduziert werden. Unsere Ergebnisse zeigen, dass RLPs unter ein gewissen sinnvollen Annahmen entscheidbar und sehr ausdrucksstark sind: Sie können elegant $\exists\forall\exists$-QBFs, disjunktive Datalog und Konfigurationsprobleme unter Unvollständigkeit von Informationen ausdrücken. Dann identifizieren wir eine große Klasse von RLPs, die in disjunktiven Datalog umgeformt werden können, und präsentieren mehrere Komplexitätsergebnisse für RLPs, deren Ontologie in einigen Standard-Dls beschrieben ist.

Abstract

Since its inception in the late 2000s, the data management paradigm termed *ontology-based data access* (*OBDA*) has received a lot of attention from the scientific community. The goal of OBDA is to allow non-expert users to query multiple heterogeneous and possibly incomplete data sources in an easy way. In this approach, the structure of the underlying data is hidden from the users. Instead, they are presented with an *ontology* that defines a high-level conceptual view of the application domain and provides background knowledge about it in the form of a logical theory. The users can then write queries using the vocabulary of the ontology that are answered over the data integrated from different sources and supplemented with the facts that can be inferred using the available knowledge. Such queries are called *ontology-mediated queries (OMQs)* and answering them is one of the central tasks of OBDA.

Description logics (DLs) are undoubtedly one of the most popular families of formalisms used for expressing ontologies. In DLs, we model the domain of interest using *individuals* (=*constants*), and unary and binary predicate symbols, called *concept names* and *role names*, respectively. DL knowledge bases consist of a *TBox* containing *terminological axioms* that specify relationships between concepts and roles, and an *ABox* containing *facts* that assert participation of certain individuals in concepts/roles. Individual DLs differ in terms of available constructors for building complex concepts/roles as well as the shapes of terminological axioms that they allow. Regardless, most DLs can be seen as decidable fragments of first-order logic and as such they employ the *open-world assumption (OWA)*. Intuitively, the OWA states that everything that is not forbidden is possible and it is generally appropriate when data is considered incomplete. However, if we know that a certain part of data is complete, then it is more appropriate to employ the *closed-world assumption (CWA)* which states that what is not known to be true must be false. Real-world applications often require that incomplete and complete parts of data interact. As a result, various approaches have been proposed for combining OWA and CWA in DLs.

One particularly prominent approach involves enriching DLs with the possibility of specifying which part of the signature is considered complete. These predicates are called *closed* and are to be interpreted under the CWA. In this thesis, we focus on very expressive description logics with closed predicates and we study their computational and model-theoretic properties. In particular, we provide new tight data complexity bounds of some

standard reasoning tasks in expressive DLs with closed predicates that simultaneously support nominals, inverses, and number restrictions – a combination of constructors that is known to cause computational issues even without closed predicates. More precisely, we show that the problem of deciding the consistency of knowledge bases formulated in the DL $\mathcal{ALCHOIQ}$ remains NP-complete in data complexity in the presence of closed predicates. This is a positive result, as it shows that closed predicates do not cause an increase in the computational complexity of basic reasoning tasks in expressive DLs. We then turn to OMQs whose ontology component contains closed predicates and is expressed in one of our DLs of interest and we investigate their expressive power following two different directions. The first one is concerned with the *relative expressiveness* of these OMQ languages compared to standard query languages like Datalog and its extensions. In particular, we present a polynomial-time rewriting of a large class of queries mediated by $\mathcal{ALCHOIQ}$ ontologies with closed predicates into Datalog with negation under the stable model semantics. As an important by-product of our translation, we get that the query answering problem is co-NP-complete in data complexity for the considered class of OMQs. The other direction investigates the expressive power of these OMQ languages from the *descriptive complexity* perspective, where the central question is to understand whether a given OMQ language is powerful enough to express *all* queries that can be computed within some bound on time or space. Our results show that the OMQ language that pairs atomic queries with ontologies in the very expressive DL $\mathcal{ALCHOI}$ with closed predicates cannot express all co-NP-computable Boolean queries, despite being co-NP-complete in data complexity. Following this negative result, we propose an extension of the considered OMQ language and prove that it is indeed expressive enough to precisely capture the class of all Boolean queries computable in co-NP. Finally, we observe that declaring some predicates to be closed can have an interesting effect on the knowledge base models. Namely, it sometimes happens that the interaction between closed predicates and the numeric constraints in the TBox results in certain open predicates having only extensions of bounded size. We present an algorithm to identify such predicates and we give a general worst-case optimal bound for the size of their extensions. Our results yield a promising tool for supporting the construction of high-quality ontologies as well as a new way to push the decidability frontiers of challenging data management tasks.

We next turn our attention to another approach for accommodating partial CWA in DLs that is based on combining ontologies and non-monotonic rules into so-called *hybrid knowledge bases*. Along these lines, we introduce a new hybrid formalism called *Resilient Logic Programs (RLPs)*, coupling first-order ontologies and non-monotonic rules with the goal of providing reasoning support for systems that should be *resilient*, i.e., react correctly in all possible situations. In this setting, the ontology describes all possible situations that the system might face and the rules are used to react to these situations. The task of deciding whether the system always has a correct reaction can then be reduced to checking the consistency of such hybrid knowledge bases. Our results show that, under a couple of natural assumptions, RLPs are decidable and very expressive: they can elegantly express $\exists\forall\exists$-QBFs, disjunctive Datalog, and configuration problems

under incompleteness of information. We then go on to identify a large class of RLPs that can be rewritten into disjunctive Datalog and we present several complexity results for RLPs whose ontology is written in some standard DLs.

Contents

CHAPTER 1

Introduction

It is often said that data is the gold of the digital age, and yet, it is worthless without ways to convert it into knowledge and draw new conclusions from it. A traditional approach to data management goes back to the early 1970s and the introduction of the *relational database model* by Codd [Cod70], which has since then become a de facto standard for managing and querying data in many domains. In this model, data is organized into tables, also known as relations, where each row in some table represents a data record and each column represents a specific attribute. Within a specific table, a single data record is uniquely identified by its primary key, i.e., a set of attributes whose combination of values is unique for each data record in a given table. The relationships between data stored in different tables are then established via foreign keys, i.e., by storing references to the primary key attributes of another table. However, despite their popularity, there are several challenges that relational database management systems (RDBMS) struggle with. First, they are very rigid in structure, as they require a predefined relational schema which is difficult to adapt in case the data does not conform to it. This rigidity also makes the integration of data coming from relational databases with different underlying schemata a challenging task. RDBMS offer very limited support for representing complex semantic relationships which often results in a large amount of attributes and auxiliary tables. This often makes querying relational databases a rather cumbersome task as all of the burden is placed on the user. In order to write a meaningful query, the user must be familiar with the structure of the database, the names of the attributes, and how individual tables relate to each other and then build a query that logically encodes all this information. Finally, RDBMS are not suitable for dealing with incomplete data, i.e., with situations where we want to infer and make conclusions about the existence and properties of some data record that is not explicitly stored in the database.

One way of mitigating these issues is through the use of *ontologies* – formal descriptions of the domain of interest that detail the relevant concepts in this domain, their characteristics, and relationships that hold between them. In this context, an ontology defines a

vocabulary and provides background knowledge about the application domain, thus adding meaning back to the data which can further be exploited for integration and completion purposes. Ontologies are at the heart of the data management paradigm termed *ontology-based data access (OBDA)* [PLC+08]. The central idea of OBDA is to allow non-expert users to query multiple heterogeneous and possibly incomplete data sources in an easy way by hiding from the users the actual structure of underlying data sources. Instead, users are presented with an ontology that defines a high-level conceptual view of the application domain that should ideally match their intuitions about this domain. All interactions with the data pass through the ontology. Namely, users can write queries over the vocabulary of the ontology, essentially querying the conceptual view, which is often an easier task than formulating a query over multiple data sources with potentially different structures. For example, it was observed in [KHJR+15] that the adoption of OBDA significantly reduced the amount of time spent on information gathering, as non-expert users were suddenly able to independently formulate queries due to their user-oriented vocabulary. Query answering in OBDA works by mapping the data from different sources to the concepts and relationships in the ontology and thus defining a unified view of the data that uses only the vocabulary of the ontology. User queries are then answered over this data view enriched with the facts that can be inferred from it using the background knowledge stored in the ontology, therefore providing higher-quality answers.

Description logics (DLs)[BHLS17] are a prominent family of formalisms specifically tailored for expressing structured knowledge. Backed up by years of active research that resulted in efficient reasoners along with a relatively good understanding of the computational properties of these languages, DLs are often the number one choice for building and reasoning about ontologies. For instance, the Web Ontology Language (OWL) family proposed by W3C for defining ontologies on the Web [Hor08, OWL09] is based on description logics. In DLs, the domain of interest is represented via *individuals* (i.e., constants) and unary and binary predicate symbols, referred to as *concept names* and *role names*, respectively. In the standard setting, a DL knowledge base consists of two components: (i) a *TBox* (i.e., an ontology) that contains terminological axioms detailing the relationships that hold between different individuals, concepts, and roles, and (ii) an *ABox* (i.e., data) that contains factual assertions about the participation of specific individuals in concepts/roles. As already mentioned, the term *description logics* refers to a whole family of logics, where individual logics differ in terms of their expressiveness (i.e., what types of axioms are permitted by the logic) and computational costs of reasoning. DLs are generally separated into two broad categories: *lightweight* and *expressive* DLs. Lightweight DLs, like those belonging to the DL-Lite [CDL+07] and $\mathcal{EL}$ [BBL05] families, prioritize computational tractability and scalability over expressiveness, making them suitable for applications requiring efficient reasoning over large knowledge bases. In contrast to the lightweight DLs, expressive DLs offer fine-grained modeling possibilities but they are almost always intractable. A wide variety of reasoning tasks has been considered in the DL literature, including answering database queries over DL knowledge bases, which is also one of the key reasoning tasks of OBDA. In this setting, we deal with

so-called *ontology-mediated queries (OMQs)* that couple an ontology $\mathcal{T}$ expressed as a TBox in some DL and a database query q. Given an ABox $\mathcal{A}$ containing concrete data, we are interested in computing the *certain answers* to $(\mathcal{T}, q)$ over $\mathcal{A}$, which are defined as tuples of individuals that are answers to q over every relational structure that extends $\mathcal{A}$ and satisfies the ontological axioms in $\mathcal{T}$.

It is important to mention that query answering in DLs is more involved than simple query evaluation done by traditional database management systems – in fact, it amounts to logical reasoning. The reason behind this is different underlying assumptions about the completeness of data. More specifically, traditional RBDMS make the so-called *closed-world assumption (CWA)*, which intuitively says that only the facts that are stored in the database are considered true, while everything else is assumed false. In other words, we assume that our data is complete and all relevant information is present in the database. On the other hand, the OBDA approach in general, and DLs in particular, were designed to deal with potential information incompleteness. As such, they employ the *open-world assumption (OWA)*, stating that things that are not known to be true are not considered false, but rather unknown. As a result, when answering OMQs we have to consider different possible completions of the data, explaining the need for a notion like certain answers.

However, things are rarely binary, and real-world data can often contain some parts that come from trusted sources and are known to be complete (for example, flight and train schedules, or a list of courses offered by a university) as well as other parts that are considered incomplete. Moreover, incomplete and complete parts may be required to interact with each other, in which case both CWA and OWA on their own are inadequate and we need formalisms that offer flexible support for both. As a result, there has been a lot of research effort to extend description logics with some form of partial closed world assumption, see, e.g., [DNR02, EIL$^+$08, BLW09, SKH11]. In this thesis, we consider two such prominent approaches: (i) extending DLs with *closed predicates* and (ii) combining DLs with *non-monotonic rules*.

The basic idea behind the first approach is to equip DLs with the possibility of specifying which part of the signature should be interpreted under the closed-world assumption – these are the so-called *closed predicates* [SFdB09, LSW13]. More specifically, a DL knowledge base with closed predicates, in addition to the TBox $\mathcal{T}$ and the ABox $\mathcal{A}$, also contains a set Σ of predicates that are interpreted exactly as given in $\mathcal{A}$ and all axioms of $\mathcal{T}$ that would typically allow us to infer new information over the closed predicates are viewed as integrity constraints. Naturally, closed predicates affect reasoning – they make ontology-mediated query languages non-monotonic and they have also been shown to increase the complexity of basic reasoning tasks in some cases. In the first part of this thesis, we aim to investigate the effects that closed predicates have on reasoning in very expressive description logics in terms of their computational and model-theoretic properties. In particular, we aim to characterize data complexity and expressiveness of ontology-mediated query languages based on very expressive description logics that simultaneously support closed predicates, nominals, inverse roles, and qualified

number restrictions. When it comes to expressiveness, we consider two directions: (i) *relative expressiveness* that compares the expressive power of considered OMQ languages to traditional database query languages, in our case a non-monotonic extension of Datalog [GL88], and (ii) expressiveness from a *descriptive complexity* perspective that investigates whether some OMQ language is powerful enough to express all queries of some complexity class. Moving on, we make an interesting observation that closed predicates can sometimes interact with the axioms in the TBox in a way that also bounds the size of extensions of open predicates in the models of a given DL knowledge base. In this work, we are interested in developing a procedure that can decide when this is the case and provide suitable upper bounds on the sizes of such predicates.

We remark that our choice of logic makes things rather challenging. Indeed, the combination of nominals, inverse roles, and number restrictions is known to be quite tricky, even if the integers occurring in the TBox are at most one [GKL11]. This is because their interaction can result in the creation of infinitely many domain elements that are anonymous (i.e., they do not represent individuals occurring in the knowledge base), but nonetheless uniquely identifiable in any model of the considered knowledge base. Unfortunately, this almost completely destroys the forest model property which is heavily exploited in decision procedures for less expressive DLs. As a result, we have that some of the basic reasoning problems, like ontology satisfiability, are NExpTime-hard for such logics [Tob00]. For comparison, this problem is in ExpTime if any of the three constructors is omitted.

The second approach for combining OWA and CWA that we consider in this thesis are combinations of description logics and non-monotonic logic rules into so-called *hybrid knowledge bases* [Ros05]. Within this context, we develop a new hybrid formalism that also couples a first-order theory (or DL ontology) and a non-monotonic program. However, unlike other approaches in the literature where the interaction between the program and the theory is limited to consistency or query entailment tests, in our formalism the answer sets of the program must be 'resilient' to the models of the theory, allowing the program to respond differently to different models. This combination yields a very powerful language that can elegantly express $\exists\forall\exists$-QBFs, disjunctive ASP, and configuration problems under incompleteness of information.

Summing up, the high-level goal of this thesis is to better understand the effects that the partial CWA has on reasoning in expressive DLs as well as how it can be exploited for modeling some common practical problems. To this end, we identify the following concrete goals:

- **Characterize data complexity of reasoning in expressive DLs with closed predicates**
- **Characterize relative expressiveness of OMQs in the presence of closed predicates compared to classical query languages**

- **Investigate expressiveness of OMQs with closed predicates from the descriptive complexity perspective**
- **Understand interactions between closed predicates and number restrictions**
- **Develop new formalisms based DLs with closed predicates**

1.1 State of the Art

We next review the state of the art related to the goals we listed above and we point out some challenges.

1.1.1 Mixing OWA and CWA in DLs

Closed predicates. Allowing some predicates to be *closed* is a prominent way of supporting partial closed-world assumption in DLs. One of the earliest such approaches proposed replacing ABoxes with *DBoxes* [SFdB09]. Syntactically, both ABoxes and DBoxes are sets of assertions, however, there is a semantic difference between the two: a knowledge base with a DBox $\mathcal{D}$ requires its models to interpret every concept and role name occurring in $\mathcal{D}$ exactly as given in $\mathcal{D}$. These predicates are thus considered closed – their extensions are fixed and must be the same in every model of the knowledge base. In contrast, the rest of the predicates are open and their extensions might vary among interpretations. A generalization of this approach was presented in [LSW13] which does not interpret all predicates in the DBox in this way but allows the user to explicitly specify a subset of the signature that is considered closed. This gave rise to *description logics with closed predicates*.

Naturally, closed predicates have an effect on reasoning. Recall that in ontology-mediated query answering (OMQA) we are usually concerned with certain answers, i.e., tuples of individuals that are answers to the query in *every* model of the knowledge base. However, in the presence of closed predicates, we are no longer interested in arbitrary models but only in those that "obey" the closed predicates by interpreting them according to the provided data. This may alter our set of certain answers and introduce non-monotonicity.

The computational complexity of standard OMQA is very well understood. There is a wide range of complexity results in the literature that cover many different DLs and query languages using various techniques. For lightweight DLs like the members of the DL-Lite and $\mathcal{EL}$ families, answering conjunctive queries (CQs) consisting of existentially quantified conjunctions of first-order atoms is tractable in data complexity but not in combined complexity (see e.g. a recent survey [BO15]). For expressive DLs that extend $\mathcal{ALC}$ answering of CQs is usually coNP-complete in terms of data-complexity, which follows from the data complexity of the very expressive fragment of first-order logic that subsumes most of the expressive DLs [PH09]. Furthermore, the same task is 2ExpTime-complete in combined complexity for some extensions of $\mathcal{ALC}$ [Lut08, ELOŠ09, NOŠ16].

Regarding the complexity of query answering in the presence of closed predicates, the picture is not as detailed but some initial work has been done. For example, [FIS11] shows that closed predicates make query answering in DL-Lite$_\mathcal{F}$ intractable in data complexity, namely, the data complexity is increased from AC0 to coNP-completeness. This was expanded upon in [LSW13] where a non-uniform analysis of the data complexity of answering CQs mediated by certain members of the DL-Lite and $\mathcal{EL}$ families extended with closed predicates was performed. The goal of this work was to separate individual ontologies into tractable and intractable cases, thereby obtaining a dichotomy for the data complexity of CQ answering between AC0 and coNP for DL-Lite ontologies and between PTime and coNP for $\mathcal{EL}$. Furthermore, some results on the combined complexity of query answering can be found in [NOŠ16]. However, these works mainly focus on lightweight DLs.

In the case of expressive DLs with closed predicates, it was observed in [SFdB09] that closed predicates can be simulated with the help of nominals in DLs that support disjunction. As a result, the combined complexity of answering OMQs in these logics extended with closed predicates remains unaffected. As this involves encoding the ABox into the TBox, the same strategy is not applicable for obtaining data complexity bound of the same task. It was shown in [LSW19] that the data complexity of answering CQs mediated by the expressive DL $\mathcal{ALCHI}$ with closed predicates is coNP-complete. Furthermore, the recent result in [AOŠ20] shows that answering queries consisting of a single first-order atom (also called *instance queries, or IQs for short*) and a certain fragment of CQs in the expressive DL $\mathcal{ALCHIO}$ with closed predicates is also coNP complete in data complexity.

To the best of our knowledge, so far there are no results on the data complexity of expressive description logics that simultaneously support nominals, inverses, number restrictions, and closed predicates. This leads us to one of the main goals of this thesis: characterizing data complexity of KB satisfiability and query answering for $\mathcal{ALCHOIQ}$ in the presence of closed predicates.

Description Logics and Rules Hybrid languages that couple description logic ontologies with non-monotonic rules represent another way of accommodating partial CWA in DLs. In general, these approaches can be separated into two broad categories: (i) *world-centric* and (ii) *entailment-centric*.

In the first category, an intended model of the knowledge base is a *single* first-order structure that is, in some sense, acceptable to both components. One of the first hybrid formalisms that allow the use of non-monotonic rules is *r-hybrid* [Ros05]. In this approach, a knowledge base is a pair $\mathcal{H} = (\mathcal{T}, \mathcal{P})$, where $\mathcal{T}$ is a first-order theory (or a DL ontology) and $\mathcal{P}$ is a disjunctive Datalog with negation. The non-monotonic semantics of r-hybrid is intuitively given as follows: all predicates occurring in the ontology are considered to be open. That is, in a model $\mathcal{I}$ of $\mathcal{H}$ these predicates can have arbitrary extensions as long as $\mathcal{I}$ still satisfies $\mathcal{T}$ (in a classical sense). The remainder of the predicates occurring in the rules are considered closed and their extensions cannot be arbitrary – they must

be minimal in some sense, i.e., justified by the program. However, it was shown early on that, if one is not careful, combining DLs with rules (even the non-monotonic ones) leads to the loss of decidability [LR98]. To combat this issue, the usual approach is to introduce a syntactic safety condition that ensures that variables in the program range only over a finite number of constants. Rosati's r-hybrid employs the well-known *DL-safety* condition, which requires that all rule variables occur in predicates which cannot occur in the DL ontology. The intuition behind this is that the variables are bound to range only over the constants explicitly mentioned in the knowledge base, restoring decidability. This restriction was then regarded as too strong and relaxed in a later work [Ros06] of the same author as well as in [BOŠ18]).

The second category involves approaches in which rules have little to no access to *individual models* of the DL component, and instead, they perform the reasoning based on the logical consequences of the DL component. A notable representative in this class are *dl-programs* [EIL$^+$08] and its various generalizations. In this approach, the rules and the ontology component are seen as separate layers that safely interact through well-defined interfaces. More precisely, dl-rules allow standard Datalog programs with negation to pose queries to DL knowledge bases with the possibility of (temporarily) modifying the ABox before querying. This limited form of interaction eliminates the need for previously discussed syntactic safety conditions.

A further example of an entailment-centric approach are so-called *hybrid MKNF* knowledge bases [MR07, MHRS06], based on Lifschitz's *logic of minimal knowledge and negation as failure (MKNF)* [Lif91]. In this formalism, a hybrid KB again consists of a DL component and a set of non-monotonic rules that can additionally use a modal operator **K** to inspect the consequences of the DL knowledge base.

To account for the scenarios where both extremes might be inadequate, in this thesis, we introduce a new hybrid formalism that blurs the lines between world-centric and entailment-centric approaches. We thus study hybrid knowledge bases that may process different models of the input ontology in different ways, in the spirit of world-centric approaches, but the intended answer sets, which must be resilient to the different scenarios, are defined via a universal quantification over the models of the ontology, in the spirit of entailment-centric approaches.

1.1.2 Relative Expressiveness and Rewriting Techniques for OMQs

A distinguishing feature of the logics in the DL-Lite family is that they are first-order rewritable, i.e., we can reduce query answering and other reasoning tasks to evaluating first-order queries over a relational database. More specifically, given any OMQ $Q = (\mathcal{T}, q)$ with a TBox $\mathcal{T}$ in some DL and a first-order CQ q, we can define a new query q', expressed as the union of conjunctive queries, that encodes the semantics of $\mathcal{T}$ and q. The query q' has the following convenient property: for any ABox $\mathcal{A}$, the answers to q' over $\mathcal{A}$ coincide with the certain answers to Q over $\mathcal{A}$. Such rewritings come with many benefits: they allow us to reuse the existing database technology that is highly optimized, they

help us compare the expressive powers of the query languages, and in some cases allow us to transfer known results like complexity bounds (see e.g., [CDL+07, CGL+13]). In fact, first-order rewritability underlies many of the known results on the tractability in data complexity of reasoning in DL-Lite. However the original rewriting algorithm for DL-Lite (see PerfRef algorithm in [CDL+05, CDL+07] has the problem that the size of the computed query q' increases exponentially with the number of atoms of the CQ q. This prompted investigations into shorter and more efficient rewritings, see, e.g, [RA10, Ros12, CTS11, RMKZ13], and the existence of succinctness of FO rewritings has been investigated additionally in [BKK+18, GKK+14]. It was then shown in [GS12] that there exists a polynomial time rewriting of CQs mediated by DL-Lite ontologies into non-recursive Datalog, with the small caveat that ABox must contain two distinct individuals. There has also been interest in developing rewriting techniques for the DLs in the $\mathcal{EL}$ family [Ros07b, HLSW15]. One prominent tactic here is to adopt a *combined approach* to query rewriting introduced in [LTW09], which produces a succinct non-recursive Datalog rewriting of the input OMQ if this OMQ is first-order-rewritable, otherwise, it reports non-FO-rewritability. Differently from the previous "pure" rewritings, this approach involves slightly extending the input ABox over which the query is being answered by certain facts that can be inferred from the TBox.

Due to their intractability in data complexity, the existence of first-order rewritings is generally ruled out for expressive description logics. In these cases, the alternative is to go for translations into variants of Datalog. One of the first such approaches provides a rewriting of instance queries mediated by expressive DL $\mathcal{SHIQ}$ ontologies to reasoning in disjunctive Datalog [HMS07] in exponential time in the size of the TBox, assuming the unary coding of numbers. In this approach, the DL ontology is viewed as a set of first-order sentences, as shown possible by [Bor96], which is then converted into clauses and saturated using a first-order resolution/superposition calculus, and the corresponding program is obtained by eliminating function symbols from the saturated theory. On the basis of this translation, the authors show that answering instance queries in $\mathcal{SHIQ}$ is coNP-complete in data complexity. Furthermore, it was shown in [EOŠ12] that CQs mediated by $\mathcal{SH}$ and $\mathcal{SHQ}$ ontologies can also be rewritten into an exponentially-sized Datalog program. The technique that has been used in this rewriting hinges on the fact that one can focus on forest-shaped interpretations when answering CQs over $\mathcal{SH}$ knowledge bases. The authors develop a way to finitely represent such models using so-called *knots*, which are further exploited to compute the desired rewriting. Another result shown in [BtCLW14] states that unions of CQs mediated by certain expressive DLs (though not as expressive as $\mathcal{ALCHOIQ}$) can be rewritten into monadic disjunctive Datalog in exponential time.

Regarding succinct polynomial-time rewritings of expressive DLs, the earliest such rewriting was presented in [ORŠ10]. In this work, the authors show that the KB satisfiability for the Horn fragment of the expressive DL $\mathcal{SHOIQ}$ (i.e., the fragment of $\mathcal{SHOIQ}$ that cannot express disjunctive information) can be translated into a polynomially-sized Datalog program. The presented technique exploits the fact that the knowledge bases expressed

in this logic have a universal model property over which we can do query answering.

Recently, a new rewriting technique was introduced [AOŠ20] for ontology-mediated instance queries in full $\mathcal{ALCHOI}$ with closed predicates, albeit without number restrictions. In this work, the authors provide a game-theoretic characterization of the semantics of the considered class of OMQs, namely instance queries and a restricted fragment of CQs, and based on this characterization, they develop a Datalog program with stable negation that decides the existence of a winning strategy, reminiscent of known type-elimination algorithms for DLs.

Finally, while in this thesis we focus on "pure rewritings", the previously-mentioned combined approach has also been investigated in the context of expressive DLs [CDK18, FCS+15], albeit for Horn fragments.

When it comes to query rewriting in description logics with closed predicates, only very few rewritability results are available: the results from [LSW15] showing that quantifier-free UCQs in DL-Lite$_\mathcal{R}$ with closed predicates are first-order-rewritable, the one in [SFdB09] showing how to rewrite instance queries in $\mathcal{ALC}$ with closed predicates into first-order queries (when possible) and the previously-mentioned result in [AOŠ20] showing that instance queries in $\mathcal{ALCHOI}$ with closed predicates can be rewritten into Datalog with negation under the stable model semantics.

Looking at the results above, it becomes apparent that there are no rewriting results for expressive description logics with or without closed predicates that simultaneously support nominals, role inverses, and number restrictions. In this thesis, we aim to close this gap by providing a polynomial translation of a large class of OMQs mediated by $\mathcal{ALCHOIQ}$ ontologies with closed predicates into Datalog with negation under the stable model semantics. We note that the techniques mentioned above cannot be readily adapted to obtain the desired rewriting for $\mathcal{ALCHOIQ}$, as they are exponential in size, rely on model-theoretic properties that our logic does not possess, or are based on resolution that is known to be tricky for non-monotonic logics.

1.1.3 Expressive Power of OMQs with Closed Predicates

Descriptive complexity [Imm99] is a useful notion that helps us measure the expressiveness of some query language. In this context, we ask whether a given language is powerful enough to express all queries computable within some time or space resources, i.e., belonging to a certain complexity class. In this case, we say that this query language *captures* the considered complexity class.

It is well-known that Datalog, although PTime-complete in data complexity, cannot express all PTime computable queries [Kol91]. However, if we equip input instances with a total ordering of their domain, then Datalog precisely captures PTime [DEGV01]. Furthermore, it was also shown that Datalog with negation under the stable model semantics captures coNP [Sch95], while disjunctive Datalog can express all Σ_2^P-computable queries [EGM97].

In contrast, the expressive power of OMQs has thus far received limited attention. It was shown in [BtCLW14] that there are OMQ languages that capture certain subclasses of coNP related to *constraint satisfaction problems* (CSPs). More recently, a close connection between OMQ languages with the so-called *closed predicates* and *surjective CSPs* was shown in [LSW19].

In this thesis, we continue this line of work and we focus on finding an OMQ language that precisely captures the complexity class coNP. As there are many coNP-computable (or even PTime-computable) non-monotonic queries, such an OMQ language inevitably needs to offer some support for non-monotonic reasoning, making expressive DLs with closed predicates excellent candidates.

1.2 Main Contributions

Keeping in mind our previously-introduced goals and the observations from the previous section, we next summarize the main contributions of this thesis:

- **A polynomial time rewriting of OMQs over $\mathcal{ALCHOIQ}$ with closed predicates into Datalog with negation under the stable model semantics and novel data complexity results.**

  In this thesis, we present a polynomial-time rewriting of *safe-range first-order* queries mediated by ontologies expressed in the DL $\mathcal{ALCHOIQ}$ with closed predicates into a variant of Datalog that supports negation under the stable model semantics. Informally speaking, safe-range queries are first-order queries in which all variables are somehow guarded by closed predicates. We also present a variant of this translation that was specifically tailored for $\mathcal{ALCHOIF}$, a fragment of $\mathcal{ALCHOIQ}$ where global role functionality is the only available type of number restrictions. Both of these rewritings use a very different approach from all other rewritings in the literature and are based on a characterization of the satisfiability problem for this logic as a system of linear inequalities with some side conditions.

  Satisfiability via integer programming. Inspired by the techniques in [Cal96, LST05, PH05], we show that, given a $\mathcal{ALCHOIQ}$ (resp. $\mathcal{ALCHOIF}$) knowledge base $\mathcal{K}$, we can construct a system of integer linear inequalities enriched with implications between inequalities (*enriched systems*, for short) that has a solution in which every variable is assigned either a natural number or a special value representing infinity if and only if this knowledge base admits a model (with some caveats). In such a system obtained from $\mathcal{K}$, every variable corresponds to a *tile* for $\mathcal{K}$. In a nutshell, a tile describes a particular kind of domain element that can be found in the models of $\mathcal{K}$ in terms of the basic concepts that it satisfies and the kinds of neighbors it must possess. Conditions are placed on tiles to ensure local consistency of such descriptions. The enriched system then defines conditions that must be satisfied in order to ensure global consistency, i.e., to

ensure that a model of $\mathcal{K}$ can be built by instantiating all tiles according to the values that their corresponding variables are assigned.

Building and solving enriched systems in Datalog. Based on the characterization above, we build a modular Datalog program $\mathcal{P}_{sat}^{\mathcal{T},\Sigma}$ with negation under the stable model semantics that depends only on the TBox $\mathcal{T}$ and the set Σ of closed predicates and has the following property. For any input ABox $\mathcal{A}$, the intended models of $\mathcal{P}_{sat}^{\mathcal{T},\Sigma}$ over $\mathcal{A}$ correspond to the solutions of the enriched system for the KB $(\mathcal{T}, \Sigma, \mathcal{A})$. Moreover, $\mathcal{P}_{sat}^{\mathcal{T},\Sigma}$ is polynomial in the size of the input TBox and the set of closed predicates assuming the unary coding of numbers occurring in the TBox.

Rewriting safe-range queries in Datalog. We introduce the notion of safe-range queries in which non-answer variables are guarded by closed predicates, which has an effect that during query answering we can focus on the parts of models that only involve those individuals that occur in the knowledge base. Exploiting this property, we adapt the program $\mathcal{P}_{sat}^{\mathcal{T},\Sigma}$ by adding the rules to non-deterministically guess the part of the model over the known individuals. We can then answer database queries over this program instead of the original ontology. We note that safe-range OMQs subsume instance queries and quantifier-free (unions of) conjunctive queries.

Novel data complexity results. As an important consequence of our results, we get that the knowledge base satisfiability in $\mathcal{ALCHOIQ}$ with closed predicates is NP-complete and the considered class of safe-range OMQs is coNP-complete in data complexity. Indeed, in our work, we analyze the size of the enriched systems that are obtained from a given knowledge base $\mathcal{K}$ and we see that such a system is in general exponential in the size of $\mathcal{K}$, however, it is only polynomial if the TBox and the set of closed predicates are considered fixed. Moreover, deciding whether an enriched system has a solution can be done using ordinary integer programming techniques, yielding the desired upper bounds. We also mention that the same upper bounds follow from our translation and known results about the complexity of Datalog with negation. This is a positive result, as it shows that closed predicates in $\mathcal{ALCHOIQ}$ do not increase the data complexity.

- **An OMQ language based on $\mathcal{ALCHOIF}$ with closed predicates that can precisely express all coNP computable queries over ABoxes.**

  Encouraged by our novel complexity results, we show that we can define an OMQ language with closed predicates that can precisely express all generic Boolean queries over ABoxes that are computable in coNP. We first observe that, due to their monotonicity, standard DLs cannot express all coNP queries. Furthermore, we also show that instance queries based on the *non-monotonic* $\mathcal{ALCHOI}$ with closed predicates are still not powerful enough to capture coNP. As a first step towards a positive result, we present an extension of $\mathcal{ALCHOIF}$ with closed predicates with *nominal schemata* [KMKH11] of very restricted shapes and argue that they

do not cause a complexity increase by suitably modifying our integer programming characterization. Finally, we show that the OMQ language that couples inconsistency queries (i.e., queries that ask whether a KB has a model) with ontologies in this extended language has the desired expressive power. Intuitively speaking, a generic Boolean query q is computable in coNP if there exists a non-deterministic Turing machine that in polynomial time decides whether q is "false" over the input ABox. Our proposed OMQ language can accurately express the computations of such NTMs, thereby capturing precisely the complexity class coNP.

- **A procedure for deciding predicate boundedness in the presence of closed predicates.**

  Declaring some predicates closed can result in certain open predicates having only extensions of bounded size due to possible interaction between closed predicates and number restrictions. For example, assume we have a DL TBox $\mathcal{T}$ stating that each employee of a company can take part in at most 5 projects and that all projects have at least one employee: $\mathsf{Empl} \sqsubseteq \leq 5\ \mathsf{assgdTo}.\mathsf{Proj}$ and $\mathsf{Proj} \sqsubseteq \exists \mathsf{assgdTo}^{-}.\mathsf{Empl}$. Further, assume Empl is a closed predicate and we are given an ABox $\mathcal{A}$ that contains a list of all n employees. Then we can easily infer that there are at most $5n$ projects. We may not know exactly which projects these are, but in any model of the knowledge base $(\mathcal{T}, \{\mathsf{Empl}\}, \mathcal{A})$ the extension of Proj will contain at most 5 elements, i.e., Proj is in a way *bounded* by the TBox and the closed predicates.

  Two notions of boundedness. We define two notions of *bounded predicates* for DLs with closed predicate: (i) with respect to a particular knowledge base (*weak boundedness*) and (ii) with respect to a given TBox and a set of closed predicates, independently from the actual data (*strong boundedness*). The intuition behind the two notions is as follows. We say that a predicate p is bounded in a given knowledge base $(\mathcal{T}, \Sigma, \mathcal{A})$ with closed predicates if there exists an integer constant that provides an upper bound on the cardinality of the predicate's extension in all models of this knowledge base. Moreover, if p is bounded regardless of the concrete contents of the ABox $\mathcal{A}$, p is (strongly) bounded w.r.t. $\mathcal{T}$ and Σ.

  Decision procedures, complexity results, and concrete upper bounds on bounded predicate sizes. We provide procedures to decide predicate boundedness when the TBox is expressed in $\mathcal{ALCHOIQ}$ that are once again based on our integer programming characterization of the satisfiability problem for $\mathcal{ALCHOIQ}$ with closed predicates. For strong boundedness, this step involves a reformulation in terms of *finite-infinite satisfiability* problem, where we ask whether a TBox has a model in which some specific predicates have *finite extensions*, while some other given predicates have *infinite extensions*. Both procedures are worst-case optimal and show that checking predicate boundedness in the weak and the strong sense is co-NExpTime-complete for $\mathcal{ALCHOIQ}$ TBoxes. Moreover, if boundedness w.r.t. a KB $\mathcal{K}$ is inferred for some predicate, the concrete bound can be computed and it is double exponential in the size

of the input ontology. For strong boundedness, we establish the existence of a function $f_{\mathcal{T},\Sigma}$, depending on $\mathcal{T}$ and Σ, that computes a finite upper bound on the number of distinct domain elements occurring in extensions of predicates bounded by $\mathcal{T}$ and Σ, in all models of $(\mathcal{T}, \Sigma, \mathcal{A})$, where $\mathcal{A}$ is any ABox over the signature of $\mathcal{T}$ with a maximum of n distinct individuals. While this function is generally a double exponential function, it becomes polynomial when the TBox is fixed, implying the number of elements in bounded predicates grows polynomially relative to the input ABox size.

Relaxation for safe-range queries and safety conditions for ontology-mediated Datalog queries. Finally, we exploit our results to relax the syntactic restriction used to define the above-mentioned safe-range queries by allowing the use of open predicates as guards, as long as they are strongly bounded. We then apply the same tactic to define a new safety condition for OMQs based on $\mathcal{ALCHOIQ}$ and Datalog, providing worst-case optimal results on the combined and data complexity along the way.

- **A novel combination of DLs and non-monotonic rules.** We provide a new hybrid formalism called *Resilient Logic Programs*, or *RLPs* for short, that combine first-order theories (or DL ontologies) and rules with non-monotonic negation to provide reasoning support for systems that should react correctly in all possible situations. In this setting, the theory component describes the situations that the system might face, and the rules are used to react to these situations. Our novel semantics allows us to reduce reasoning tasks like ensuring that the system always has a correct reaction to common reasoning problems for hybrid knowledge bases such as consistency checking.

  Expressiveness analysis. We investigate the expressiveness of RLPs and show they are very powerful. In fact, they easily capture disjunctive Datalog with negation under the stable model semantics and $\exists\forall\exists$-QBFs. The latter result already shows that reasoning in RLPs is Σ_3^P-hard in data complexity, setting them apart from previous hybrid languages. We also illustrate the power of RLPs for configuration problems with incomplete information.

  Decidability and complexity bounds. We show that RLPs are decidable, assuming that reasoning in the logic expressing the theory component is decidable and some rule safety conditions are applied, including the previously-mentioned relaxed safety condition for Datalog queries. We also provide a general complexity upper bound that applies to RLPs whose theory component is written in very expressive first-order fragments like the *guarded negation fragment (GFNO)*. Moreover, we provide concrete algorithms and complexity results for the case where ontologies are written in $\mathcal{ALCHOIQ}$, $\mathcal{ALCHI}$, and DL-Lite$_{\mathcal{F}}$.

  Translation into disjunctive Datalog. Finally, we introduce a restricted fragment of RLPs, in which theories are given as sets of positive disjunctive rules, and the use of default negation is slightly restricted and we show that this

fragment can be rewritten into disjunctive Datalog with negation, opening up a perspective for potential implementation.

1.3 Organization and Relevant Publications

The contributions of this thesis are based on the following peer-reviewed publications, listed in chronological order:

[OPŠ19] Magdalena Ortiz, **Sanja Pavlović**, and Mantas Šimkus. "Answer Set Programs Challenged by Ontologies". In Proceedings of the 32nd International Workshop on Description Logics, DL 2019, CEUR-WS, 2019.

[LOŠ20] **Sanja Lukumbuzya**, Magdalena Ortiz, Mantas Šimkus. "Resilient Logic Programs: Answer Set Programs Challenged by Ontologies". In Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, pp. 2917-2924. 2020.

[GLOŠ20] Tomasz Gogacz, **Sanja Lukumbuzya**, Magdalena Ortiz, and Mantas Šimkus. "Datalog rewritability and data complexity of $\mathcal{ALCHOIF}$ with closed predicates". In Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning, KR 2020, vol. 17, no. 1, pp. 434-444. 2020.

[LŠ21] **Sanja Lukumbuzya**, and Mantas Šimkus. "Bounded Predicates in Description Logics with Counting". In Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, pp. 1966-1972. 2021.

[LOŠ23] **Sanja Lukumbuzya**, Magdalena Ortiz, and Mantas Šimkus. "On the expressive power of ontology-mediated queries: Capturing coNP". In Proceedings of the 36th International Workshop on Description Logics, DL 2023, CEUR-WS, 2023.

[LOŠ24] **Sanja Lukumbuzya**, Magdalena Ortiz, Mantas Šimkus. "Datalog Rewritability and Data Complexity of $\mathcal{ALCHOIQ}$ with closed predicates". Artificial Intelligence (2024): 104099.

The rest of this thesis is structured as follows:

- In Chapter 2, we introduce the essential concepts, definitions, and background information that is necessary to understand the subsequent chapters. In particular, we give an introduction to description logics and Datalog, formally defining their syntax, semantics, and relevant reasoning tasks as well as the concrete languages that we will focus on throughout the rest of this thesis.

- In Chapter 3, we introduce description logics with closed predicates, focusing specifically on the expressive DLs $\mathcal{ALCHOIQ}$ and $\mathcal{ALCHOIF}$. We then show how we can characterize the knowledge satisfiability problem for these logics via integer programming and we state the desired data complexity bounds. The characterization presented in this chapter also plays a major role in the subsequent three chapters.

  The results in this chapter were originally presented in [GLOŠ20] (for $\mathcal{ALCHOIF}$) and in the appendix to [LŠ21] (for $\mathcal{ALCHOIQ}$). These two works were recently combined and extended in [LOŠ24].

- In Chapter 4, we present our rewriting from ontology-mediated safe-range queries in $\mathcal{ALCHOIQ}$ with closed predicates into Datalog, based on the satisfiability characterization from the previous chapter. The results from this chapter were published in [LOŠ24].

- In Chapter 5, we investigate expressiveness of OMQ languages. In particular, we first prove that certain languages are not expressive enough to express all coNP-computable queries. We then present an extension of $\mathcal{ALCHOIF}$ with closed predicates together along with the proof of its complexity and we show that we can define an OMQ language based on this logic that is expressive enough to capture the desired complexity class. The results from this chapter were published in [LOŠ23].

- In Chapter 6, we formalize two notions of predicate boundedness and present corresponding decision procedures and worst-case optimal complexity bounds. We also provide concrete bounds for the number of different objects that can occur in the extensions of bounded predicates, which we then exploit to define relaxed notions of safe-range and safe Datalog queries mediated by $\mathcal{ALCHOIQ}$ ontologies with closed predicates. The results from this chapter were published in [LŠ21].

- In Chapter 7, we formally introduce our resilient logic programs that combine first-order theories with non-monotonic rules. We then proceed to investigate their decidability and expressiveness, and we provide algorithms and complexity results for the cases in which the theory component is written in certain description logics. The results from this chapter were originally published in the workshop paper [OPŠ19] and its conference version [LOŠ20].

- Finally, chapter 8 serves as the conclusion to this thesis and it provides a summary of our results, as well as a discussion of interesting problems that remain open and an outlook on our future plans.

CHAPTER 2

Preliminaries

This chapter gives a tour through selected background notions that this thesis builds upon. In particular, in Section 2.1, we fix some general notations that are used in the sequel. In Section 2.2 we review basic notions relating to computational complexity. Section 2.3 serves as an introduction to Description Logics (DL), reviewing the basic terminology and reasoning tasks, and formally introducing the specific DL languages that are the objects of our investigation. Section 2.4 is dedicated to Datalog and its relevant variants, specifically, disjunctive Datalog and Datalog with negation under the answer set semantics. We note that this chapter offers a good starting point, however, it may not cover all the material from the literature that is used in the remainder of this thesis. Other relevant notions will be recalled as we go along.

2.1 General Notations

We next specifically recall a few relevant notions from first-order logic (FO), but in general, familiarity with the syntax and basics of the model theory of FO is assumed. For a detailed introduction to first-order logic, we refer interested readers to any standard textbook on formal logic and model theory (see, e.g., [Fit96, Mar06]).

We assume the following countably infinite and mutually disjoint sets:

- $\mathsf{N_P}$ of *predicate symbols*,
- $\mathsf{N_V}$ of *variable symbols*, and
- $\mathsf{N_I}$ of *constant symbols*.

Moreover, each predicate symbol $p \in \mathsf{N_P}$ is associated with a non-negative integer called *arity* and denoted by $art(p)$. With the slight abuse of standard FO terminology, the elements in $\mathsf{N_P}$ are often simply referred to as *predicates* or *relations*.

Terms are the elements in $\mathsf{N_V} \cup \mathsf{N_I}$, *atoms* are expressions of the form $p(t_1, \ldots, t_n)$, where p is a predicate symbol from $\mathsf{N_P}$ that has the arity n, and $t_1, \ldots, t_n$ are terms. When the arity of the predicate is either clear or irrelevant, we simply write $p(\vec{t})$, where $\vec{t}$ is a tuple of terms of length $art(p)$. If α is an atom, the expressions of the form α and $\neg\alpha$ are called *literals*. For a literal l, we denote by $terms(l)$ the set of all terms and by $vars(l)$ the set of all variables occurring in l. We say that a term, an atom, or a literal is *ground* if it contains no variables.

2.2 Basics of Complexity Theory

Generally, we assume the reader to be familiar with the basics of complexity theory as well as the usual time and space complexity classes defined upon the Turing model of computation. For a detailed introduction to this topic, we refer to any standard textbook, e.g., [Pap94].

We next review some relevant notions, starting with Turing machines.

2.2.1 Turing Machines

Introduced by Alan Turing in 1936 [Tur37], a *Turing machine* is a relatively simple abstract computational device with the following main parts:

- A *tape* with a left-margin that is infinite on the right, divided into cells in which we can write symbols from a given alphabet. Moreover, we can also replace the symbols that have been already written on the tape with different ones, or we can delete them altogether, by replacing them with a special blank symbol.
- A read/write *head* positioned over the tape that points to one specific cell that is currently being inspected.
- A *state register* that stores the current state of the Turing machine.
- A *transition relation* according to which the computation is performed. Depending on the current state of the machine and the symbol in the cell under the machine's head, this relation gives us instructions on how to proceed by telling us what symbol should be written in this cell, if and how the head should be moved, as well as, what is the next state that the machine should assume. If there is only one way to proceed, the Turing machine is called *deterministic*, and if there are multiple possible ways, then it is called *non-deterministic*.
- Three distinguished states: the *initial state*, the *accepting state* and the *rejecting state*.

Formally, Turing machines are defined as follows.

Definition 2.2.1 (Turing machine)**.** *A* Turing machine (TM) *is a tuple* $M = (\Gamma, Q, \delta, q_0, q_{acc}, q_{rej})$*, where* Γ *is a set of symbols called an* alphabet*,* Q *is a set of* states*,* $\delta \subseteq (\Gamma \cup \{B\}) \times Q \times (\Gamma \cup \{B\}) \times Q \times \{-1, +1\}$ *is a* transition relation*, and* $q_0, q_{acc}, q_{rej} \in Q$ *are the* initial state*, the* accepting state*, and the* rejecting state*, respectively. The symbol* B *is the* blank *symbol. Moreover, if for every* $(s, q) \in (\Gamma \cup \{B\}) \times Q$*, there is exactly one* $(s', q', n) \in (\Gamma \cup \{B\}) \times Q \times \{-1, +1\}$ *such that* $(s, q, s', q', n) \in \delta$*,* M *is called a* deterministic Turing machine (DTM)*, otherwise* M *is a* non-deterministic Turing machine (NTM)*.*

Let Γ be an alphabet. We denote by Γ^* the set of all words (i.e., strings) over Γ, including the empty word ε. Turing machines take words as input and perform computations on them. When a Turing machine M is presented a word w over the specified alphabet Γ, this word is written onto its tape surrounded by all blank symbols, the head is positioned over the first symbol of w, and M is assumed to be in its initial state. The computation is then performed according to the transition relation, as explained above. If at some point during the computation M reaches the accepting state, we say that M accepts w. The set of all input words over Γ which are accepted by M is considered the *language of* M. To formalize this, we next define the notions of *configurations* and *computations* of TMs.

Definition 2.2.2 (Configuration)**.** *Let* $M = (\Gamma, Q, \delta, q_0, q_{acc}, q_{rej})$ *be a TM. A* configuration *of* M *is a triple* (q, w, x, u)*, where* $w, u \in (\Gamma \cup \{B\})^*$*,* $x \in \Gamma \cup \{B\}$*, and* $q \in Q$*.*

The intuition behind a configuration $c = (q, w, x, u)$ is as follows. The head of the TM M points to a cell with the symbol x, while w is the word on the left of the head, and u is the word on the right of the head, after which there are only blank symbols written on the tape. This means that if u is empty, x is followed by only blank symbols. Moreover, M is in state q. We can now define what it means for M to perform one step of computation.

Definition 2.2.3 (Computation step, computation)**.** *Given two configurations* $c_1 = (q_1, w_1, x_1, u_1)$ *and* $c_2 = (q_2, w_2, x_2, u_2)$ *of a TM* $M = (\Gamma, Q, \delta, q_0, q_{acc}, q_{rej})$*, we say that* c_1 yields in one step c_2*, in symbols* $c_1 \to_M c_2$*, if* $q_1 \neq q_{acc}$*,* $q_1 \neq q_{rej}$*, and there is some tuple* $(x_1, q_1, b, q_2, n) \in \delta$ *such that one of the following holds:*

- *if* $n = +1$ *and* $u_1 = a \cdot u_1'$*, where* $a \in \Gamma \cup \{B\}$*, then* $w_2 = w_1 \cdot b$*,* $x_2 = a$*, and* $u_2 = u_1'$*,*
- *if* $n = +1$ *and* u_1 *is empty, then* $w_2 = w_1 \cdot b$*,* $x_2 = \{B\}$*, and* $u_2 = \varepsilon$*, or*
- *if* $n = -1$ *and* $w_1 = w_1' \cdot a$*, where* $a \in \Gamma \cup \{B\}$*, then* $w_2 = w_1'$*,* $x_2 = a$*,* $u_2 = b \cdot u_1$*.*

A (possibly infinite) sequence of configurations $c_1, c_2, \ldots$ *is called a* computation *of* M *if:*

- $c_i \to_M c_{i+1}$*, for all* $i \geq 1$*, and*

- *if the sequence is finite with c_n being the last configuration in it, then there is no configuration c_{n+1} s.t. $c_n \to_M c_{n+1}$.*

A finite computation $c_1, c_2, \ldots, c_n$ is called terminating *if $c_n = (q, u, x, u')$ and $q \in \{q_{acc}, q_{rej}\}$.*

We next define the notion of a computation of M on a word w as well as what it means for M to accept w.

Definition 2.2.4. *Let $M = (\Gamma, Q, \delta, q_0, q_{acc}, q_{rej})$ be a TM and let w be a word over Γ. If w is non-empty, i.e., $w = a \cdot w'$, we call a computation $c_1, c_2, \ldots$ where $c_1 = (q_0, \varepsilon, a, w')$ a computation of M on w. The notion is defined analogously for $w = \varepsilon$, where $c_1 = (q_0, \varepsilon, B, \varepsilon)$.*

Definition 2.2.5 (Word acceptance)**.** *Let $M = (\Gamma, Q, \delta, q_0, q_{acc}, q_{rej})$ be a TM and let w be a word over Γ. We say that M* accepts *w if there is some terminating computation $c_1, \ldots, c_n$ of M on w such that $c_n = (q_{acc}, u, x, u')$, $u, u' \in (\Gamma \cup \{B\})^*$.*

We can now we define what it means for a Turing machine to decide a language.

Definition 2.2.6. *Let $M = (\Gamma, Q, \delta, q_0, q_{acc}, q_{rej})$ be a TM and let $\mathcal{L}$ be a formal language over Γ. We say that M* decides *$\mathcal{L}$ if the following holds, for every word $w \in \Gamma^*$:*

- *all computations of M on w are terminating, and*
- *$w \in \mathcal{L}$ if and only if M accepts w.*

2.2.2 Complexity Classes

As the name suggests, *complexity classes* categorize problems based on how difficult they are to solve. In this thesis, we restrict our attention to *decision problems* that are simply yes-or-no questions on a specified set of instances. Each decision problem Π can be seen as a formal language $\mathcal{L}(\Pi)$ over some alphabet Γ consisting of all the words that encode the yes-instances of this problem, together with the question of whether a given word $w \in \Gamma^*$ encoding a particular problem instance belongs to $\mathcal{L}(\Pi)$. The complexity of Π is measured in terms of how much running time (i.e., computation steps) or memory space (i.e., a number of visited cells) a Turing machine needs to determine whether some input word belongs to $\mathcal{L}(\Pi)$ in the *worst-case scenario* and is usually given as a function of the size of the input instance.

We next list some of the most important complexity classes.

- PTime: the class of all problems that can be decided by a *deterministic* Turing machine in the amount of time polynomial in the size of the input word. More precisely, we say that a decision problem Π is in PTime, if there exists some DTM

M and some polynomial $p(n)$ such that M can decide in $p(|w|)$ steps whether the input word w belongs to $\mathcal{L}(\Pi)$, for every word w encoding an instance of Π.

- NP: the class of all problems that can be decided by a *non-deterministic* Turing machine in the amount of time polynomial in the size of the input word. Formally, this class is defined analogously to PTime, with the difference that we are allowed to use non-deterministic Turing machines.
- ExpTime: the class of all problems that can be decided by a *deterministic* Turing machine in the amount of time that is exponential in the size of the input word. More precisely, a problem Π is in ExpTime, if there exists some DTM M and some polynomial $p(n)$, such that M can decide in $2^{p(|w|)}$ steps whether the input word w belongs to $\mathcal{L}(\Pi)$, for every w encoding an instance of Π.
- NExpTime: the class of all problems that can be decided by a *non-deterministic* Turing machine in the amount of time exponential in the size of the input word. Formally, this class is defined analogously to ExpTime, with the difference that we are allowed to use non-deterministic Turing machines.

The classes kExpTime and NkExpTime generalize ExpTime and NExpTime, respectively, but instead of a single exponential function $2^{p(n)}$, they allow k-time exponential functions. For example, the class 2ExpTime (resp. N2ExpTime) is defined as the class of all problems that can be decided by a DTM (resp. NTM) M within $2^{2^{p(n)}}$ computation steps, where $p(n)$ is a polynomial.

Furthermore, we can also define complexity classes based on how much memory a TM needs to decide a problem. We say that a problem Π is in PSpace (resp. NPSpace) if there exists some DTM (resp. NTM) M and some polynomial $p(n)$ such that M can decide whether the input word w encoding an instance of Π belongs to $\mathcal{L}(\Pi)$ by using at most $p(|w|)$ tape cells. The classes ExpSpace and NExpSpace are defined analogously, but we are allowed to use exponentially many tape cells.

We say that a decision problem Π is a complement of another decision problem Π' if they have the same set of instances, but the yes-instances of Π are no-instances of Π' and vice versa. In terms of the corresponding formal languages, we have that $\mathcal{L}(\Pi)$ is the complement of $\mathcal{L}(\Pi')$, i.e., $\mathcal{L}(\Pi) = \overline{\mathcal{L}(\Pi')}$. We remark that two formal languages over some alphabet Γ are complements of each other if they are (i) disjoint and (ii) their union is not necessarily Γ^*, but some trivially recognizable set. For example, in case of decision problems, we have that $\mathcal{L}(\Pi') \cup \overline{\mathcal{L}(\Pi)}$ is the set of all valid encodings of instances of Π'. Based on this, we can also define complements of complexity classes. For any given complexity class $\mathcal{C}$, the complement complexity class co-$\mathcal{C}$ is defined as the class of all problems whose complement belongs to $\mathcal{C}$.

The following results are known regarding the relationship between individual complexity classes:

$$\text{PTime} \subseteq \text{NP} \subseteq \text{PSpace} \subseteq \text{ExpTime} \subseteq \text{NExpTime} \subseteq \text{ExpSpace} \subseteq \text{2ExpTime}$$

Additionally, as a corollary to Savitch's theorem [Sav70], we know that PSpace = NPSpace and ExpSpace =NExpSpace. Whether the inclusions above are proper or not is a long-standing open problem in computer science, but it is widely believed that they are.

Polynomial hierarchy. Within the class PSpace, there is a whole hierarchy of complexity classes, called the *polynomial hierarchy*, that generalizes the classes NP and coNP. These classes are defined using Turing machines with oracles. In this context, an *oracle* is a subroutine that a Turing machine can call to decide problems in a certain complexity class, under the assumption that these oracle calls take constant time. Given a complexity class $\mathcal{C}$ and an oracle A, we denote by $\mathcal{C}^A$ the class of all the problems that can be solved within the resources allowed by $\mathcal{C}$ by using an oracle to decide problems in the class A. The classes in the polynomial hierarchy are then defined as follows:

$$\begin{aligned} \Delta_0^P := \Sigma_0^P := \Pi_0^P &:= \text{PTime}, \\ \Delta_{i+1}^P &:= \text{PTime}^{\Sigma_i^P}, \\ \Sigma_{i+1}^P &:= \text{NP}^{\Sigma_i^P}, \\ \Pi_{i+1}^P &:= \text{coNP}^{\Sigma_i^P}. \end{aligned}$$

So far, we defined what it means for a problem to belong to a certain complexity class. Next, for a complexity class $\mathcal{C}$, we say that a problem Π is *hard for* $\mathcal{C}$ if every problem in $\mathcal{C}$ can be reduced to Π. A problem Π' can be *reduced* to Π if there is a function f that maps the instances of Π' to instances of Π such that x is a yes-instance of Π' if and only if $f(x)$ is a yes-instance of Π, for all instances x of Π'. One important caveat is that f must be efficiently computable, i.e., for every instance x of Π', we must be able to compute $f(x)$ using strictly fewer resources than allowed by the class $\mathcal{C}$.

2.3 Description Logics

First, a little bit of history. Following the findings from cognitive science and the assumption that the human brain tends to organize information in terms of concepts and relationships between them [SA77], much of the early efforts in the field of knowledge representation were devoted to developing formalisms that allow us to conceptually model some domain of interest. Within this context, two particularly popular approaches emerged: *semantic networks* and *frames*. The main idea behind semantic networks [Qui67] was a rather natural one – represent knowledge graphically in terms of labeled directed graphs (a.k.a networks), where the nodes of the graph correspond to the concepts in the domain of interest and arcs represent the relationships between them. However, with the desire to support a large number of problems, semantic networks grew more and more complex, exposing the need for more structured nodes and links. To this end, many specialized formalisms were introduced, see, e.g., [Bra79] for an overview of the history of semantic networks. Based on the seminal paper of Minsky [Min74], frame-based

approaches represent knowledge via *frames*, which are structures that describe objects or categories through a collection of attributes and their values. Moreover, one can also specify relationships between them, basically forming a network of frames. While initially these approaches were considered preferable to fully-fledged logic-based systems due to their human-centered design, they soon received significant criticism for their lack of formal semantics, which made it difficult to interpret and draw conclusions from the represented knowledge [Woo75]. *Description Logics (DLs)*[BCM+03] were born out of a desire to provide a logical account for the network-based approaches that facilitates both the representation and the reasoning. Most description logics can be viewed as decidable fragments of first-order logic in a simplified syntax that was tailored specifically for representing structured knowledge. Owing to their origins in network-based systems, description logics describe the knowledge using the following basic building blocks:

- *Concept names*, i.e., unary predicates that correspond to atomic classes of objects,
- *Role names*, i.e., binary predicates that correspond to atomic binary relationships between (classes of) objects, and
- *Individuals names*, i.e., constants, that correspond to *named* objects that exist within this domain.

For example, consider the university domain which is often used for explaining the basic concepts relating to description logics and also serves as a running example in [BCM+03, BHLS17][1]. In this context, we could imagine having concept names like Student, Professor, and Course, representing the atomic classes of students, professors, and courses, respectively. Moreover, we could have role names like attends and teaches that represent binary relationships between students and the courses that they attend, as well as professors and the courses they teach. Finally, we can have individual names like *Alice* or *Bob*, representing two named individuals, Alice and Bob, respectively, or *CS101*, representing the course with the same name. This is a good point to mention that there are different ways of interpreting individuals. In this thesis, we make the *Standard Name Assumption (SNA)*, which stipulates that individual names are interpreted as themselves. Roughly speaking, this means that the individual name *Alice* always has to be interpreted as Alice and can never be interpreted as anything else. SNA also implies the *Unique Name Assumption (UNA)*, which ensures that distinct individual names always represent distinct objects.

Concept names, role names, and potentially other primitive elements can be combined using *concept constructors* to build *(complex) concepts*. For example the complex concept

$$\mathsf{Student} \sqcap \exists \mathsf{attends}.\mathsf{Course}$$

corresponds to a class of students that are attending at least one course. In the example above, the symbol $\sqcap$ is to be interpreted as the set intersection (often referred to as

[1] In fact, this section follows closely the explanations and examples given in [BHLS17].

concept conjunction, in DL jargon) and the expression $\exists$attends.Course is to be interpreted as the class of all objects for which there exists an object in the class Course which is connected to this object via the binary relation attends. Other commonly used concept constructors include $\sqcup$ interpreted as set union, $\neg$ interpreted as set complement, and $\forall$ interpreted as universal restriction on objects connected via the specified role. Moreover, we can also build *(complex) roles* using *role constructors*. For example, the role attend^- represents the inverse of the role attend, i.e., if Alice attends Computer Science 101, then *CS101* is connected via attend^- to *Alice*. Recall that at the beginning of this section, we said that the term description logics refers to a *family* of logics. Individual DLs differ mostly in what type of concept/role constructors they permit to build complex expressions.

In the DL setting, the knowledge about some domain of interest is stored in *knowledge bases* that consist of two parts. The first part is the *terminological part* (or a *TBox*) that formally describes relationships between relevant concepts and roles in this domain in the form of a logical theory. The TBox consists of finitely many *terminological axioms*, each one expressing a piece of domain knowledge. The most common type of axiom is called a *concept inclusion*, saying that one concept is completely subsumed by another. Going back to our university example, we can assert the fact that all professors are staff members by writing the following axiom

$$\mathsf{Professor} \sqsubseteq \mathsf{Staff},$$

where the symbol $\sqsubseteq$ is to be interpreted as set inclusion. Similarly, we can say that all teachers must teach at least one course by writing

$$\mathsf{Teacher} \sqsubseteq \exists \mathsf{teaches.Course}.$$

Apart from concept inclusions, other types of axioms are possible, like asserting subsumption relationships between roles, role functionality, role transitivity, etc. These will be introduced a little later. Similarly to before, which types of terminological axioms are allowed in the TBox is governed by the description logic of choice.

The second part of DL knowledge bases is the *assertional part* (or an *ABox*) that stores the data (also called *assertions*), usually, ground atoms over concept/role names. For example, the fact that Alice is attending Computer Science 101 could be stored in an ABox by writing

$$\mathsf{attends}(\mathit{Alice}, \mathit{CS101}).$$

So far, we have been very informal in explaining the basic notions of DLs. However, their main advantage is that they are, in fact, *logics*, and as such they come with precise syntax and semantics. In the remainder of this section, we formalize the intuitions given above.

2.3.1 Expressive DLs

To avoid repeated definitions, we first define some general DL notions. We then shift our focus to the basic description logic $\mathcal{ALC}$, formally introducing its syntax. This logic acts

as a separator between *lighweight DLs*, which are DLs that are strictly less expressive than $\mathcal{ALC}$ and are usually less computationally expensive, and *expressive DLs* which include $\mathcal{ALC}$ and its extensions. In this thesis, we mostly focus on expressive DLs. A few language extensions of $\mathcal{ALC}$ that are particularly important for us are recalled in this section together with $\mathcal{ALC}$, while other DLs, expressive or lightweight, are introduced where relevant. After recounting the syntax of the chosen DLs, we formally define their semantics and relevant reasoning tasks.

Definition 2.3.1. *Assume the following countably infinite, mutually disjoint sets* $\mathsf{N_C} \subseteq \mathsf{N_P}$ *of* concept names*,* $\mathsf{N_R} \subseteq \mathsf{N_P}$ *of* role names*, and* $\mathsf{N_I}$ *of* constants *(also called* individual names*, in DL jargon). The tuple* $(\mathsf{N_C}, \mathsf{N_R}, \mathsf{N_I})$ *forms a* vocabulary *that is used to represent knowledge.*

Definition 2.3.2. *Let* $\mathcal{L}$ *be a description logic. An* $\mathcal{L}$ ABox *is a finite set of assertional axioms permitted in* $\mathcal{L}$ *and an* $\mathcal{L}$ *TBox is a finite set of terminological axioms permitted in* $\mathcal{L}$*. An* $\mathcal{L}$ knowledge base ($\mathcal{L}$ KB) *is a pair* $(\mathcal{T}, \mathcal{A})$*, where* $\mathcal{T}$ *is* $\mathcal{L}$ *TBox and* $\mathcal{A}$ *is a* $\mathcal{L}$ *ABox.*

When $\mathcal{L}$ is irrelevant or clear from the context, we simply refer to $\mathcal{L}$ TBoxes, $\mathcal{L}$ ABoxes, and $\mathcal{L}$ knowledge bases as *TBoxes*, *ABoxes* and *knowledge bases (KBs)*, respectively.

2.3.2 $\mathcal{ALC}$ and Extensions

We begin by formally introducing the syntax of $\mathcal{ALC}$ and selected extensions.

Definition 2.3.3 (Syntax of $\mathcal{ALC}$)**.** *In* $\mathcal{ALC}$*, the set of roles coincides with the set of* $\mathsf{N_R}$ *of role names, while* concepts *are defined inductively as follows:*

- *every concept name* $A \in \mathsf{N_C}$ *is a concept,*
- $\top$ (top) *and* $\bot$ (bottom) *are concepts,*
- *if* C_1 *and* C_2 *are concepts, then so are* $C_1 \sqcap C_2$ *(*concept conjunction*),* $C_1 \sqcup C_2$ *(*concept disjunction*), and* $\neg C_1$ *(*concept negation*), and*
- *if* C *is a concept and* r *is a role, then* $\exists r.C$ *(*existential restriction*) and* $\forall r.C$ *(*universal restriction*) are concepts.*

A concept inclusion (CI) *is an expression of the form* $C \sqsubseteq D$*, where* C *and* D *are concepts. CIs are the only* terminological axioms *(or simply,* axioms*) in* $\mathcal{ALC}$*. An* assertional axiom *(or simply, an* assertion*) is an expression of the form* $C(a)$ *(*positive concept assertion*),* $\neg C(a)$ *(*negative concept assertion*),* $r(a, b)$ *(*positive role assertion*), or* $\neg r(a, b)$ *(*negative role assertion*), where* C *is a concept,* r *is a role, and* a *and* b *are individuals in* $\mathsf{N_I}$*.*

Remark 2.3.4. *For ease of presentation, our definition of ABoxes allows negative role assertions. Note that this is, however, a slight deviation from the standard definition, which only allows positive role assertions.*

As usual in DLs, we sometimes use $C \equiv D$ as an abbreviation for $C \sqsubseteq D$ and $D \sqsubseteq C$, where C and D are concepts. Furthermore, we may write $\exists r$ (resp. $\forall r$) to abbreviate $\exists r.\top$ (resp. $\forall r.\top$).

We next give an example of an $\mathcal{ALC}$ KB. To this end, we once again consider the university domain and we assume that we have the following concept names available in $\mathsf{N_C}$: Student, Professor, TA, Teacher, Course, GradCourse, BScStudent, whose intuitive meanings should be clear from the chosen names. Moreover, we assume we have the role name teaches in $\mathsf{N_R}$ and $\mathsf{N_I}$ contains the individuals *Alice* and *CS101*.

Example 2.3.5. *Let $\mathcal{T}_1$ be a TBox consisting of the following axioms:*

$$\begin{aligned} \mathsf{BScStud} &\sqsubseteq \mathsf{Student}, \\ \mathsf{Student} \sqcap \exists \mathsf{teaches}.\mathsf{Course} &\sqsubseteq \mathsf{TA}, \\ \mathsf{BScStud} &\sqsubseteq \forall \mathsf{teaches}.\neg \mathsf{GradCourse}, \\ \mathsf{Professor} \sqcup \mathsf{TA} &\sqsubseteq \mathsf{Teacher}. \end{aligned}$$

Let $\mathcal{A}_1$ be an ABox consisting of the following assertions:

$$\mathsf{BScStud}(\mathit{Alice}), \qquad \mathsf{Course}(\mathit{CS101}), \qquad \mathsf{teaches}(\mathit{Alice}, \mathit{CS101}).$$

Intuitively, the axioms in $\mathcal{T}_1$ formalize the following pieces of knowledge: bachelor students are students, students who teach some courses are teaching assistants, bachelor students only teach courses that are not bachelor courses, and professors and teaching assistants are teachers. The assertions in $\mathcal{A}_1$ simply state that Alice is a bachelor student teaching the CS101 course.

The pair $\mathcal{K}_1 = (\mathcal{T}_1, \mathcal{A}_1)$ is an $\mathcal{ALC}$ knowledge base.

As already mentioned, $\mathcal{ALC}$ can be extended with additional concept and role constructors, as well as terminological axioms. We next recount some of the most common extensions.

- *Role inclusions* ($\mathcal{H}$): terminological axioms additionally include *role inclusions (RIs)*, which are expressions of the form $r \sqsubseteq s$, where r and s are roles.
- *Nominals* ($\mathcal{O}$): $a \in \mathsf{N_I}$, then $\{a\}$ is a concept (also called a *nominal*).
- *Role inverses* ($\mathcal{I}$): if for every $p \in \mathsf{N_R}$, then p^- is a role.
- *Qualified number restrictions* ($\mathcal{Q}$): if C is a concept, r is a role, and n is a non-negative integer and $\leq nr.C$, $\geq nr.C$ are concepts.

- *Role transitivity* ($\mathcal{S}$): terminological axioms additionally include expressions of the form $\mathsf{trans}(r)$, where r is a role.
- *Role functionality* ($\mathcal{F}$): terminological axioms additionally include expressions of the form $\mathsf{func}(r)$, where r is a role.

According to the well-established naming schema for DLs, the letters given in the brackets next to the extensions listed above are appended to $\mathcal{ALC}$ to denote the presence of the additional constructors that this extension allows. The only exception is transitivity – we simply write $\mathcal{S}$ to denote $\mathcal{ALC}$ with transitivity. Throughout this thesis, we will mostly focus on $\mathcal{ALCHOIQ}$ and its sublogics $\mathcal{ALCHOIF}$ and $\mathcal{ALCHI}$. As the convention suggests, all of these logics extend $\mathcal{ALC}$ with role inclusions, nominals, and inverses. Additionally, $\mathcal{ALCHOIQ}$ supports general qualified number restrictions, while $\mathcal{ALCHOIF}$ allows for global role functionality.

Let us illustrate the modeling capabilities of $\mathcal{ALCHOIQ}$ on an example. Assume that, in addition to the concept and role names introduced so far, we also have the following concept names in $\mathsf{N_C}$: Curriculum, BCsCurriculum, Subject, CompSci, LargeCourse, and the following role names in $\mathsf{N_R}$: about, hasPart.

Example 2.3.6. *Recall the KB $\mathcal{K}_1 = (\mathcal{T}_1, \mathcal{A}_1)$ from Example 2.3.5. Let $\mathcal{T}_2$ be the extension of $\mathcal{T}_1$ with the following axioms:*

$$\begin{aligned}
\mathsf{Curriculum} &\sqsubseteq \exists \mathsf{about}.\mathsf{Subject},\\
\mathsf{Curriculum} &\sqsubseteq \geq 5\,\mathsf{hasPart}.\mathsf{Course} \sqcap \leq 20\,\mathsf{hasPart}.\mathsf{Course},\\
\mathsf{BCsCurriculum} &\sqsubseteq \mathsf{Curriculum},\\
\mathsf{CompSci} &\sqsubseteq \mathsf{Subject},\\
\mathsf{BCsCurriculum} \sqcap \exists \mathsf{about}.\mathsf{CompSci} &\sqsubseteq \exists \mathsf{hasPart}.\{\mathit{CS101}\},\\
\mathsf{Course} &\sqsubseteq \exists \mathsf{teaches}^{-}.\mathsf{Teacher}\\
\mathsf{LargeCourse} &\sqsubseteq \geq 3\,\mathsf{teaches}^{-}.\mathsf{Teacher}
\end{aligned}$$

The axioms above express the following: each curriculum is about some subject and consists of at least 5 and at most 20 courses, every bachelor curriculum is a curriculum, computer science is a subject, CS101 is a part of every bachelor curriculum that is about computer science, every course is taught by at least one teacher, and large courses must be taught by at least three teachers.

Furthermore, let $\mathcal{A}_2 = \mathcal{A}_1 \cup \{\mathsf{Curriculum}(\mathit{CSCurr})\}$.

The pair $\mathcal{K}_2 = (\mathcal{T}_2, \mathcal{A}_2)$ is an $\mathcal{ALCHOIQ}$ knowledge base. In fact, as we do not have any RIs, $\mathcal{K}_2$ is also an $\mathcal{ALCOIQ}$ KB.

Before we move on to the semantics, some notational remarks are in order. We denote the set of all roles as $\mathsf{N_R^+}$. With a slight abuse of notation, we write r^- to denote p^- if

$r = p$, and p if $r = p^-$, for a role name $p \in \mathsf{N_R}$. We say that the role r^- is the *inverse* of r. Given a set R of roles, R^- denotes the set $\{r^- : r \in R\}$. Furthermore, we let $\mathsf{N_C^+} = \mathsf{N_C} \cup \{\top, \bot\} \cup \{\{c\} : c \in \mathsf{N_I}\}$. The concepts in $\mathsf{N_C^+}$ are called the *basic concepts*. For a TBox, an ABox, or a KB $\mathcal{X}$, we denote by $\mathsf{N_C}(\mathcal{X})$ and $\mathsf{N_R}(\mathcal{X})$ the set of concept and role names occurring in $\mathcal{X}$, their union being the *signature* of $\mathcal{X}$, denoted by $\mathsf{sig}(\mathcal{X})$. Moreover, we denote by $\mathsf{N_C^+}(\mathcal{X})$ the set of basic concepts occurring in $\mathcal{X}$, by $\mathsf{N_R^+}(\mathcal{X})$ the set of roles occurring in $\mathcal{X}$ and their inverses, and by $\mathsf{N_I}(\mathcal{X})$ the set of individuals occurring in $\mathcal{X}$. In line with the standard database terminology, we also refer to the set $\mathsf{N_I}(\mathcal{X})$ as the *active domain* of $\mathcal{X}$.

Naturally, as DLs are fragments of first-order logic, the semantics is given in terms of first-order interpretations.

Definition 2.3.7 (Semantics)**.** *An* interpretation *is a pair $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$, where $\Delta^\mathcal{I}$ is a non-empty set called the* domain *and $\cdot^\mathcal{I}$ is the* interpretation function *that assigns to each $a \in \mathsf{N_I}$ a domain element $a^\mathcal{I} \in \Delta^\mathcal{I}$, to each $A \in \mathsf{N_C}$ a set $A^\mathcal{I} \subseteq \Delta^\mathcal{I}$, and to each $r \in \mathsf{N_R}$ a set $r^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$. The interpretation function is extended to the remaining concepts and roles in the language as summarized in Table 2.1.*

For a concept or a role name P and an interpretation I, we call the set $P^\mathcal{I} \subseteq \Delta^\mathcal{I}$ the *extension of P (in $\mathcal{I}$)*. Furthermore, we say that a domain element $d \in \Delta^\mathcal{I}$ *participates in the concept C (in $\mathcal{I}$)*, if d is in the extension of C. Similarly, for a pair of domain elements (e, d), we say that (e, d) participates in the role r, if (e, d) is in the extension of r.

We next formally define what it means for an interpretation to satisfy axioms or assertions.

Definition 2.3.8. *Let $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ be an interpretation. We say that $\mathcal{I}$* satisfies

- *a concept inclusion $C_1 \sqsubseteq C_2$, if $C_1^\mathcal{I} \subseteq C_2^\mathcal{I}$,*
- *a role inclusion $r_1 \subseteq r_2$, if $r_1^\mathcal{I} \subseteq r_2^\mathcal{I}$,*
- *a role transitivity axiom $\mathsf{trans}(r)$, if for every $e_1, e_2, e_3 \in \Delta^\mathcal{I}$, $\{(e_1, e_2), (e_2, e_3)\} \subseteq r^\mathcal{I}$ implies $(e_1, e_3) \in r^\mathcal{I}$,*
- *a role functionality axiom $\mathsf{func}(r)$, if $\{(e_1, e_2), (e_1, e_3)\} \subseteq r^\mathcal{I}$ implies $e_2 = e_3$, for every $e_1, e_2, e_3 \in \Delta^\mathcal{I}$,*
- *a concept assertion $C(a)$, if $a \in C^\mathcal{I}$, and*
- *a role assertion $r(a, b)$, if $(a, b) \in r^\mathcal{I}$.*

We next define what it means for an interpretation to be a model of a TBox, an ABox, or a knowledge base.

Definition 2.3.9 (TBox, ABox and KB models)**.** *Let $\mathcal{T}$ be a TBox and $\mathcal{A}$ be an ABox. We say that $\mathcal{I}$* satisfies *$\mathcal{T}$, if it satisfies every axiom in $\mathcal{T}$ and additionally $a^{\mathcal{I}} = a$, for each individual a occurring in $\mathcal{T}$ (due to SNA). Similarly, $\mathcal{I}$* satisfies *$\mathcal{A}$, if it satisfies every assertion in $\mathcal{A}$ and $a^{\mathcal{I}} = a$, for each individual a occurring in $\mathcal{A}$. Finally, $\mathcal{I}$* satisfies *the knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, if $\mathcal{I}$ satisfies both $\mathcal{T}$ and $\mathcal{A}$. If $\mathcal{I}$ satisfies $\mathcal{K}$ (resp. $\mathcal{T}$ or $\mathcal{A}$), then $\mathcal{I}$ is called a* model *of $\mathcal{K}$ (resp. $\mathcal{T}$ or $\mathcal{A}$).*

For an interpretation $\mathcal{I}$ and an axiom, an assertion, a TBox, an ABox, or a KB $\mathcal{X}$, we sometimes write $\mathcal{I} \vDash \mathcal{X}$ to denote that $\mathcal{I}$ satisfies $\mathcal{X}$.

Example 2.3.10. *Let $\mathcal{I}$ be an interpretation with $\Delta^{\mathcal{I}} = \{Alice, CS101, CSCurr, c_1, c_2, c_3, c_4, p_1, p_2, cs\}$, and $\cdot^{\mathcal{I}}$ is defined as follows:*

$$
\begin{aligned}
\mathsf{BScStud}^{\mathcal{I}} = \mathsf{Student}^{\mathcal{I}} = \mathsf{TA}^{\mathcal{I}} &= \{Alice\} \\
\mathsf{Professor}^{\mathcal{I}} &= \{p_1, p_2\} \\
\mathsf{Teacher}^{\mathcal{I}} &= \{Alice, p_1, p_2\} \\
\mathsf{Course}^{\mathcal{I}} &= \{CS101, c_1, c_2, c_3, c_4\} \\
\mathsf{LargeCourse}^{\mathcal{I}} &= \{CS101\} \\
\mathsf{GradCourse}^{\mathcal{I}} &= \emptyset \\
\mathsf{Curriculum}^{\mathcal{I}} = \mathsf{BCsCurriculum}^{\mathcal{I}} &= \{CSCurr\} \\
\mathsf{Subject}^{\mathcal{I}} = \mathsf{CompSci}^{\mathcal{I}} &= \{cs\} \\
\mathsf{teaches}^{\mathcal{I}} &= \{(Alice, CS101), (p_1, CS101), (p_2, CS101), \\
&\quad (p_1, c_1), (p_1, c_2), (p_2, c_3), (p_2, c_4)\} \\
\mathsf{attends}^{\mathcal{I}} &= \{(Alice, c_2)\} \\
\mathsf{hasPart}^{\mathcal{I}} &= \{(CSCurr, CS101\} \\
\mathsf{about}^{\mathcal{I}} &= \{(CSCurr, cs)\}
\end{aligned}
$$

Moreover, due to SNA, we have that $Alice^{\mathcal{I}} = Alice$, $CS101^{\mathcal{I}} = CS101$, and $CSCurr^{\mathcal{I}} = CSCurr$.

It is easy to verify that the interpretation $\mathcal{I}$ is a model of $\mathcal{K}_2$ from Example 2.3.6. Consider now another interpretation $\mathcal{I}'$ that is obtained from $\mathcal{I}$ by removing $(Alice, CS101)$ from $\mathsf{teaches}^{\mathcal{I}}$. $\mathcal{I}'$ is not a model of $\mathcal{K}_2$ as it violates the axiom $\mathsf{LargeCourse} \sqsubseteq \geq 3\mathsf{teaches}.\mathsf{Teacher}$.

A little remark about SNA is in order. In the DL literature, SNA is sometimes defined as *interpreting every individual in $\mathsf{N_I}$ as itself*. This has the following effect: all models of some TBox, ABox, or KB must include all of $\mathsf{N_I}$ in their domain, which makes these domains infinite. Generally speaking, this is not really a problem, but it is sometimes counter-intuitive in very expressive logics, like those that simultaneously support nominals, inverse roles, and number restrictions. Namely, simple KBs that one would expect to be

Constructor	Syntax	Interpretation
Concept names	A	$A^{\mathcal{I}}$
Top	$\top$	$\Delta^{\mathcal{I}}$
Bottom	$\bot$	$\emptyset$
Nominals	$\{a\}$	a
Negation	$(\neg C)$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
Conjunction	$(C_1 \sqcap C_2)$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
Disjunction	$(C_1 \sqcup C_2)$	$C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
Existential restr.	$(\exists r.C)$	$\{d_1 \in \Delta^{\mathcal{I}} :$ for some $d_2 \in \Delta^{\mathcal{I}}, (d_1, d_2) \in r^{\mathcal{I}}$ and $d_2 \in C^{\mathcal{I}}\}$
Universal restr.	$(\forall r.C)$	$\{d_1 \in \Delta^{\mathcal{I}} :$ for all $d_2 \in \Delta^{\mathcal{I}}, (d_1, d_2) \in r^{\mathcal{I}}$ implies $d_2 \in C^{\mathcal{I}}\}$
Number restr.	$(?nr.C)^{\mathcal{I}}$	$\{d_1 \in \Delta^{\mathcal{I}} : \lvert\{d_2 \in \Delta^{\mathcal{I}} : (d_1, d_2) \in r^{\mathcal{I}}$ and $d_2 \in C^{\mathcal{I}}\}\rvert \, ? \, n\}$, for $? \in \{\leq, \geq\}$.
Role names	r	$r^{\mathcal{I}}$
Inverse role	r^-	$\{(d_1, d_2) : (d_2, d_1) \in r^{\mathcal{I}}\}$

Table 2.1: Interpretation function extended to complex concepts and roles, where $A \in \mathsf{N_C}$, $r \in \mathsf{N_R}$, C, C_1, and C_2 are concepts, and $a \in \mathsf{N_I}$.

satisfiable are rendered unsatisfiable due to an infinite domain that otherwise plays no role. To illustrate this, consider the following example:

Example 2.3.11. *Consider the following KB* $\mathcal{K} = (\{A \sqsubseteq \{a\}, \top \sqsubseteq \exists r.A, \mathsf{func}(r^-)\}, \emptyset)$. *If we do not force interpretations to interpret every individual from* $\mathsf{N_I}$, *the interpretation* $\mathcal{I}$ *with* $\Delta^{\mathcal{I}} = \{a\}$, $A^{\mathcal{I}} = \{a\}$, *and* $r^{\mathcal{I}} = \{(a, a)\}$ *would be a model of* $\mathcal{K}$ *and thus,* $\mathcal{K}$ *would be satisfiable. Notice that any interpretation* $\mathcal{I}'$ *with* $\mathsf{N_I} \subseteq \Delta^{\mathcal{I}'}$ *cannot be a model of* $\mathcal{K}$, *which means that* $\mathcal{K}$ *is unsatisfiable under the standard definition of SNA, which forces interpretation domains to contain the countably infinite set* $\mathsf{N_I}$. *Indeed, due to* $A \sqsubseteq \{a\}$ *we can have at most one element participating in* A, *i.e.,* $A^{\mathcal{I}'} = \{a\}$. *Furthermore, due to* $\top \sqsubseteq \exists r.A$, *every element* d *in the domain must participate in* r *together with* a, *i.e.,* $(d, a) \in r^{\mathcal{I}'}$, *for every* $d \in \Delta^{\mathcal{I}'}$. *However, this violates the functionality of* r *so* $\mathcal{I}'$ *does not satisfy* $\mathcal{K}$.

For this reason, our definition of models under SNA requires the interpretations to only interpret those individual names that are explicitly mentioned in the TBox, ABox, or KB.

We note that, as description logics use only unary (concept names) and binary (role names) predicate symbols, a DL interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ can be viewed as a labeled directed graph whose nodes correspond to the domain elements in $\Delta^{\mathcal{I}}$ labeled with the basic concepts that they participate in, and an arc from some node e to some node d represents the pair $(e, d) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ and is labeled with the roles that (e, d) participates in. More formally, $\mathcal{I}$ induces the graph $\mathcal{G}_{\mathcal{I}} = (\Delta^{\mathcal{I}}, E, \mathcal{L})$, where $E = \{(d_1, d_2) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} : (d_1, d_2) \in r^{\mathcal{I}}$ for some role $r\}$, and $\mathcal{L}$ is a labeling function that associates a label to each vertex and arc in the graph as follows:

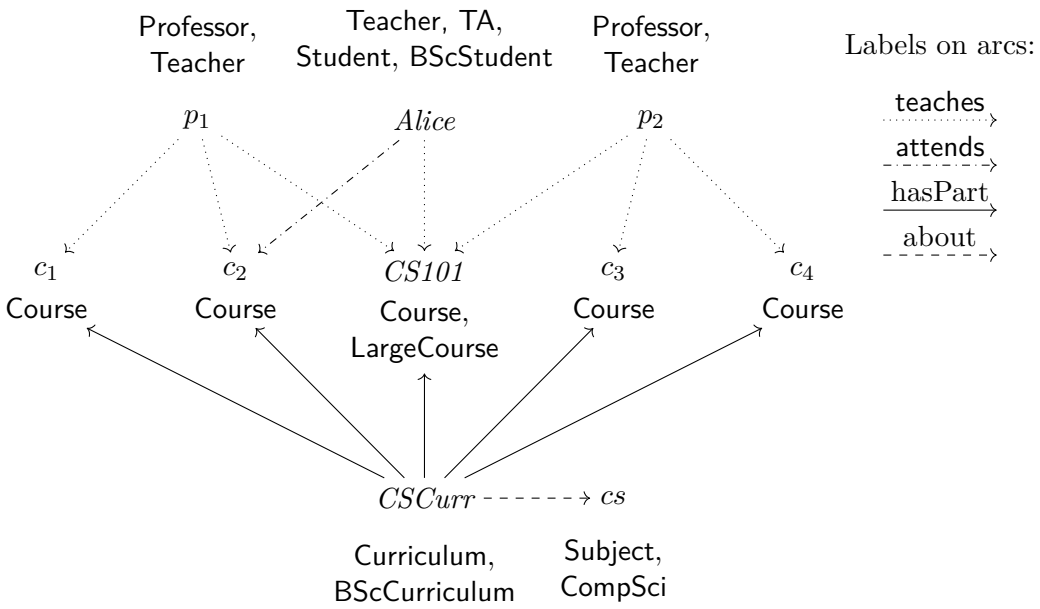


Figure 2.1: Graphical representation of the interpretation $\mathcal{I}$ from Example 2.3.10.

- for $d \in \Delta^{\mathcal{I}}$, $\mathcal{L}(d) = \{C \in \mathsf{N}_{\mathsf{C}}^{+} : d \in C^{\mathcal{I}}\}$, and
- for $(d_1, d_2) \in E$, $\mathcal{L}((d_1, d_2)) = \{r \in \mathsf{N}_{\mathsf{R}}^{+} : (d_1, d_2) \in r^{\mathcal{I}}\}$.

We give an example to illustrate this.

Example 2.3.12. *The graph depicted in Figure 2.1 corresponds to the interpretation $\mathcal{I}$ from Example 2.3.10.*

In the rest of this thesis, we often talk about *R-successors* of some domain element e in some interpretation $\mathcal{I}$, where $R \subseteq \mathsf{N}_{\mathsf{R}}^{+}$ is a set of roles. Keeping the previous definition in mind, it should be intuitively clear that this refers to all domain elements d, such that $(e, d) \in E$ and $\mathcal{L}(e, d) = R$. Being able to visualize interpretations as graphs will be particularly helpful for explaining the intuitions behind some of our results, but more on that later.

2.3.3 Standard Reasoning Tasks and Complexity

Description logics were introduced to facilitate reasoning with ontologies and a large number of different reasoning tasks has been considered in the literature.

Basic reasoning tasks. One of the most basic questions one can ask is whether a given knowledge base admits a model.

Definition 2.3.13 (KB satisfiability)**.** *A KB $\mathcal{K}$ is* satisfiable *(or* consistent*) if it has a model, otherwise, it is* unsatisfiable *(or* inconsistent*). The corresponding decision problem of determining whether $\mathcal{K}$ is satisfiable is called the* knowledge base satisfiability problem*, sometimes also referred to as the* knowledge base consistency problem*.*

Some other common reasoning tasks in DLs include:

- **Concept satisfiability w.r.t. a KB:** given a KB $\mathcal{K}$ and a concept C, is C satisfiable w.r.t. $\mathcal{T}$, i.e., does there exist a model of $\mathcal{K}$ in which the extension of C is non-empty?
- **Concept/role subsumption w.r.t a KB:** given a KB $\mathcal{K}$ and two concepts C and D (resp. roles r and s), is $C \sqsubseteq D$ (resp. $r \sqsubseteq s$) entailed by $\mathcal{K}$?
- **Instance checking w.r.t a KB:** given a KB $\mathcal{K}$, a concept C (resp. role r) and an individual a (resp. pair of individuals (a, b)), is a (resp. (a, b)) an *instance* of C (resp. r), i.e., does $\mathcal{K}$ entails $C(a)$ (resp. $r(a, b)$)?

Query answering. As a technique for information retrieval, answering queries has also been considered in the context of DL knowledge bases. In this setting, the DL TBox is to be taken into account as background knowledge, turning simple query evaluation into a logical reasoning task and resulting in the paradigm often referred to as *ontology-mediated query answering*. However, before we focus on ontology-mediated queries, let us first introduce the general notion of a *query*.

Definition 2.3.14 (First-Order Query)**.** *A* (first-order) query *(or an* FO query*) q is a first-order formula over the predicates in $\mathsf{N_C} \cup \mathsf{N_R}$, the variables from $\mathsf{N_V}$, and the constants from $\mathsf{N_I}$. W.l.o.g. we assume that no variable occurs both quantified and free in q, and we call the free variables of q* answer variables*. We may write $q(\vec{x})$ to denote the query q with answer variables $\vec{x}$. If q contains no free variables, q is a* Boolean query*.*

We often call FO queries also *database queries*, to emphasize the distinction between them and ontology-mediated queries that are introduced below. The *answers* to an FO query q in some interpretation $\mathcal{I}$ are defined in terms of *matches*, which are mappings from the terms occurring in q, denoted by $terms(q)$, to the domain of $\mathcal{I}$ that make the query true in $\mathcal{I}$.

Definition 2.3.15 (Query Match, Query Answer)**.** *Given an interpretation $\mathcal{I}$, a* match *π for a query $q(\vec{x})$ with answer variables $\vec{x} = (x_1, \ldots, x_n)$ in $\mathcal{I}$ is a mapping $terms(q(\vec{x})) \to \Delta^{\mathcal{I}}$ such that the following holds:*

- *$\pi(a) = a^{\mathcal{I}}$, for every $a \in \mathsf{N_I}$ that occurs in q, and*
- *$\mathcal{I} \vDash q(\pi(\vec{x}))$, where $q(\pi(\vec{x}))$ denotes the formula obtained from $q(\vec{x})$ by substituting $\pi(x_i)$ for x_i, for each $1 \leq i \leq n$.*

A tuple $\vec{a} = (a_1, \ldots, a_n)$ *of domain elements in* $\Delta^{\mathcal{I}}$ *is an* answer to $q(\vec{x})$ in $\mathcal{I}$ *if there exists a match* π *for* q *in* $\mathcal{I}$ *such that* $\pi(x_i) = a_i$*, for* $1 \leq i \leq n$*. We denote by* $ans(q, \mathcal{I})$ *the set of all answers to* q *in* $\mathcal{I}$*.*

We can now define what it means to answer queries over DL knowledge bases which is defined in terms of the *certain answer semantics*.

Definition 2.3.16 (Query Answering over a KB)**.** *Given an KB* $\mathcal{K}$ *and a query* q*, a tuple of individuals* $\vec{a} = (a_1, \ldots, a_n)$ *from* $\mathsf{N_I}(\mathcal{K})$ *is a* certain answer *to* q *over* $\mathcal{K}$*, if* $\vec{a}$ *is an answer to* q *in every model of* $\mathcal{K}$*. We denote by* $cert(q, \mathcal{K})$ *the set of all certain answers of* q *over* $\mathcal{K}$*. The problem of deciding whether* $\vec{a} \in cert(q, \mathcal{K})$ *is called the* query answering problem*.*

In the case of Boolean queries, we also talk about *query entailment*.

Definition 2.3.17 (Query Entailment Problem)**.** *Given a KB* $\mathcal{K}$ *and a query* q*, we say that* $\mathcal{K}$ *entails* q*, in symbols* $\mathcal{K} \models q$*, if* q *is true in every model of* $\mathcal{K}$*. The problem of deciding whether* q *is entailed by* $\mathcal{K}$ *is called the* query entailment problem*.*

Finally, it often makes sense to consider the setting in which one has a fixed TBox $\mathcal{T}$ and wants to answer the same database query q over different data taking the information from $\mathcal{T}$ into account. To this end, we introduce the notion of *ontology-mediated queries*.

Definition 2.3.18 (Ontology-Mediated Query)**.** *An* ontology-mediated query (OMQ) *is a pair* $(\mathcal{T}, q)$*, where* $\mathcal{T}$ *is a DL TBox and* q *is an FO query.*

We can now define certain answers of OMQs over ABoxes as follows:

Definition 2.3.19 (Ontology-Mediated Query Answering (OMQA))**.** *Given an OMQ* $Q = (\mathcal{T}, q)$ *and an ABox* $\mathcal{A}$*, a tuple of individuals* $\vec{a} = (a_1, \ldots, a_n)$ *from* $\mathsf{N_I}(\mathcal{T}) \cup \mathsf{N_I}(\mathcal{A})$ *is a* certain answer *to* Q *over* $\mathcal{A}$*, if* $\vec{a}$ *is an answer to* q *in every model of* $(\mathcal{T}, \mathcal{A})$*. We denote by* $cert(Q, \mathcal{A})$ *the set of all certain answers of* Q *over* $\mathcal{A}$*. The problem of deciding whether* $\vec{a} \in cert(Q, \mathcal{A})$ *is called the* ontology-mediated query answering (OMQA) problem*.*

An *ontology-mediated query language* is defined by the choice of the description logic in which the TBox component of OMQs is expressed as well as the fragment of first-order logic to which the FO query component belongs. For example, the language of instance queries mediated by $\mathcal{ALCHOIQ}$ ontologies consists of all OMQs $(\mathcal{T}, q)$ where $\mathcal{T}$ is an $\mathcal{ALCHOIQ}$ TBox and q is an atomic query of the shape $C(x)$ or $r(x, y)$, where $C \in \mathsf{N_C}$ and $r \in \mathsf{N_R}$.

Monotonicity As previously stated, most standard DLs, including $\mathcal{ALCHOIQ}$ and its sublogics, can be seen as fragments of first-order logic and, as such, they obey the principle of *monotonic entailment*. Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ and $\mathcal{K}' = (\mathcal{T}', \mathcal{A}')$ be two knowledge bases. We say that $\mathcal{K}'$ *extends* $\mathcal{K}$ if $\mathcal{T} \cup \mathcal{A} \subseteq \mathcal{T}' \cup \mathcal{A}'$. The principle of monotonic entailment then states that if $\mathcal{K}$ logically entails some Boolean query q, then any KB $\mathcal{K}'$ that extends $\mathcal{K}$ also entails q. As a consequence, given an OMQ $Q = (\mathcal{T}, q)$ whose TBox can be expressed in first-order logic, an ABox $\mathcal{A}$ and tuple $\vec{a}$ of individuals, we have that if $\vec{a}$ is a certain answer to Q over $\mathcal{A}$ then $\vec{a}$ is a certain answer to Q over any ABox extending $\mathcal{A}$. We say that an OMQ language is *monotonic* if all OMQs expressible in this language are monotonic.

Complexity of Reasoning. From the very beginning, there has been a lot of interest in the DL community in characterizing the precise computational complexity of various reasoning problems across different DLs. As usual, we consider the *worst-case complexity* of the problem at hand and we measure it in the size of the input. With a slight abuse of notation, we use $|X|$ to denote the length of the encoding of X.

We also distinguish between the following two complexity measures, depending on what parts of the KB/OMQ are considered parts of the input:

- *Combined complexity:* everything is considered part of the input. Thus, for the basic reasoning tasks listed above, the input size is given as $|\mathcal{T}| + |\mathcal{A}|$, where $(\mathcal{T}, \mathcal{A})$ is the input KB. In the OMQA setting, the input size additionally takes the size of the database query q and is given as $|\mathcal{T}| + |\mathcal{A}| + |q|$ [2].
- *Data complexity:* only the ABox $\mathcal{A}$ is considered part of the input and the input size is given as $|\mathcal{A}|$, while the remaining parameters are considered fixed and their size is considered constant.

2.4 Datalog

This section serves as a short introduction to Datalog, a database query language rooted in Prolog and born out of the desire to bring logic programming and database systems closer together. The motivation behind this was two-fold. In the mid 1970s, with the rise of Prolog, the logic programming community was facing the following issue: logic programs predominantly stored their rules and facts in the local memory, which quickly proved to be a less-than-optimal approach if a large number of facts was present. For this reason, the logic programming community started looking into storing facts using external relational databases[MTKW18]. At around the same time, SQL was being established as the database query language of choice. However, it was recognized early on that SQL suffered from limited expressiveness. In particular, SQL's lack of support for

[2]The size of the rest of the input, e.g., a pair of concept/role names or a tuple of individuals, is negligibly small compared to the size of the KB and is therefore usually discarded.

recursive queries meant that many natural queries were inexpressible, typical examples being reachability queries and computing the transitive closure [AU79]. This prompted a formal investigation into Datalog– a subset of Prolog that is more declarative and data-oriented in nature and reminiscent of the relational calculus that the database community was already familiar with [MTKW18].

Datalog *programs* consist of *facts* and *rules*. Facts are ground atoms that are assumed to be true. For example, consider the following facts:

$$\begin{aligned}\mathsf{PhDStudent}(\mathit{Barbara}),\\ \mathsf{advisedBy}(\mathit{Barbara}, \mathit{Charlie}),\\ \mathsf{PhDThesis}(\mathit{cthesis}, \mathit{Charlie}),\\ \mathsf{advised}(\mathit{Dave}, \mathit{cthesis}),\\ \mathsf{advised}(\mathit{Erika}, \mathit{cthesis}),\end{aligned}$$

expressing that Barbara is a PhD student who is advised by Charlie, and Charlie wrote their PhD thesis "cthesis" which was advised by Dave and Erika.

On the other hand, Datalog *rules* are inference rules that tell us how we can derive new atoms from the already derived ones. For example, the following is a Datalog rule saying that some person Y is an academic descendant of a person Y', if Y wrote a thesis that was supervised by Y':

$$\mathsf{academicDesc}(Y, Y') \leftarrow \mathsf{PhDThesis}(X, Y), \mathsf{advised}(Y', X).$$

Observe also that, unlike facts, rules may contain variables in order to make them more general. For example, the rule above is applicable whenever we have some constants t, p_1, p_2 such that $\mathsf{PhDThesis}(t, p_1)$ and $\mathsf{advised}(p_2, t)$ are either facts or have already been derived, and we can derive $\mathsf{academicDesc}(p_1, p_2)$.

If we further want to include the people who have not yet written their PhD thesis but are currently being advised by someone, we can add the following rule:

$$\mathsf{academicDesc}(X, Y) \leftarrow \mathsf{PhDStudent}(X), \mathsf{advisedBy}(X, Y).$$

For a rule ρ, we distinguish between the *body* and the *head* of ρ, occurring respectively on the right-hand side and the left-hand side of the $\leftarrow$ sign.

As already mentioned, Datalog offers simple support for recursion, by allowing rules to use the same predicate in both the body and the head. For example, we can compute the transitive closure of the 'academic descendant' relation by the following recursive rule:

$$\mathsf{academicDesc}(X, Z) \leftarrow \mathsf{academicDesc}(X, Y), \mathsf{academicDesc}(Y, Z).$$

There are different ways to assign semantics to Datalog programs, however, they all boil down to the same intuition: *models* of Datalog programs can be viewed simply as sets of ground atoms that are minimal in the sense that they only contain the original facts

occurring in the program as well as the atoms that were derived using program rules. For example, the following is a model of the program consisting of the rules and facts listed above:

$$\{\mathsf{PhDStudent}(\mathit{Barbara}), \mathsf{advisedBy}(\mathit{Barbara}, \mathit{Charlie}),$$
$$\mathsf{PhDThesis}(\mathit{cthesis}, \mathit{Charlie}), \mathsf{advised}(\mathit{Dave}, \mathit{cthesis}),$$
$$\mathsf{advised}(\mathit{Erika}, \mathit{cthesis}), \mathsf{academicDesc}(\mathit{Charlie}, \mathit{Dave}),$$
$$\mathsf{academicDesc}(\mathit{Charlie}, \mathit{Erika}), \mathsf{academicDesc}(\mathit{Barbara}, \mathit{Charlie}),$$
$$\mathsf{academicDesc}(\mathit{Barbara}, \mathit{Dave}), \mathsf{academicDesc}(\mathit{Barbara}, \mathit{Erika}\}.$$

This concludes our informal introduction to Datalog. It is already easy to see that in its core version, Datalog also suffers from limited expressiveness, especially due to the lack of negation. However, before we introduce the relevant extensions, we formalize the intuitions presented above.

2.4.1 Plain Datalog

We begin with the formal definitions of rules and programs.

Definition 2.4.1 (Rule, program)**.** *A* Datalog rule ρ *is an expression of the form*

$$h \leftarrow b_1, \ldots, b_n,$$

where $n \geq 0$, h, $b_1, \ldots, b_n$ *are atoms over* $\mathsf{N_P}$ *with terms from* $\mathsf{N_I} \cup \mathsf{N_V}$, *and all variables in* ρ *occur in some* $b_1, \ldots, b_n$. *We call* h *the* head *of* ρ, *denoted by* $\mathit{head}(\rho)$, *and the set* $\{b_1, \ldots, b_n\}$ *is the* body *of* ρ, *denoted by* $\mathit{body}(\rho)$. *If* ρ *contains no variables,* ρ *is called* ground. Facts *are ground rules of the form* $h \leftarrow$, *abbreviated simply by* h. *A* Datalog program *is a finite set of* Datalog *rules. A* Datalog *program is called* ground *if all its rules are ground.*

We point out that the definition above only allows for *safe rules*. i.e., the rules that do not contain variables in their head that do not also appear in their body. This is known as *Datalog safety criterion* and is usually required to ensure the decidability of the formalism.

Semantics In the context of Datalog, we usually talk about *Herbrand interpretations*, which are simplified representations of ordinary FO interpretations in the sense that they list all and only those ground atoms that are considered true.

Definition 2.4.2 (Herbrand interpretation)**.** *An* Herbrand interpretation *over some set of predicates* Σ *is a finite set of ground atoms over* Σ. *If* Σ *is not specified, we assume* $\Sigma = \mathsf{N_P}$.

We note that every ABox is an Herbrand interpretation, and also any Herbrand interpretation over $\mathsf{N_C} \cup \mathsf{N_R}$ is an ABox.

We now introduce the semantics of ground Datalog programs and then extend the definition to also include programs with variables. We have mentioned already that the semantics of Datalog programs is given in terms of sets of ground atoms (a.k.a. Herbrand interpretations) that are in some sense minimal. We next formally define what it means for an Herbrand interpretation to be a *model* and a *minimal model* of a ground program.

Definition 2.4.3. *Let I be an Herbrand interpretation and $\mathcal{P}$ be a ground Datalog program. We say that I is a model of $\mathcal{P}$ if $body(\rho) \subseteq I$ implies $head(\rho) \subseteq I$, for every rule $\rho \in \mathcal{P}$. I is called a* minimal model*, or an* answer set[3] *of $\mathcal{P}$, if there is no $J \subsetneq I$ such that J is a model of $\mathcal{P}$.*

The semantics of Datalog programs with variables is defined much in the same way as the semantics of ground programs but it involves an extra preprocessing step called *grounding* in which we replace each non-ground rule ρ with the set of its *ground instances*, i.e., rules that are obtained from ρ by uniformly substituting constants for the variables in ρ.

Definition 2.4.4 (Grounding)**.** *Given a set of constants $C \subseteq \mathsf{N_I}$, the* grounding of a rule ρ w.r.t. C*, in symbols $ground(\rho, C)$, is a set of rules obtained from ρ by replacing every variable x in ρ with $\sigma(x)$, for every total function $\sigma : vars(\mathcal{P}) \to C$. The* grounding of a program $\mathcal{P}$ w.r.t. C *is then $ground(\mathcal{P}, C) = \bigcup_{\rho \in \mathcal{P}} ground(\rho, C)$.*

Grounding is normally done with respect to the set of all constants that occur in the program which is also how the semantics of the non-ground programs is defined. For a program $\mathcal{P}$, this set is called the *active domain* of the $\mathcal{P}$ and we denote it by $adom(\mathcal{P})$. We note that however, other purposes might require us to ground programs over different constants, which is the reason why we keep the definition above more general than usual. We can now define models of non-ground programs as follows:

Definition 2.4.5 (Model, minimal model, answer set)**.** *Let $\mathcal{P}$ be Datalog program and I be an Herbrand interpretation. We say that I is a model of $\mathcal{P}$ if I is a* model *of $ground(\mathcal{P}, adom(\mathcal{P}))$, and I is a* minimal model*, or an* answer set *of $\mathcal{P}$, if it is a minimal model of $ground(\mathcal{P}, adom(\mathcal{P}))$.*

It is well-known that Datalog programs have unique minimal models [Kol91].

Proposition 2.4.6. *Let $\mathcal{P}$ be a Datalog program. Then, $\mathcal{P}$ has a unique minimal model, denoted by $MM(\mathcal{P})$.*

Finally, we note that there are generally three well-established approaches for assigning semantics to Datalog programs which have been proven equivalent: the *model-theoretic*

[3] This is done to keep in line with the terminology that will be introduced later in this section when we discuss non-monotonic extensions of Datalog.

approach that we presented above, as well as the *fixed-point* and *proof-theoretic* approach. The latter two approaches are not discussed here, but more details can be found in [AHV95].

2.4.2 Non-monotonic Extensions of Datalog

As already mentioned, plain Datalog does not provide support for reasoning with negative information, which is often desired and also possible in standard database query languages such as relational algebra. For example, we cannot compute using plain Datalog the relation consisting of pairs of persons that are "academically unrelated", i.e., neither is an academic descendent of the other. To overcome this, we next present Datalog$^\neg$, the extension of Datalog with negation under the stable model semantics [GL88]. Intuitively, Datalog$^\neg$ allows us to additionally use *default negation* in front atoms in rule bodies, with the meaning that if this atom cannot be established to be true then its negation is assumed. We can then compute the desired relation as:

$$\mathsf{unrelated}(X,Y) \leftarrow \mathsf{Person}(X), \mathsf{Person}(Y), \\ \mathit{not}\ \mathsf{academicDesc}(X,Y), \mathit{not}\ \mathsf{academicDesc}(Y,X), X \neq Y.^4$$

The addition of negation results in a powerful nonmonotonic language that underlies Answer Set Programming (ASP) [EIK09], a declarative programming paradigm designed for solving combinatorial problems by expressing them in terms of logic-based rules and constraints.

We next formally introduce the syntax and the semantics of Datalog$^\neg$.

Definition 2.4.7 (Rule, program)**.** *A* Datalog$^\neg$ rule ρ *is an expression of the form*

$$h \leftarrow b_1, \ldots, b_n, \mathit{not}\ b_{n+1}, \ldots, \mathit{not}\ b_m$$

where $n, m \geq 0$, h, $b_1, \ldots, b_m$ *are atoms over* $\mathsf{N_P}$ *with terms from* $\mathsf{N_I} \cup \mathsf{N_V}$, *and all variables in* ρ *occur in some* $b_1, \ldots, b_n$. *A* Datalog$^\neg$ program *is a finite set of* Datalog$^\neg$ *rules.*

For a Datalog$^\neg$ rule ρ, $\mathit{head}(\rho)$ and $\mathit{body}(\rho)$ are defined as before, but we additionally distinguish the *positive body* of ρ given as $\mathit{body}^+(\rho) = \{b_1, \ldots b_n\}$, and the *negative body* of ρ given as $\mathit{body}^-(\rho) = \{b_{n+1}, \ldots, b_m\}$. Observe that the safety criterion was also updated to ensure that all rule variables occur in its positive body. If $\mathit{body}^-(\rho) = \emptyset$, ρ is a *positive rule*. A Datalog$^\neg$ program is *positive* if all its rules are positive.

Semantics. Following [GL88], the semantics to Datalog$^\neg$ programs is given in terms of *stable models*, also called *answer sets*. Roughly speaking, stable models of some Datalog$^\neg$ program $\mathcal{P}$ are those models I of $\mathcal{P}$ that have the following property: if we interpret all

[4] One often assumes the availability of built-in comparison predicates $=$ and $\neq$, since they can be easily axiomatized in Datalog$^\neg$.

the negated atoms as specified by I and we simplify $\mathcal{P}$ accordingly, then I is a minimal model of the obtained program. We simplify $\mathcal{P}$ with respect I by doing the following: (i) we first delete all rules that do not contribute to the derivation of new information as they are "blocked" due to some *not* α in their body with $\alpha \in I$, and (ii) we then delete all negated atoms from the remaining rules since they have already been interpreted as true. The intuition of simplifying the program w.r.t. a given interpretation has been captured in the notion of a *reduct*.

Definition 2.4.8 (Reduct)**.** *The* reduct *of a* $\mathsf{Datalog}^{\neg}$ *program* $\mathcal{P}$ *w.r.t. to an Herbrand interpretation* I *is a ground positive program* $\mathcal{P}^I$ *defined as*

$$\mathcal{P}^I = \{head(\rho) \leftarrow body^+(\rho) : body^-(\rho) \cap I = \emptyset, \rho \in ground(\mathcal{P}, adom(\mathcal{P}))\}.$$

Definition 2.4.9 (Stable models)**.** *We say that an Herbrand interpretation* I *is a* stable model *(or an* answer set*) of a* $\mathsf{Datalog}^{\neg}$ *program* $\mathcal{P}$ *if* I *is a minimal model of* $\mathcal{P}^I$*.*

We next give a short example to illustrate the most important concepts introduced in this section.

Example 2.4.10. *Consider the* $\mathsf{Datalog}^{\neg}$ *program* $\mathcal{P}$ *consisting of the following rules:*

$$\begin{aligned}\mathcal{P} = \{\ & \mathsf{Person}(Charlie),\\ & \mathsf{Person}(Dave),\\ & \mathsf{Person}(Erika),\\ & \mathsf{advisedBy}(Charlie, Dave),\\ & \mathsf{advisedBy}(Charlie, Erika)\\ \mathsf{unrelated}(X,Y) \leftarrow\ & \mathsf{Person}(X), \mathsf{Person}(Y), not\ \mathsf{advisedBy}(X,Y),\\ & not\ \mathsf{advisedBy}(Y,X), X \neq Y\}.\end{aligned}$$

Furthermore, let I *be the following Herbrand interpretation:*

$$\begin{aligned}I = \{\ & \mathsf{Person}(Charlie), \mathsf{Person}(Dave), \mathsf{Person}(Erika),\\ & \mathsf{advisedBy}(Charlie, Dave), \mathsf{advisedBy}(Charlie, Erika),\\ & \mathsf{unrelated}(Dave, Erika), \mathsf{unrelated}(Erika, Dave)\}.\end{aligned}$$

To check whether I *is an* answer set *of* $\mathcal{P}$*, we need to compute the reduct of* $\mathcal{P}$ *w.r.t.* I*, and for this, we first need to ground* $\mathcal{P}$ *over its active domain. In this case, we have:*

$$adom(\mathcal{P}) = \{Charlie, Dave, Erika\}.$$

The grounding is then done by replacing non-ground rules with the set of their ground instances. In our case, we replace the last rule in $\mathcal{P}$ *with the following set of rules:*

$$\begin{aligned}
\mathsf{unrelated}(Charlie, Charlie) \leftarrow\ & \mathsf{Person}(Charlie), \mathsf{Person}(Charlie), not\ \mathsf{advisedBy}(Charlie, Charlie),\\
& not\ \mathsf{advisedBy}(Charlie, Charlie), Charlie \neq Charlie,\\
\mathsf{unrelated}(Charlie, Dave) \leftarrow\ & \mathsf{Person}(Charlie), \mathsf{Person}(Dave), not\ \mathsf{advisedBy}(Charlie, Dave),\\
& not\ \mathsf{advisedBy}(Dave, Charlie), Charlie \neq Dave,\\
\mathsf{unrelated}(Charlie, Erika) \leftarrow\ & \mathsf{Person}(Charlie), \mathsf{Person}(Erika), not\ \mathsf{advisedBy}(Charlie, Erika),\\
& not\ \mathsf{advisedBy}(Erika, Charlie), Charlie \neq Erika,\\
\mathsf{unrelated}(Dave, Charlie) \leftarrow\ & \mathsf{Person}(Dave), \mathsf{Person}(Charlie), not\ \mathsf{advisedBy}(Dave, Charlie),\\
& not\ \mathsf{advisedBy}(Charlie, Dave), Dave \neq Charlie,\\
\mathsf{unrelated}(Dave, Dave) \leftarrow\ & \mathsf{Person}(Dave), \mathsf{Person}(Dave), not\ \mathsf{advisedBy}(Dave, Dave),\\
& not\ \mathsf{advisedBy}(Dave, Dave), Dave \neq Dave,\\
\mathsf{unrelated}(Dave, Erika) \leftarrow\ & \mathsf{Person}(Dave), \mathsf{Person}(Erika), not\ \mathsf{advisedBy}(Dave, Erika),\\
& not\ \mathsf{advisedBy}(Erika, Dave), Dave \neq Erika,\\
\mathsf{unrelated}(Erika, Charlie) \leftarrow\ & \mathsf{Person}(Erika), \mathsf{Person}(Charlie), not\ \mathsf{advisedBy}(Erika, Charlie),\\
& not\ \mathsf{advisedBy}(Charlie, Erika), Erika \neq Charlie,\\
\mathsf{unrelated}(Erika, Dave) \leftarrow\ & \mathsf{Person}(Erika), \mathsf{Person}(Dave), not\ \mathsf{advisedBy}(Erika, Dave),\\
& not\ \mathsf{advisedBy}(Dave, Erika), Erika \neq Dave.\\
\mathsf{unrelated}(Erika, Erika) \leftarrow\ & \mathsf{Person}(Erika), \mathsf{Person}(Erika), not\ \mathsf{advisedBy}(Erika, Erika),\\
& not\ \mathsf{advisedBy}(Erika, Erika), Erika \neq Erika.
\end{aligned}$$

Observe that we replaced a single non-ground rule with the set of eight ground rules, which corresponds to $art(\mathsf{advisedBy})^{|adom(\mathcal{P})|}$. *It is well-known that computing the grounding of a program generally causes an exponential blowup and is a major bottleneck for reasoning in* Datalog *and its extensions, which can in some cases be overcome through different optimization methods [LPF+ 06].*

The next step is to remove all the rules ρ *from* $ground(\mathcal{P}, adom(\mathcal{P}))$ *that contain some negated atom* $not\ \alpha$ *such that* $\alpha \in I$. *For example, the rule*

$$\begin{aligned}
\mathsf{unrelated}(Charlie, Dave) \leftarrow\ & \mathsf{Person}(Charlie), \mathsf{Person}(Dave),\\
& not\ \mathsf{advisedBy}(Charlie, Dave), not\ \mathsf{advisedBy}(Dave, Charlie),\\
& Charlie \neq Dave,
\end{aligned}$$

will be deleted as it contains $not\ \mathsf{advisedBy}(Charlie, Dave)$, *but* $\mathsf{advisedBy}(Charlie, Dave) \in I$. *We note that the expression* $a \neq a$ *can be viewed as not* $a = a$. *Since we treat* $=$ *as a built-in predicate, we do not explicitly specify in* I *which pairs of constants participate in this predicate, however, we assume that* $(a = a) \in I$, *for all individuals* a *in the active domain of* $\mathcal{P}$.

This yields the program that consists of all the facts in $\mathcal{P}$ as well as the following two rules:

$$\begin{aligned}\mathsf{unrelated}(Dave, Erika) \leftarrow\ & \mathsf{Person}(Dave), \mathsf{Person}(Erika), \mathit{not}\ \mathsf{advisedBy}(Dave, Erika),\\ & \mathit{not}\ \mathsf{advisedBy}(Erika, Dave), Dave \neq Erika,\\ \mathsf{unrelated}(Erika, Dave) \leftarrow\ & \mathsf{Person}(Erika), \mathsf{Person}(Dave), \mathit{not}\ \mathsf{advisedBy}(Erika, Dave),\\ & \mathit{not}\ \mathsf{advisedBy}(Dave, Erika), Erika \neq Dave.\end{aligned}$$

Finally, we remove all negative atoms from this program to obtain $\mathcal{P}^{\mathcal{I}}$:

$$\begin{aligned}\mathcal{P}^{\mathcal{I}} = \{\ & \mathsf{Person}(Charlie),\\ & \mathsf{Person}(Dave),\\ & \mathsf{Person}(Erika),\\ & \mathsf{advisedBy}(Charlie, Dave),\\ & \mathsf{advisedBy}(Charlie, Erika),\\ \mathsf{unrelated}(Dave, Erika) \leftarrow\ & \mathsf{Person}(Dave), \mathsf{Person}(Erika),\\ \mathsf{unrelated}(Erika, Dave) \leftarrow\ & \mathsf{Person}(Erika), \mathsf{Person}(Dave)\}.\end{aligned}$$

It is easy to see that I is indeed a minimal model of $\mathcal{P}^I$, and thus I is an answer set of $\mathcal{P}$.

Consider now another Herbrand interpretation J:

$$\begin{aligned}J = \{\ & \mathsf{Person}(Charlie), \mathsf{Person}(Dave), \mathsf{Person}(Erika),\\ & \mathsf{advisedBy}(Charlie, Dave), \mathsf{advisedBy}(Charlie, Erika)\}.\end{aligned}$$

Following the same steps as before, we compute $\mathcal{P}^J$:

$$\begin{aligned}\mathcal{P}^{J} = \{\ & \mathsf{Person}(Charlie),\\ & \mathsf{Person}(Dave),\\ & \mathsf{Person}(Erika),\\ & \mathsf{advisedBy}(Charlie, Dave),\\ & \mathsf{advisedBy}(Charlie, Erika),\\ \mathsf{unrelated}(Dave, Erika) \leftarrow\ & \mathsf{Person}(Dave), \mathsf{Person}(Erika),\\ \mathsf{unrelated}(Erika, Dave) \leftarrow\ & \mathsf{Person}(Erika), \mathsf{Person}(Dave)\}.\end{aligned}$$

We can see that J is not a model of $\mathcal{P}^J$, and thus J is not an answer set of $\mathcal{P}$.

Note that unlike Datalog programs, Datalog$^{\neg}$ programs may have no, one, or multiple answer sets, as illustrated in the following example.

Example 2.4.11. *Consider the following ground program:*

$$\mathcal{P} = \{\ \mathit{advisedBy}(\mathit{Charlie}, \mathit{Erika}) \leftarrow \mathit{not}\ \mathit{advisedBy}(\mathit{Charlie}, \mathit{Dave}),$$
$$\mathit{advisedBy}(\mathit{Charlie}, \mathit{Dave}) \leftarrow \mathit{not}\ \mathit{advisedBy}(\mathit{Charlie}, \mathit{Erika})\}.$$

The program $\mathcal{P}$ has the following two answer sets:

$$I_1 = \{\mathit{advisedBy}(\mathit{Charlie}, \mathit{Dave})\}, \qquad I_2 = \{\mathit{advisedBy}(\mathit{Charlie}, \mathit{Erika})\}.$$

We next introduce some notions related to Datalog and its extensions that will be used throughout this thesis. In what follows, unless specified otherwise, we use *program* and *rule* to denote a program or a rule written in any of the presented extensions of Datalog.

Datalog$^{\vee,\neg}$ Another natural extension of Datalog is called Datalog$^{\vee}$ and it allows us to write disjunctions of atoms in rule heads. If we additionally allow negation in the rule bodies, we obtain Datalog$^{\vee,\neg}$, whose rules have the following form:

$$\rho := h_1 \vee \cdots \vee h_l \leftarrow b_1, \ldots, b_n, \mathit{not}\ b_{n+1}, \ldots, \mathit{not}\ b_m,$$

where $l \geq 1, n, m \geq 0$, $h_1, \ldots, h_l, b_1, \ldots, b_m$ are atoms over $\mathsf{N_P}$ with terms from $\mathsf{N_I} \cup \mathsf{N_V}$, and all variables in ρ occur in one of $b_1, \ldots, b_n$. We call the set $\{h_1, \ldots, h_l\}$ the *head* of ρ, denoted by $\mathit{head}(\rho)$, while the notions of the positive and negative body remain the same.

We say that an Herbrand interpretation satisfies a positive Datalog$^{\vee,\neg}$ rule $h_1 \vee \cdots \vee h_l \leftarrow b_1, \ldots, b_n$, if whenever $\{b_1, \ldots, b_n\} \subseteq I$, then $\{h_1, \ldots, h_l\} \cap I \neq \emptyset$. With this in mind, the rest of the semantics is defined in the same way as before.

Constraints. Sometimes a program $\mathcal{P}$ can contain rules that are not used to derive new information but rather function as *integrity constraints*, ensuring that certain situations do not occur in the answer sets of $\mathcal{P}$. Such rules are called *constraints* and have the form $p \leftarrow \alpha, \mathit{not}\ p$ (abbreviated as $\leftarrow \alpha$), where p is a fresh propositional atom that does not occur elsewhere in $\mathcal{P}$. For example, we can write the following constraint to ensure that in the answer sets of $\mathcal{P}$ no person is considered unrelated to themselves.

$$\leftarrow \mathsf{unrelated}(X, X), \mathsf{Person}(X).$$

Strong negation. In general, there are two different ways to support negation. We have already covered *default negation*, which allows us to infer the negation of α, i.e., *not* α if we cannot derive α. However, we can also consider *strong* (or *classical*) *negation*, denoted by $\neg$. In this case, in order to infer a strongly negated atom $\neg\alpha$, we need to have a justification for it, i.e., it needs to be derived from the facts by using the program rules.

While some extensions of Datalog allow for the explicit use of it, strong negation is ultimately simply syntactic sugar and can be expressed as follows: we replace a strongly

negated atom $\neg P(\vec{a})$ by $\overline{P}(\vec{a})$, where $\overline{P}$ is a predicate that does not occur elsewhere in the program, and an additional constraint $\leftarrow P(\vec{a}), \overline{P}(\vec{a})$, to ensure that in no answer set of the program both $P(\vec{a})$ and $\overline{P}(\vec{a})$ hold. In this thesis, we readily make use of this trick and therefore do not consider programs with support for strong negation.

Modularity. Depending on the problem at hand, programs can get very large, and verifying their correctness may be difficult. Therefore, it can be helpful to split a large program into smaller chunks, called *modules* or *components*, where the output of one module can be used as input to another. The following generalization of Lemma 5.1 in [EGM97] allows us to define programs in a modular way.

Proposition 2.4.12. *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two* Datalog$^{\vee,\neg}$ *programs with the property that all shared predicates are EDB predicates of $\mathcal{P}_2$ and all constants of $\mathcal{P}_2$ occur in $\mathcal{P}_1$. The answer sets of $\mathcal{P}_1 \cup \mathcal{P}_2$ coincide with the set*

$$\{I : I \text{ is an answer set of } P_2 \cup J, \text{ for some answer set } J \text{ of } P_1\}.$$

External Database. Sometimes it is also convenient to view programs as two separate components: (i) the part $\mathcal{P}$ consisting of all rules and facts over IDB predicates and (ii) an external *database* D containing facts over EDB predicates. If the distinction is important, we may write $(\mathcal{P}, D)$ to denote such a program, which is semantically equivalent to $\mathcal{P} \cup D$. We also point out that D is nothing else but an Herbrand interpretation over the EDB predicates.

Queries. Datalog is primarily a database *query language* and as such it should provide a way to answer queries over databases. We next define the notion of a *Datalog query*.

Definition 2.4.13 (Datalog query)**.** *A* Datalog query *is a pair $(\mathcal{P}, Q)$, where $\mathcal{P}$ is a* Datalog *program and Q is a distinguished predicate occurring in $\mathcal{P}$.*

The answers to a query $(\mathcal{P}, Q)$ over some database D are given as all n-tuples $\vec{a}$ of constants, where n is the arity of Q, for which the program $\mathcal{P} \cup D$ entails $Q(\vec{a})$, i.e., $Q(\vec{a})$ is contained in the minimal model of $\mathcal{P} \cup D$.

Definition 2.4.14. *Let $(\mathcal{P}, Q)$ be a* Datalog *query and D be a database over the EDB predicates of $\mathcal{P}$. The set of* answers *to $(\mathcal{P}, Q)$ over D is denoted by $ans((\mathcal{P}, Q), D)$ and computed as follows:*

$$ans((\mathcal{P}, Q), D) = \{\vec{a} \in \mathsf{N_I}^{art(Q)} : Q(\vec{a}) \in MM(\mathcal{P} \cup D)\}.$$

Syntactically, queries in the extensions of Datalog are defined in the same way. However, since such programs may have more than one answer set, the question of which answer set should be considered when answering a query arises. In general, there are different ways to define answers to queries in non-monotonic extensions of Datalog, two of the most popular being *brave* and *cautious reasoning*.

Definition 2.4.15 (Brave/cautious entailment)**.** *We say that a program $\mathcal{P}$* bravely entails *and atom α if $\alpha \in I$, for* some *answer set I of $\mathcal{P}$. On the other hand, $\mathcal{P}$* cautiously entails *α, if $\alpha \in I$, for* every *answer set of $\mathcal{P}$.*

Depending on what type of entailment we consider, we get two different notions of answers: *possible answers* and *certain answers*.

Definition 2.4.16 (Possible/certain answers)**.** *Let $(\mathcal{P}, Q)$ be a $\mathsf{Datalog}^{\vee,\neg}$ query and D be a database over the EDB predicates of $\mathcal{P}$. We say that a tuple of constants $\vec{a} \in \mathsf{N_I}^{art(Q)}$ is a* possible answer *to $(\mathcal{P}, Q)$ over D, if $\mathcal{P} \cup D$ bravely entails $Q(\vec{a})$. We say that $\vec{a}$ is a* certain answer *to $(\mathcal{P}, Q)$ over D, if $\mathcal{P} \cup D$ cautiously entails $Q(\vec{a})$.*

In this thesis, we only consider cautious reasoning and certain answer semantics. We denote the set of all certain answers to a query $(\mathcal{P}, Q)$ over a database D by $cert((\mathcal{P}, Q), D)$.

Example 2.4.17. *Let $\mathcal{P}$ be a $\mathsf{Datalog}^{\neg}$ program consisting of the following rules:*

$$\begin{aligned}\mathcal{P} = \{\ & \mathsf{advisedBy}(Charlie, Dave) \leftarrow not\ \mathsf{advisedBy}(Charlie, Erika)\\ & \mathsf{advisedBy}(Charlie, Erika) \leftarrow not\ \mathsf{advisedBy}(Charlie, Dave)\\ & \mathsf{unrelated}(X, Y) \leftarrow\ \mathsf{Person}(X), \mathsf{Person}(Y), not\ \mathsf{advisedBy}(X, Y),\\ & not\ \mathsf{advisedBy}(Y, X), X \neq Y\}.\end{aligned}$$

Furthermore, let D be the following database:

$$D = \{\mathsf{Person}(Charlie), \mathsf{Person}(Dave), \mathsf{Person}(Erika)\}.$$

The program $\mathcal{P} \cup D$ has two answer sets:

$$\begin{aligned}I_1 &= D \cup \{\mathsf{advisedBy}(Charlie, Dave, \mathsf{unrelated}(Dave, Erika), \mathsf{unrelated}(Erika, Dave)\}\\ I_2 &= D \cup \{\mathsf{advisedBy}(Charlie, Erika, \mathsf{unrelated}(Dave, Erika), \mathsf{unrelated}(Erika, Dave)\}.\end{aligned}$$

Consider the query $(\mathcal{P}, \mathsf{unrelated})$. The certain and the possible answers to this query over D coincide: $(Dave, Erika)$ and $(Erika, Dave)$.

Consider now the query $(\mathcal{P}, \mathsf{advisedBy})$. This query has two possible answers over D, $(Charlie, Dave)$ and $(Charlie, Erika)$, but it has no certain answers over D.

Complexity of reasoning. We now briefly recall the most relevant reasoning tasks in Datalog and its extensions together with their complexity. In this thesis, we are mostly concerned with the following reasoning tasks:

- **Consistency:** does a given program have at least one answer set?
- **Answer set checking:** given a program $\mathcal{P}$ and an Herbrand interpretation I, is I an answer set of $\mathcal{P}$?

	Consistency	Answer set checking	Cautious query answering
Datalog	trivial	PTIME	PTIME
Datalog$^{\neg}$	NP	PTIME	co-NP
Datalog$^{\vee,\neg}$	NP	co-NP	Π^2_P

Table 2.2: Selected complexity results of standard reasoning tasks with ground programs in Datalog and its extensions.

	Consistency		Answer set checking		Cautious query answering	
	data	combined	data	combined	data	combined
Datalog	trivial	trivial	PTIME	PTIME	PTIME	EXPTIME
Datalog$^{\neg}$	NP	NEXPTIME	PTIME	PTIME	co-NP	co-NEXPTIME
Datalog$^{\vee,\neg}$	NP	NEXPTIME	co-NP	co-NP	Π^2_P	co-NEXPTIME $^{\text{NP}}$

Table 2.3: Selected complexity results of standard reasoning tasks with non-ground programs in Datalog and its extensions.

- **Cautious query answering:** given a query $(\mathcal{P}, Q)$, a database D and a tuple of constants $\vec{a}$, is $\vec{a}$ a certain answer to $(\mathcal{P}, Q)$ over D?

Table 2.2 summarizes known complexity results of the problems above for ground programs written in different dialects of Datalog (for details, see [LPF$^+$06] and references therein).

For non-ground programs, we once again distinguish between *data* and *combined complexity*. Generally, allowing variables in programs raises the complexity by one exponential due to the grounding step. For data complexity, we typically view programs as a pair $(\mathcal{P}, D)$ of rules and an external database, as previously explained. The input size is then simply given as $|D|$, while $\mathcal{P}$ and other relevant parameters are considered fixed and their size is reduced to a constant. We summarize in Table 2.3 known complexity results for non-ground programs in different dialects of Datalog. Observe that in general, data complexity of reasoning tasks for non-ground programs coincides with the complexity of ground programs, as the grounding step no longer causes an exponential explosion.

CHAPTER 3

Expressive DLs with Closed Predicates

Equipping DLs with the possibility of specifying which part of the signature should be regarded as *complete* is a relatively simple way of accommodating partial closed-world reasoning in DLs. Nevertheless, OMQ languages based on DLs that support closed predicates exhibit *non-monotonic* behavior which allows us to express many natural queries that cannot be expressed in traditional query languages like first-order logic or Datalog. One such example is the famous *parity query* Q_{parity} – given an ABox (or a database) and a unary relation A, is the number of objects in A odd? This query is obviously non-monotonic – e.g. the answer to Q_{parity} over the ABox $\mathcal{A}_1 = \{A(a)\}$ is 1 (or "true") but it is 0 (or false) over $\mathcal{A}_2 = \mathcal{A}_1 \cup \{A(b)\}$. We will see a little later how we can express this query as an OMQ with the help of closed predicates. We next formally introduce *description logics with closed predicates*, with the focus on the very expressive $\mathcal{ALCHOIQ}$ that simultaneously supports nominals, inverses, and number restrictions. As already mentioned, this particular combination is known to be computationally challenging and it increases the complexity of basic reasoning problems from ExpTime to NExpTime [Tob00]. Complexity-optimal algorithms for DLs supporting at the same time inverses, nominals, and number restrictions, all rely on some type of characterization of the KB satisfiability problem as an integer linear programming problem. In this chapter, we follow this line of work and we provide a characterization of the KB satisfiability problem of $\mathcal{ALCHOIQ}$ with closed predicates as an integer programming problem with some side conditions, that serves as the basis for showing the results in the later chapters. In particular, our approach was inspired by works on finite model reasoning in DLs. Building on the technique from [CLN94], Calvanese shows in [Cal96] that satisfiability and subsumption w.r.t. TBoxes in $\mathcal{ALCIQ}$ can be decided in 2-ExpTime by constructing a system of linear inequalities from a given knowledge base, and relating the existence of its solutions to the existence of finite models of the considered knowledge base. The authors

of [LST05] improve Calvanese's bound by showing that these reasoning problems are in fact ExpTime-complete for $\mathcal{ALCIQ}$, for both unary and binary encoding of integers. Their line of reasoning is also based on the same core idea – due to the interplay of inverses and number restrictions, solving combinatorial issues is an important aspect of deciding finite satisfiability in $\mathcal{ALCIQ}$, and such problems can be addressed using a suitable integer programming characterization. More recently, a similar technique was used in [GGI+20] to investigate *mixed satisfiability* of $\mathcal{ALCHOIF}$ knowledge bases, a generalization of standard satisfiability problems which requires that some predicates have finite extensions, while others are unrestricted. Finally, in the area of first-order logic, [PH05] show that both the finite and the unrestricted satisfiability problem for the two-variable fragment of FO logic with counting quantifiers is decidable in NExpTime using a reduction to integer programming, and [PH09] shows that these problems are NP-complete in data complexity.

Contributions and Relevant Publications The main contribution of this chapter is a worst-case optimal satisfiability procedure for $\mathcal{ALCHOIQ}$ with closed predicates. More precisely, we show how to encode the knowledge base satisfiability problem for this logic as a system of integer linear inequalities enriched with implications that is i) polynomial in the size of the ABox and exponential in the size of the TBox, and ii) has a solution if and only if the corresponding KB admits a model. The existence of the solution to the obtained system can be decided using established integer programming techniques, yielding a novel NP upper bound for the considered reasoning task.

The results presented here have been published in:

> [LOŠ24] **Sanja Lukumbuzya**, Magdalena Ortiz, Mantas Šimkus. "Datalog Rewritability and Data Complexity of $\mathcal{ALCHOIQ}$ with closed predicates". Artificial Intelligence (2024): 104099.

This journal paper was in turn based on the following two conference papers:

[GLOŠ20] Tomasz Gogacz, **Sanja Lukumbuzya**, Magdalena Ortiz, and Mantas Šimkus. "Datalog rewritability and data complexity of $\mathcal{ALCHOIF}$ with closed predicates". In Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning, KR 2020, vol. 17, no. 1, pp. 434-444. 2020.

Using the same integer programming technique, this paper presented a simpler characterization for $\mathcal{ALCHOIF}$, the fragment of $\mathcal{ALCHOIQ}$ that in terms of number restrictions allows only global functionality.

[LŠ21] **Sanja Lukumbuzya**, and Mantas Šimkus. "Bounded Predicates in Description Logics with Counting". In Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, pp. 1966-1972. 2021.

In the main body of this paper, we presented an integer encoding for the *mixed satisfiability problem* for $\mathcal{ALCHOIQ}$ TBoxes, i.e., the problem of deciding whether a TBox has a model, where certain predicates are required to have finite extensions. In the appendix to the same paper, we showed how this encoding can be extended to reasoning in the presence of data and closed predicates.

Organization Section 3.1 formally introduces description logics with closed predicates as well as the normal form for $\mathcal{ALCHOIQ}$ with closed predicates that will be used throughout the rest of this thesis. The main technical result is presented in Section 3.2 and, for easier readability, it is divided into five distinct parts. In Section 3.2.1, we introduce the notion of chromatic models that satisfy certain properties that make our characterization easier and we show that we can safely restrict our attention to only such models. In Section 3.2.2, we present the actual characterization of the considered satisfiability problem in terms of the existence of mosaics, i.e., functions that tell us how many domain elements of a certain kind we need in order to build a model and we show in Section 3.2.3 the correctness of this characterization. In Section 3.2.4, we introduce enriched systems of integer linear inequalities that extend ordinary systems of linear inequalities with the set of implications between inequalities and we recall some results on the complexity of solving such systems. A connection between KB satisfiability and integer programming is made in Section 3.2.5. Finally, in Section 3.3, we present a version of our previous characterization that was specifically tailored to $\mathcal{ALCHOIF}$ with closed predicates.

3.1 DLs with Closed Predicates

As previously mentioned, DL knowledge bases with closed predicates differ syntactically from plain DL KBs in that they additionally contain a set of predicates Σ specifying the part of the signature that is considered complete. Semantically, the set of models of such KBs is restricted to those interpretations that satisfy the underlying KB and agree with the ABox on the extensions of the predicates in Σ. We illustrate this with a short example.

Example 3.1.1. *Recall $\mathcal{K}_2 = (\mathcal{T}_2, \mathcal{A}_2)$ from Example 2.3.6, and consider its variation $\mathcal{K}_2' = (\mathcal{T}_2, \{\mathsf{Curriculum}, \mathsf{Course}\}, \mathcal{A}_2)$ with closed predicates that additionally states that the information we have about the available curricula and courses is complete, i.e., $\mathsf{Curriculum}$ and Course are closed predicates. We note that $\mathcal{K}_2'$ is expressed in $\mathcal{ALC}(\mathcal{H})\mathcal{OIQ}$ with closed predicates.*

Next, recall the model $\mathcal{I}$ of $\mathcal{K}_2$ from Example 2.3.10. Observe that this interpretation is not a model of $\mathcal{K}_2'$ since it violates the closed predicate Course, as $c_i \in \mathsf{Course}^{\mathcal{I}}$ and $\mathsf{Course}(c_i) \notin \mathcal{A}_2$, for $1 \leq i \leq 4$. In fact, $\mathcal{K}_2'$ admits no models at all – both Course and $\mathsf{Curriculum}$ are closed, the models of this KB must consist of exactly one curriculum and

exactly one course. This is however inconsistent with the axiom

$$\mathsf{Curriculum} \sqsubseteq \geq 5\,\mathsf{hasPart}.\mathsf{Course} \sqcap \leq 20\,\mathsf{hasPart}.\mathsf{Course},$$

requiring that every curriculum consists of at least five and at most 20 courses. However, $\mathcal{I}$ *is a model of* $(\mathcal{T}_2, \{\mathsf{Curriculum}, \mathsf{Course}\}, \mathcal{A}_2 \cup \{c_1, c_2, c_3, c_4\})$.

We next give formal definitions of the syntax and the semantics of description logic knowledge bases with closed predicates, without focusing on a specific logic.

Definition 3.1.2. *Let* $\mathcal{T}$ *be a TBox,* $\Sigma \subseteq \mathsf{N_C} \cup \mathsf{N_R}$ *be a set of predicates, and* $\mathcal{A}$ *be an ABox. The triple* $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$ *is a* knowledge base with closed predicates.

For some DL $\mathcal{L}$, we say that $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$ is expressed in $\mathcal{L}$ with closed predicates, if $(\mathcal{T}, \mathcal{A})$ is an $\mathcal{L}$ knowledge base. In what follows, if it is clear that we are in the setting of description logics with closed predicates, we refer to knowledge bases with closed predicates simply as *knowledge bases (KBs)*.

Definition 3.1.3. *Let* $\mathcal{I}$ *be an interpretation,* $\Sigma \subseteq \mathsf{N_C} \cup \mathsf{N_R}$ *be a set of closed predicates and* $\mathcal{A}$ *be an ABox. We say that* $\mathcal{I}$ satisfies $\mathcal{A}$ *w.r.t.* Σ*, in symbols* $\mathcal{I} \models_\Sigma \mathcal{A}$*, if:*

- $\mathcal{I} \models \mathcal{A}$,
- *for every* $A \in \Sigma \cap \mathsf{N_C}$, $b \in A^\mathcal{I}$ *implies* $A(b) \in \mathcal{A}$, *and*
- *for every* $r \in \Sigma \cap \mathsf{N_R}$, $(a, b) \in r^\mathcal{I}$ *implies* $r(a, b) \in \mathcal{A}$.

We say that $\mathcal{I}$ satisfies *a knowledge base with closed predicates* $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$*, in symbols* $\mathcal{I} \models \mathcal{K}$*, if* $\mathcal{I} \models \mathcal{T}$ *and* $\mathcal{I} \models_\Sigma \mathcal{A}$*. If* $\mathcal{I}$ *satisfies* $\mathcal{K}$*,* $\mathcal{I}$ *is a* model *of* $\mathcal{K}$*. We say that* $\mathcal{K}$ *is* satisfiable *(or* consistent*) if it has at least one model, otherwise* $\mathcal{K}$ *is* unsatisfiable *(or* inconsistent*).*

We remind the reader that all DLs in this thesis, as well as their extensions, make the standard name assumption that forces the models to interpret the knowledge base individuals as themselves. We also briefly recall some useful notations that were introduced in the preliminaries to this thesis. For a TBox, an ABox, or a KB $\mathcal{X}$, $\mathsf{N_C}(\mathcal{X})$, $\mathsf{N_R}(\mathcal{X})$, and $\mathsf{N_I}(\mathcal{X})$ denote the set of concept names, role names and individuals occurring in $\mathcal{X}$, respectively. Furthermore $\mathsf{N_C^+}(\mathcal{X}) = \{\top, \bot\} \cup \mathsf{N_C}(\mathcal{X}) \cup \{\{a\} : a \in \mathsf{N_I}(\mathcal{X})\}$ is the set of basic concepts occurring in $\mathcal{X}$ and $\mathsf{N_R^+}(\mathcal{X}) = \{r, r^- : r \in \mathsf{N_R}(\mathcal{X})\}$ is the set of roles occurring in $\mathcal{X}$ and their inverses.

3.1.1 Normal Form

It is often convenient to assume that DL knowledge bases are given in some normal form. This allows one to focus on knowledge bases with restricted syntax without losing generality, which greatly simplifies algorithms and proofs. The precise choice of the normal form depends on the application of interest. In what follows, we first introduce some general assumptions that we make about the knowledge bases and then describe the normalization procedure for $\mathcal{ALCHOIQ}$ knowledge bases with closed predicates that will be used in the remainder of this thesis. Finally, we point out how this normal form is adapted for its sublogic $\mathcal{ALCHOIF}$.

First off, from now on we *always* assume that any given ABox $\mathcal{A}$ only contains assertions that have one of the following forms:

- $A(c)$ or $\neg A(c)$, where A is a concept name in $\mathsf{N_C}$ and c is an individual in $\mathsf{N_I}$, or
- $r(c,d)$ or $\neg r(c,d)$, where r is a role name in $\mathsf{N_R}$ and c, d are individual names in $\mathsf{N_I}$.

Suppose $\mathcal{A}$ is a part of some knowledge base $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$. It is well-known (and also easy to see) that replacing any concept assertion $C(a) \in \mathcal{A}$ over a complex concept C by $A_C(a)$ and adding the CI $A_C \sqsubseteq C$ to the corresponding TBox $\mathcal{T}$, where $A_C \in \mathsf{N_C}$ is assumed to be a *fresh* concept name, i.e., A_C does not occur elsewhere in $\mathcal{T}$, results in a knowledge base that is equivalent to $\mathcal{K}$, up to the difference in the signature. Moreover, any role assertion $r^-(a,b) \in \mathcal{A}$ (resp. $\neg r^-(a,b)$), where r is a role name in $\mathsf{N_R}$, can simply be replaced by $r(b,a)$ (resp. $\neg r(b,a)$).

Another useful assumption that we often make is that TBoxes are closed under role inclusions.

Definition 3.1.4. *We say that a TBox $\mathcal{T}$ is* closed under role inclusions *if the following conditions hold:*

(i) $p \sqsubseteq p \in \mathcal{T}$, *for each role name* $p \in \mathsf{N_R}(\mathcal{T})$,

(ii) if, $r \sqsubseteq s \in \mathcal{T}$, *then* $r^- \sqsubseteq s^- \in \mathcal{T}$, *and*

(iii) if $r_1 \sqsubseteq r_2 \in \mathcal{T}$ *and* $r_2 \sqsubseteq r_3 \in \mathcal{T}$, *then also* $r_1 \sqsubseteq r_3 \in \mathcal{T}$.

Every TBox $\mathcal{T}$ can be transformed into an equivalent TBox $\mathcal{T}'$ that is closed under role inclusions in time polynomial in the size of $\mathcal{T}$. Indeed, this can be achieved in the following three steps:

1. For every role name $p \in \mathsf{N_R}(\mathcal{T})$, add $p \sqsubseteq p$ to $\mathcal{T}$. This step is linear in the number of roles that occur in $\mathcal{T}$.
2. For every axiom $r \sqsubseteq s \in \mathcal{T}$, add $r^- \sqsubseteq s^-$ to $\mathcal{T}$. This step is linear in the number of RIs in $\mathcal{T}$.

3. Compute the transitive closure over role inclusions. This can be done using any standard algorithm for computing the transitive closure over a binary relation and is known to be possible in cubic time in the number of RIs in $\mathcal{T}$ (e.g., by using the Floyd-Warshall algorithm [Flo62]).

Let $\mathcal{T}'$ denote the resulting TBox. As we only added RIs that are logically entailed by $\mathcal{T}$, it is immediate that $\mathcal{T}'$ is equivalent to $\mathcal{T}$.

Negation Normal Form. One particular normal form that is often used in the literature is *negation normal form* in which the negation is allowed only in front of atomic concepts (i.e., concept names and nominals).

Definition 3.1.5 (Negation Normal Form (NNF))**.** *A concept is in* negation normal form (NNF) *if negation only occurs in front of concept names or nominals. A TBox is in* negation normal form *if all its CIs are of the form* $\top \sqsubseteq C$*, where* C *is in NNF.*

Proposition 3.1.6. *Every concept* C *can be transformed in polynomial time into an equivalent concept that is in NNF, denoted by* $NNF(C)$*. Furthermore, every TBox* $\mathcal{T}$ *can be transformed in polynomial time into an equivalent TBox* $NNF(\mathcal{T})$ *that is in NNF.*

Proof. To transform C into $NNF(C)$, we simply push the negation inwards until it occurs only in front of nominals or concept names. To this end, we use the following equivalences:

$$\begin{aligned} \neg\neg C &\equiv C, \\ \neg(C \sqcap D) &\equiv \neg C \sqcup \neg D, & \neg(C \sqcup D) &\equiv \neg C \sqcap \neg D, \\ \neg\exists r.C &\equiv \forall r.\neg C, & \neg\forall r.C &\equiv \exists r.\neg C, \\ \neg(\leq nr.C) &\equiv \geq (n+1)r.C, & \neg(\geq nr.C) &\equiv \begin{cases} \bot, & \text{if } n = 0, \\ \leq (n-1)r.C, & \text{otw.} \end{cases} \end{aligned}$$

It is easy to see that obtaining $NNF(C)$ can be done in time polynomial in the size of C.

The TBox $NNF(\mathcal{T})$ is obtained from $\mathcal{T}$ by replacing every CI $C \sqsubseteq D$ in $\mathcal{T}$ by $\top \sqsubseteq NNF(\neg C \sqcup D)$. This is obviously a polynomial time procedure and $NNF(\mathcal{T})$ is equivalent to $\mathcal{T}$. □

It is important to note that normalization procedures must preserve relevant properties of the TBox that is being normalized. In the case of NNF, we obtain an equivalent TBox so all properties are automatically preserved, however may not always the case. For the purposes of this thesis, it is important that the TBox produced by some normalization procedure is equi-satisfiable and has the same atomic consequences as the original TBox. We thus introduce the notion of conservative extensions of TBoxes.

Definition 3.1.7. *Let* $\mathcal{T}$ *and* $\mathcal{T}'$ *be two TBoxes. We say that* $\mathcal{T}'$ *is a* conservative extension of $\mathcal{T}$ *if the following holds:*

- *sig*$(\mathcal{T}) \subseteq$ *sig*$(\mathcal{T}')$,
- *every model of* $\mathcal{T}'$ *is a model of* $\mathcal{T}$, *and*
- *every model* $\mathcal{I}$ *of* $\mathcal{T}$ *can be extended into a model* $\mathcal{I}'$ *of* $\mathcal{T}'$ *such that* $\mathcal{I}$ *and* $\mathcal{I}'$ *agree on the interpretation of the shared signature, i.e.,* $q^{\mathcal{I}'} = q^{\mathcal{I}}$, *for all* $q \in \mathsf{sig}(\mathcal{T})$.

It is well-known that conservative extensions preserve the satisfiability and atomic consequences of the original TBox.

$\mathcal{ALCHOIQ}$ Normal Form. We now formally define what it means for an $\mathcal{ALCHOIQ}$ TBox or a KB to be in normal form and show that we can transform arbitrary $\mathcal{ALCHOIQ}$ KBs into normalized ones in polynomial time.

Definition 3.1.8 ($\mathcal{ALCHOIQ}$ Normal Form)**.** *An* $\mathcal{ALCHOIQ}$ *TBox* $\mathcal{T}$ *is in* normal form *(or* normalized*), if every axiom in* $\mathcal{T}$ *has one of the following forms:*

$$\textbf{(N1)}\ B_1 \sqcap \cdots \sqcap B_{k-1} \sqsubseteq B_k \sqcup \cdots \sqcup B_m,$$
$$\textbf{(N2)}\ B_1 \sqsubseteq = n\, p.B_2 \qquad \textbf{(N3)}\ B_1 \sqsubseteq \forall p.B_2, \qquad \textbf{(N4)}\ r \sqsubseteq s,$$

where $\{B_1, \ldots, B_m\} \subseteq \mathsf{N}_C^+$, $n \geq 0$, $k > 1$, $m \geq k$, $p \in \mathsf{N}_R$, *and* $\{r, s\} \subseteq \mathsf{N}_R^+$, *and* $= n\, p.B$ *is an abbreviation for* $\geq n\, p.B \sqcap \leq n\, p.B$. *A KB* $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$ *is in* normal form *(or* normalized*), if* $\mathcal{T}$ *is in normal form.*

We refer to the axioms of type **(N2)** as *counting axioms* and the roles that occur in such axioms as *counting roles*.

To see that restricting our attention to normalized KBs is not a true restriction and, unless stated otherwise, our results apply to arbitrary KBs, we prove the following result.

Proposition 3.1.9. *Let* $\mathcal{T}$ *be an* $\mathcal{ALCHOIQ}$ *TBox. We can obtain an* $\mathcal{ALCHOIQ}$ *TBox* $\mathcal{T}'$ *from* $\mathcal{T}$ *in polynomial time such that* $\mathcal{T}'$ *is a conservative extension of* $\mathcal{T}$ *and* $\mathcal{T}'$ *in normal form.*

In view of Proposition 3.1.6, we can assume w.l.o.g that $\mathcal{T}$ is given in NNF. Furthermore, we assume that no concept in $\mathcal{T}$ is of the form $\exists r.C$, as we can replace such concepts with the equivalent concept $\geq 1 r.C$. We also assume that in any expression of the form $\leq n p.C$ or $\geq n p.C$, p is a role name. Indeed, expressions of the form $\leq n p^-.C$ (resp. $\geq n p^-.C$), where p is a role name, can be replaced by the CI $\leq n p'.C$ (resp. $\geq n p'.C$) and two RIs $p' \sqsubseteq p^-$ and $p^- \sqsubseteq p'$, where p' is a fresh role name.

The normalization procedure now involves the following two steps:

Step 1: We remove all nested concept occurrences by substituting fresh concept and role names for complex expressions in $\mathcal{T}$, similarly to the normalization procedure in [Sim13]. Namely, for each concept C occurring in $\mathcal{T}$, we assume two fresh concept names

A_C and $A_{\neg C}$ in $\mathsf{N_C}$. We then define the *structural transformation* of C, denoted as $st(C)$, as follows:

$$\begin{aligned} st(A) &:= A, & st(C \sqcap D) &:= A_C \sqcap A_D, \\ st(\top) &:= \top, & st(C \sqcup D) &:= A_C \sqcup A_D, \\ st(\bot) &:= \bot, & st(\forall r.C) &:= \forall r.A_C, \\ st(\{a\}) &:= \{a\}, & st(\leq nr.C) &:= \leq nr.A_C, \\ st(\neg C) &:= \neg A_C, & st(\geq nr.C) &:= \geq nr.A_C, \end{aligned}$$

where A is a concept name, a is an individual, C, D are concepts, r is a role, and n is a non-negative integer.

Let $\mathcal{T}'$ be the TBox consisting of all RIs in $\mathcal{T}$ as well as the following CIs:

- $st(C) \sqsubseteq A_C$ and $A_C \sqsubseteq st(C)$, for every concept C occurring in $\mathcal{T}$,
- $A_C \sqcap A_{\neg C} \sqsubseteq \bot$, and $\top \sqsubseteq A_C \sqcup A_{\neg C}$, for every concept C occurring in $\mathcal{T}$,
- $\top \sqsubseteq A_C$, for every $\top \sqsubseteq C$ in $\mathcal{T}$.

To show that $\mathcal{T}'$ is a conservative extension of $\mathcal{T}$, we first prove the following claim:

Lemma 3.1.10. *Let $\mathcal{I}$ be a model of $\mathcal{T}'$ and let C be an arbitrary concept occurring in $\mathcal{T}$. Then $\mathcal{I}$ has the following properties:*

- $(A_C)^{\mathcal{I}} = st(C)^{\mathcal{I}}$,
- $(\neg A_C)^{\mathcal{I}} = (A_{\neg C})^{\mathcal{I}}$,
- $C^{\mathcal{I}} = (A_C)^{\mathcal{I}}$, *and*
- $(\neg C)^{\mathcal{I}} = (A_{\neg C})^{\mathcal{I}}$,

Proof. As $\mathcal{T}'$ contains both $A_C \sqsubseteq st(C)$ and $st(C) \sqsubseteq A_C$ and $\mathcal{I}$ is a model of $\mathcal{T}'$, we have that $(A_C)^{\mathcal{I}} = st(C)^{\mathcal{I}}$. Furthermore, since $\mathcal{T}'$ contains both $A_C \sqcap A_{\neg C} \sqsubseteq \bot$, and $\top \sqsubseteq A_C \sqcup A_{\neg C}$, we have $(A_{\neg C})^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus (A_C)^{\mathcal{I}} = (\neg A_C)^{\mathcal{I}}$.

To prove that $C^{\mathcal{I}} = (A_C)^{\mathcal{I}}$, we proceed by structural induction on C:

Base case: Let C be a basic concept occurring in $\mathcal{T}$, i.e., C is of the form $A, \top, \bot$, or $\{a\}$, where $A \in \mathsf{N_C}$ and $a \in \mathsf{N_I}$. Observe that in this case $st(C) = C$, and so we immediately get that $(A_C)^{\mathcal{I}} = C^{\mathcal{I}}$.

Induction step: Let C be a concept occurring in $\mathcal{T}$ and assume the claim holds for all top-level subconcepts of C. We make the following case distinction:

$\neg D$ Due to our induction hypothesis, we have that $(A_D)^{\mathcal{I}} = D^{\mathcal{I}}$, and thus also $(\neg A_D)^{\mathcal{I}} = (\neg D)^{\mathcal{I}}$. As $(A_{\neg D})^{\mathcal{I}} = (st(\neg D))^{\mathcal{I}} = (\neg A_D)^{\mathcal{I}}$, we can conclude that $(\neg D)^{\mathcal{I}} = (A_{\neg D})^{\mathcal{I}}$. Due to our induction hypothesis, we have that $(A_{C_1})^{\mathcal{I}} = (C_1)^{\mathcal{I}}$ and $(A_{C_2})^{\mathcal{I}} = (C_2)^{\mathcal{I}}$. As $(A_{C_1 \sqcap C_2})^{\mathcal{I}} = (A_{C_1} \sqcap A_{C_2})^{\mathcal{I}} = (A_{C_1})^{\mathcal{I}} \cap (A_{C_2})^{\mathcal{I}} = (C_1)^{\mathcal{I}} \sqcap (C_2)^{\mathcal{I}} = (C_1 \sqcap C_2)^{\mathcal{I}}$, the claim follows.

$C_1 \sqcap C_2$ Due to our induction hypothesis, we have that $(A_{C_1})^{\mathcal{I}} = (C_1)^{\mathcal{I}}$ and $(A_{C_2})^{\mathcal{I}} = (C_2)^{\mathcal{I}}$. As $(A_{C_1 \sqcup C_2})^{\mathcal{I}} = (A_{C_1} \sqcup A_{C_2})^{\mathcal{I}} = (A_{C_1})^{\mathcal{I}} \cup (A_{C_2})^{\mathcal{I}} = (C_1)^{\mathcal{I}} \sqcup (C_2)^{\mathcal{I}} = (C_1 \sqcup C_2)^{\mathcal{I}}$, the claim follows.

$C_1 \sqcup C_2$ Due to our induction hypothesis, we have that $(A_{C_1})^{\mathcal{I}} = (C_1)^{\mathcal{I}}$ and $(A_{C_2})^{\mathcal{I}} = (C_2)^{\mathcal{I}}$. As $(A_{C_1 \sqcup C_2})^{\mathcal{I}} = (A_{C_1} \sqcup A_{C_2})^{\mathcal{I}} = (A_{C_1})^{\mathcal{I}} \cup (A_{C_2})^{\mathcal{I}} = (C_1)^{\mathcal{I}} \sqcup (C_2)^{\mathcal{I}} = (C_1 \sqcup C_2)^{\mathcal{I}}$, the claim follows.

$\forall p.D$ As $(A_{\forall p.D})^{\mathcal{I}} = (\forall p.A_D)^{\mathcal{I}}$ and, due to our induction hypothesis, $(A_D)^{\mathcal{I}} = D^{\mathcal{I}}$, then also $(A_{\forall p.D})^{\mathcal{I}} = (\forall p.D)^{\mathcal{I}}$.

$\leq np.D$ As $(A_{\leq np.D})^{\mathcal{I}} = (\leq np.A_D)^{\mathcal{I}}$ and, due to our induction hypothesis, we have that $(A_D)^{\mathcal{I}} = D^{\mathcal{I}}$, then also $(A_{\leq np.D})^{\mathcal{I}} = (\leq np.D)^{\mathcal{I}}$.

$\geq np.D$ As $(A_C)^{\mathcal{I}} = (\geq np.A_D)^{\mathcal{I}}$ and, due to our induction hypothesis, we have that $(A_D)^{\mathcal{I}} = D^{\mathcal{I}}$, then also $(A_{\geq np.D})^{\mathcal{I}} = (\geq np.D)^{\mathcal{I}}$.

Finally, due to $(A_C)^{\mathcal{I}} = C^{\mathcal{I}}$ and $(A_{\neg C})^{\mathcal{I}} = (\neg A_C)^{\mathcal{I}}$, we get $(A_{\neg C})^{\mathcal{I}} = (\neg C)^{\mathcal{I}}$. □

Lemma 3.1.11. *$\mathcal{T}'$ is a conservative extension of $\mathcal{T}$ and can be obtained in time polynomial in the size of $\mathcal{T}$.*

Proof. Let $\mathcal{I}$ be a model of the original TBox $\mathcal{T}$ and let $\mathcal{I}'$ be the interpretation that extends $\mathcal{I}$ by interpreting A_C and $A_{\neg C}$, for every concept C occurring in $\mathcal{T}$, as follows: $(A_C)^{\mathcal{I}'} = C^{\mathcal{I}}$ and $(A_{\neg C})^{\mathcal{I}'} = (\neg C)^{\mathcal{I}}$. Obviously, $\mathcal{I}'$ is a model of $\mathcal{T}'$.

Conversely, assume that $\mathcal{I}$ is a model of $\mathcal{T}'$. $\mathcal{I}$ satisfies all RIs in $\mathcal{T}$ since they also occur in $\mathcal{T}'$. In view of Lemma 3.1.10 we have that $(A_C)^{\mathcal{I}} = C^{\mathcal{I}}$. Moreover, for every CI $\top \sqsubseteq C \in \mathcal{T}$, we have the CI $\top \sqsubseteq A_C \in \mathcal{T}'$, and so $\top^{\mathcal{I}} \subseteq (A_C)^{\mathcal{I}}$. Putting the two together, this means that $\top^{\mathcal{I}} \subseteq C^{\mathcal{I}}$, and so $\mathcal{I}$ satisfies $\top \sqsubseteq C$.

We introduce two fresh concept names and four CI for every complex concept that occurs in $\mathcal{T}$, as well as one CI for every CI in $\mathcal{T}$. Thus, $\mathcal{T}'$ is linear in the size of $\mathcal{T}$. Moreover, it is easy to see that the normalization procedure described above runs in polynomial time. □

Step 2: Observe however that $\mathcal{T}'$ is not yet in normal form. Thus, we perform the transformations listed below to obtain a normalized TBox $\mathcal{T}''$ that is a conservative extension of $\mathcal{T}'$ and therefore also over $\mathcal{T}$. Let α be an axiom in $\mathcal{T}'$ that violates the normal form. We make the following case distinction depending on the form of α:

1. $\neg A_B \sqsubseteq A_{\neg B}$, for some basic concept B occurring in $\mathcal{T}$ (recall that we started from an TBox in NNF). We replace α by $\top \sqsubseteq A_B \sqcup A_{\neg B}$.

2. $A_{\neg B} \sqsubseteq \neg A_B$, for some basic concept B occurring in $\mathcal{T}$. We replace α by $A_{\neg B} \sqcap A_B \sqsubseteq \bot$.

3. $A_{C_1 \sqcap C_2} \sqsubseteq A_{C_1} \sqcap A_{C_2}$, for some $C_1 \sqcap C_2$ occurring in $\mathcal{T}$. In this case, we replace α by two CIs $A_{C_1 \sqcap C_2} \sqsubseteq A_{C_1}$ and $A_{C_1 \sqcap C_2} \sqsubseteq A_{C_2}$.

4. $A_{C_1} \sqcup A_{C_2} \sqsubseteq A_{C_1 \sqcup C_2}$, for some $C_1 \sqcup C_2$ occurring in $\mathcal{T}$. In this case, we replace α by two CIs $A_{C_1} \sqsubseteq A_{C_1 \sqcup C_2}$ and $A_{C_2} \sqsubseteq A_{C_1 \sqcup C_2}$.

5. $\forall r.A_C \sqsubseteq A_{\forall r.C}$, for some $\forall r.C$ occurring in $\mathcal{T}$. In this case, α is equivalent to $\leq 0 r.\neg A_C \sqsubseteq A_{\forall r.C}$. Due to Lemma 3.1.10, we can replace α by $\leq 0 r.A_{\neg C} \sqsubseteq A_{\forall r.C}$.

The transformations above make use of well-known equivalences and simple renaming. It is therefore easy to see that applying them yields a TBox that is a conservative extension of $\mathcal{T}'$ and polynomial in the size of $\mathcal{T}'$. The axioms that involve number restrictions require greater care and are treated as follows:

6. $A_{\leq np.C} \sqsubseteq \leq np.A_C$, for some $\leq np.C$ occurring in $\mathcal{T}$. For $0 \leq i \leq n$, we assume a fresh concept name $A_{=ip.C}$ and we add the following CI to $\mathcal{T}$: $A_{=ip.C} \sqsubseteq = ip.A_C$. We then replace α by $A_{\leq np.C} \sqsubseteq \bigsqcup_{0 \leq i \leq n} A_{=ip.C}$.

   To see that doing this yields a TBox that is a conservative extension of $\mathcal{T}'$, let $\mathcal{I}$ be an interpretation and assume $\mathcal{I} \vDash \mathcal{T}'$ and, thus, $\mathcal{I} \vDash A_{\leq np.C} \sqsubseteq \leq np.A_C$. We define the interpretation $\mathcal{I}'$ as the interpretation that extends $\mathcal{I}$ by interpreting the fresh concept names as follows: $(A_{=ip.C})^{\mathcal{I}'} = (= ip.A_C)^{\mathcal{I}}$, for all $0 \leq i \leq n$. Let $d \in (A_{\leq np.C})^{\mathcal{I}'}$. As $\mathcal{I}' \vDash A_{\leq np.C} \sqsubseteq \leq np.A_C$, this means that $d \in (\leq np.A_C)^{\mathcal{I}}$. By the semantics of $\leq np.A_C$, there exists k_d, $0 \leq k_d \leq n$, such that there are k_d distinct elements $e \in (A_C)^{\mathcal{I}}$ with $(d, e) \in p^{\mathcal{I}}$. This means that $d \in (= k_d p.A_C)^{\mathcal{I}}$, and thus, by definition of $\mathcal{I}'$, $d \in (A_{=k_d p.C})^{\mathcal{I}'}$. Hence, $d \in \sqcup_{0 \leq i \leq n} (A_{=ip.C})^{\mathcal{I}'} = (\bigsqcup_{0 \leq i \leq n} A_{=ip.C})^{\mathcal{I}'}$, which means that $\mathcal{I}' \vDash A_{\leq np.C} \sqsubseteq \bigsqcup_{0 \leq i \leq n} A_{=ip.C}$.

   Conversely, assume that $\mathcal{I} \vDash A_{\leq np.C} \sqsubseteq \bigsqcup_{0 \leq i \leq n} A_{=ip.C}$ and, for $0 \leq i \leq n$, $\mathcal{I} \vDash A_{=ip.C} \sqsubseteq = ip.A_C$. It is easy to see that $\mathcal{I} \vDash A_{\leq np.C} \sqsubseteq \bigsqcup_{0 \leq i \leq n} = ip.A_C$, and so $\mathcal{I} \vDash A_{\leq np.C} \sqsubseteq \leq np.A_C$.

7. $A_{\geq np.C} \sqsubseteq \geq np.A_C$, for some $\leq np.C$ occurring in $\mathcal{T}$.

   - $n = 0$. We replace the axiom by the equivalent $A_{\geq np.C} \sqsubseteq \top$.
   - $n \geq 1$. Assume a fresh role name p', and add $p' \sqsubseteq p$ to $\mathcal{T}$. We replace α by $A_{\geq np.C} \sqsubseteq = np'.A_C$

     Let $\mathcal{I}$ be an interpretation and assume that $\mathcal{I} \vDash A_{\geq np.C} \sqsubseteq \geq np.A_C$. For every element $e \in (\geq np.A_C)^{\mathcal{I}}$, there are at least n elements $e_1, \ldots, e_n$ in $A_C^{\mathcal{I}}$ such that $(e, e_i) \in p^{\mathcal{I}}$, for each $1 \leq i \leq n$. Let $\mathcal{I}'$ be an interpretation that extends $\mathcal{I}$ by interpreting p' as the set $\{(e, e_i) : e \in (\geq np.A_C)^{\mathcal{I}}, 1 \leq i \leq n\}$. Obviously, $\mathcal{I}' \vDash p' \sqsubseteq p$ and $(\geq np.A_C)^{\mathcal{I}'} = (= np'.A_C)^{\mathcal{I}'}$. Thus, $\mathcal{I}' \vDash A_{\geq np.C} \sqsubseteq = np'.A_C$. Conversely, assume that $\mathcal{I} \vDash A_{\geq np.C} \sqsubseteq = np'.A_C$ and $\mathcal{I} \vDash p' \sqsubseteq p$. Then also $\mathcal{I} \vDash A_{\geq np.C} \sqsubseteq = np.A_C$.

8. $\leq np.A_C \sqsubseteq A_{\leq np.C}$, for some $\leq np.C$ occurring in $\mathcal{T}$. This axiom is equivalent to $\neg A_{\leq np.C} \sqsubseteq \geq (n+1)p.A_C$. We then replace α by $A_{\neg \leq np.C} \sqsubseteq \geq (n+1)p.A_C$.

   Let $\mathcal{I}$ be a model of $\mathcal{T}'$ and assume that $\mathcal{I} \models \neg A_{\leq np.C} \sqsubseteq \geq (n+1)p.A_C$. Since $\mathcal{I} \models \mathcal{T}'$, we have $(A_{\neg \leq np.C})^{\mathcal{I}} = (\neg A_{\leq np.C})^{\mathcal{I}}$. Therefore, $\mathcal{I} \models A_{\neg \leq np.C} \sqsubseteq \geq (n+1)p.A_C$.

   Conversely, assume that $\mathcal{I}$ satisfies the TBox obtained from $\mathcal{T}'$ by replacing α by $A_{\neg \leq np.C} \sqsubseteq \geq (n+1)p.A_C$. In this case, we still have that $(A_{\neg \leq np.C})^{\mathcal{I}} = (\neg A_{\leq np.C})^{\mathcal{I}}$. Thus, $\mathcal{I} \models \neg A_{\leq np.C} \sqsubseteq \geq (n+1)p.A_C$, and so $\mathcal{I} \models \leq np.A_C \sqsubseteq A_{\leq np.C}$.

   To bring $A_{\neg \leq np.C} \sqsubseteq \geq (n+1)p.A_C$ into normal form, we proceed as explained in item 6.

9. $\geq np.A_C \sqsubseteq A_{\geq np.C}$, for some concept $\geq np.C$ occurring in $\mathcal{T}$.

   - $n = 0$. We replace the axiom by the equivalent $\top \sqsubseteq A_{\geq np.C}$.
   - $n \geq 1$. This axiom is equivalent to $\neg A_{\geq np.C} \sqsubseteq \leq (n-1)p.A_C$. We replace α by $A_{\neg \geq np.C} \sqsubseteq \leq (n-1)p.A_C$.
     The proof that this transformation yields a TBox that is a conservative extension of the original TBox $\mathcal{T}$ is the same as in the previous item.
     Finally, we treat $A_{\neg \geq np.C} \sqsubseteq \leq (n-1)p.A_C$ as explained in item 7.

We have shown along the way that performing the steps above yields a TBox $\mathcal{T}''$ that is a conservative extension of $\mathcal{T}'$ and is in the desired normal form. Moreover, $\mathcal{T}''$ is obtained from $\mathcal{T}'$ in polynomial time and it involves only linearly many new axioms, provided that integers are coded in unary. Together with Observation 3.1.11, this means that $\mathcal{T}''$ is a normalized TBox that is a conservative extension of and linear in the size of the original TBox $\mathcal{T}$ and can be obtained from $\mathcal{T}$ in polynomial time. Thus, Proposition 3.1.9 holds. As a corollary of Proposition 3.1.9, we have that every $\mathcal{ALCHOIQ}$ KB $\mathcal{K}$ with closed predicates can be normalized in polynomial time in a way that preserves the satisfiability and atomic consequences over the signature of $\mathcal{K}$:

Proposition 3.1.12. *Let $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$ be an $\mathcal{ALCHOIQ}$ KB with closed predicates. We can obtain a TBox $\mathcal{T}'$ from $\mathcal{T}$ in polynomial time such that $\mathcal{T}'$ is in normal form and for $\mathcal{K}' = (\mathcal{T}', \Sigma, \mathcal{A})$ and every interpretation $\mathcal{I}$ the following holds:*

1. *if $\mathcal{I} \models \mathcal{K}'$, then $\mathcal{I} \models \mathcal{K}$, and*
2. *if $\mathcal{I} \models \mathcal{K}$, then $\mathcal{I}$ can be extended into $\mathcal{I}'$ such that $\mathcal{I}' \models \mathcal{K}'$ and $q^{\mathcal{I}'} = q^{\mathcal{I}}$, for all $q \in \mathsf{sig}(\mathcal{K})$.*

We next fix some notation that remains the same for the remainder of this thesis, summarized in Table 3.1. For a given $\mathcal{ALCHOIQ}$ TBox $\mathcal{T}$, we let $n_{\mathcal{T}}$ be the number of concept names in $\mathsf{N_C}(\mathcal{T})$, $k_{\mathcal{T}}$ be the number of roles in $\mathsf{N_R^+}(\mathcal{T})$, $m_{\mathcal{T}}$ be the number of counting axioms occurring in $\mathcal{T}$, and we let $c_{\mathcal{T}}$ be the maximum integer occurring in $\mathcal{T}$.

$n_{\mathcal{T}}$	number of different concept names in $\mathsf{N_C}(\mathcal{T})$
$k_{\mathcal{T}}$	number of different roles in $\mathsf{N_R^+}(\mathcal{T})$
$m_{\mathcal{T}}$	number of counting axioms occurring in $\mathcal{T}$
$c_{\mathcal{T}}$	maximum integer occurring in $\mathcal{T}$

Table 3.1: Summary of symbols with fixed meaning, for a TBox $\mathcal{T}$

$\mathcal{ALCHOIF}$ Normal Form. We can easily adapt the $\mathcal{ALCHOIQ}$ normalization procedure to also show that every $\mathcal{ALCHOIF}$ TBox $\mathcal{T}$ can be transformed in polynomial time into a TBox $\mathcal{T}'$ that is a conservative extension of $\mathcal{T}$ and in $\mathcal{ALCHOIF}$ *normal form*, described below.

Definition 3.1.13. *[$\mathcal{ALCHOIF}$ Normal Form] An $\mathcal{ALCHOIF}$ TBox $\mathcal{T}$ is in* normal form *(or* normalized*), if every axiom in $\mathcal{T}$ has one of the following forms:*

$$\textbf{(N1)}\ B_1 \sqcap \cdots \sqcap B_{k-1} \sqsubseteq B_k \sqcup \cdots \sqcup B_m,$$

$$\textbf{(N2')}\ B_1 \sqsubseteq \exists p.B_2 \qquad \textbf{(N3)}\ B_1 \sqsubseteq \forall p.B_2, \qquad \textbf{(N4)}\ r \sqsubseteq s, \qquad \textbf{(N5)}\ \mathsf{func}(r),$$

where $\{B_1, \ldots, B_m\} \subseteq \mathsf{N_C^+}$, $p \in \mathsf{N_R}$, and $r \in \mathsf{N_R^+}$.

A KB $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$ is in normal form *(or* normalized*), if $\mathcal{T}$ is in normal form.*

Indeed, we remove nested concept occurrences exactly as before. To further normalize the axioms that violate the normal form, we use the transformations listed in items 1-4 from Step 2 of the normalization procedure together with the following transformations:

- $\forall p.A_C \sqsubseteq A_C$, for some $\forall p.C$ occurring in $\mathcal{T}$. We replace this axiom by $A_{\neg\forall p.C} \sqsubseteq \exists p A_{\neg C}$.
- $\exists p.A_C \sqsubseteq A_{\exists p.C}$, for some $\exists p.C$ occurring in $\mathcal{T}$. We replace this axiom by $A_C \sqsubseteq \forall p'.A_{\exists p.C}$ and add two RIs $p' \sqsubseteq p^-$ and $p^- \sqsubseteq p'$.

Let $\mathcal{T}'$ be the resulting TBox together with all RIs and functionality axioms from $\mathcal{T}$. Then $\mathcal{T}'$ is a conservative extension of $\mathcal{T}$ that is in $\mathcal{ALCHOIF}$ normal form. Moreover, $\mathcal{T}'$ was obtained in polynomial time. Thus, we have the following result.

Proposition 3.1.14. *Let $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$ be an $\mathcal{ALCHOIF}$ KB with closed predicates. We can obtain a TBox $\mathcal{T}'$ from $\mathcal{T}$ in polynomial time such that $\mathcal{T}'$ is in normal form and for $\mathcal{K}' = (\mathcal{T}', \Sigma, \mathcal{A})$ and every interpretation $\mathcal{I}$ the following holds:*

1. *if $\mathcal{I} \models \mathcal{K}'$, then $\mathcal{I} \models \mathcal{K}$, and*
2. *if $\mathcal{I} \models \mathcal{K}$, then $\mathcal{I}$ can be extended into $\mathcal{I}'$ such that $\mathcal{I}' \models \mathcal{K}'$ and $q^{\mathcal{I}'} = q^{\mathcal{I}}$, for all $q \in \mathsf{sig}(\mathcal{K})$.*

3.2 Characterizing KB Satisfiability via Integer Programming

We next devise a procedure that decides the satisfiability of $\mathcal{ALCHOIQ}$ KBs with closed predicates using integer programming techniques. Our approach is closely related to techniques in [Cal96, LST05, PH05] that reduce the *finite satisfiability problem* to finding integer solutions to a system of linear inequalities. We use parts of the inequality systems in [GGI+20], which (among other results) shows that deciding whether a given $\mathcal{ALCHOIF}$ TBox has a model in which *some* input predicates have finite extensions can be characterized via integer programming. For our purposes, we modify the procedure to support ABoxes, closed predicates, and generalized counting axioms instead of functionality. In the remainder of this chapter, we show that we can correctly characterize the problem of satisfiability of $\mathcal{ALCHOIQ}$ KBs with closed predicates as a system of linear inequalities together with certain side conditions. In view of the results from the previous section, we restrict our attention to KBs $K = (\mathcal{T}, \Sigma, \mathcal{A})$ where $\mathcal{T}$ is in $\mathcal{ALCHOIQ}$ normal form and closed under role inclusions and $\mathcal{A}$ contains only assertions and negated assertions over concept and role names.

3.2.1 Chromatic models

Inspired by a technique used in [PH05], we begin by introducing the notion of *chromatic models* of knowledge bases. This notion will aid us in the correct characterization of counting axioms and its importance will become obvious in the remainder of this chapter.

We first recall a standard notion of a *type* and a *role type* in description logics.

Definition 3.2.1. *Given a knowledge base $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$, a* role type *for $\mathcal{K}$ is any set $R \subseteq \mathsf{N}_R^+(\mathcal{K})$ and a* type *for $\mathcal{K}$ is any set $T \subseteq \mathsf{N}_C^+(\mathcal{K})$ such that*

1. *$\top \in T$ and $\bot \notin T$;*
2. *for all $a, b \in \mathsf{N}_I(\mathcal{K})$, if $\{a\}, \{b\} \in T$, then $a = b$.*

We denote the set of all types for $\mathcal{K}$ by $\mathrm{Types}(\mathcal{K})$. A type lists the basic concepts that some domain element participates in and a role type lists the roles that a pair of domain elements participates in. The intuition behind the first condition in Definition 3.2.1 is obvious and the second condition arises due to the SNA.

An important auxiliary notion is the notion of *invertibility*, formally defined as follows.

Definition 3.2.2. *Let $\mathcal{T}$ be an $\mathcal{ALCHOIQ}$ TBox, T, T' be types for $\mathcal{T}$, $R \subseteq \mathsf{N}_R^+(\mathcal{T})$. We say that (T, R, T') is* invertible *if the following holds:*

1. *there exists an axiom $A \sqsubseteq = n\, s.B \in \mathcal{T}$ s.t. $A \in T$, $B \in T'$, and $s \in R$, and*

2. *there exists an axiom* $A' \sqsubseteq = m\, p.B' \in \mathcal{T}$ *s.t.* $A' \in T'$, $B' \in T$, *and* $p^- \in R$.

Consider now an interpretation $\mathcal{I}$. For a domain element $d \in \Delta^{\mathcal{I}}$, we let $\mathsf{t}(d) = \{B \in \mathsf{N}_{\mathsf{C}}^{+}(\mathcal{T}) : d \in B^{\mathcal{I}}\}$ denote the type of d in $\mathcal{I}$, and, for a pair of elements $d, d' \in \Delta^{\mathcal{I}}$, we let $\mathsf{rt}(d, d') = \{r \in \mathsf{N}_{\mathsf{R}}^{+}(\mathcal{T}) : (d, d') \in r^{\mathcal{I}}\}$ denote the role type of (d, d') in $\mathcal{I}$.

Definition 3.2.3. *Let $\mathcal{K}$ be a knowledge base and $\mathcal{I}$ be a model of $\mathcal{K}$. We say that $\mathcal{I}$ is* chromatic *if for distinct elements $d, d', d'' \in \Delta^{\mathcal{I}}$ the following holds:*

1. *If $(\mathsf{t}(d), \mathsf{rt}(d, d'), \mathsf{t}(d'))$ is invertible, then $\mathsf{t}(d) \neq \mathsf{t}(d')$, and*
2. *If both $(\mathsf{t}(d), \mathsf{rt}(d, d'), \mathsf{t}(d'))$ and $(\mathsf{t}(d), \mathsf{rt}(d, d''), \mathsf{t}(d''))$ are invertible, then $\mathsf{t}(d') \neq \mathsf{t}(d'')$.*

Consider a knowledge base $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$. We next show that every model of $\mathcal{K}$ can be converted into a chromatic model of $\mathcal{K}$. To this end, we assume that there are $n = \lceil log(m_{\mathcal{T}} \cdot c_{\mathcal{T}}^2 + 1) \rceil$ special concept names $A_1^{\tau}, \ldots, A_n^{\tau} \in \mathsf{N}_{\mathsf{C}}(\mathcal{K})$ that occur only in $\mathcal{T}$ and only in the axioms of type $A_i^{\tau} \sqsubseteq A_i^{\tau}$, $1 \leq i \leq n$. Note that, since $A_1^{\tau}, \ldots, A_n^{\tau}$ do not occur in $\mathcal{K}$ other than in the trivial axioms of type $A_i^{\tau} \sqsubseteq A_i^{\tau}$, it is easy to see that these concept names have no effect on the satisfiability of $\mathcal{K}$. In particular, if $\mathcal{I} \models \mathcal{K}$ and $\mathcal{I}'$ is an arbitrary interpretation such that the restrictions of $\mathcal{I}'$ and $\mathcal{I}$ to the symbols in $\mathcal{K}$ other than $A_1^{\tau}, \ldots, A_n^{\tau}$ coincide, then also $\mathcal{I}' \models \mathcal{K}$.

Proposition 3.2.4. *Let $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$ be a knowledge base, and let $\mathcal{I}$ be an interpretation. If $\mathcal{I} \models \mathcal{K}$, then $\mathcal{I}$ can be modified into a chromatic model of $\mathcal{K}$ by suitably interpreting the concept names $A_1^{\mathcal{T}}, \ldots, A_n^{\mathcal{T}}$, where $n = \lceil log(m_{\mathcal{T}} \cdot c_{\mathcal{T}}^2 + 1) \rceil$.*

Proof. (Proof adapted from [PH05]) Let $G = (V, E)$ be an undirected graph, where $V = \Delta^{\mathcal{I}}$ and $E = E_1 \cup E_2$, where

$$E_1 = \{(d, d') : d \neq d' \text{ and } (\mathsf{t}(d), \mathsf{rt}(d, d'), \mathsf{t}(d')) \text{ is invertible}\}$$

$$E_2 = \{(d', d'') : d' \neq d'' \text{ and for some } d \in V, (d, d') \in E_1 \text{ and } (d, d'') \in E_1\}.$$

From Definition 3.2.2 it follows that, for all distinct $d, d' \in \Delta^{\mathcal{I}}$, if $(\mathsf{t}(d), \mathsf{rt}(d, d'), \mathsf{t}(d'))$ is invertible, then $(\mathsf{t}(d'), \mathsf{rt}(d', d), \mathsf{t}(d))$ is also invertible. This means that E_2 can equivalently be defined as:

$$E_2 = \{(d, d'') : d \neq d'' \text{ and for some } d' \in V, (d, d') \in E_1 \text{ and } (d', d'') \in E_1\}.$$

As explained above, since $A_1^{\mathcal{T}}, \ldots, A_n^{\mathcal{T}}$ occur in $\mathcal{K}$ only in the trivial axioms $A_i^{\tau} \sqsubseteq A_i^{\tau}$, $\mathcal{I}$ remains a model of $\mathcal{K}$ even if the extensions of $A_1^{\mathcal{T}}, \ldots, A_n^{\mathcal{T}}$ are modified in an arbitrary way. As $\mathcal{I}$ is a model of $\mathcal{K}$, that means that for each $d \in \Delta^{\mathcal{I}}$ there are at most $m_{\mathcal{T}} \cdot c_{\mathcal{T}}$

different $d' \in \Delta^{\mathcal{I}}$ s.t. $(d, d') \in r^{\mathcal{I}}$, for some counting role r, and so there are at most $m_{\mathcal{T}} \cdot c_{\mathcal{T}}$ distinct $d' \in \Delta^{\mathcal{I}}$ s.t. $(\mathsf{t}(d), \mathsf{rt}(d, d'), \mathsf{t}(d'))$ is invertible. Thus, the maximum degree of any node in G is $(m_{\mathcal{T}} \cdot c_{\mathcal{T}})^2$. It is well known that for an undirected graph with maximum degree k, we can color the vertices of the graph with $(k+1)$ different colors such that no edge connects two vertices with the same color. Thus, by using $A_1^{\mathcal{T}}, \ldots, A_n^{\mathcal{T}}$ to encode these colors, we can transform $\mathcal{I}$ into a chromatic model of $\mathcal{T}$. □

3.2.2 Satisfiability via Tiles & Mosaics

We next introduce the most important notions that are needed for characterizing the satisfiability of $\mathcal{ALCHOIQ}$ KBs with closed predicates via systems of linear inequalities.

Consider a knowledge base $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$ with closed predicates and recall that we can view models of $\mathcal{K}$ as labeled directed graphs. A *tile* for $\mathcal{K}$ describes a kind of domain element that can occur in a model of $\mathcal{K}$ together with the relevant part of its neighborhood in the graph-theoretical sense. Intuitively, a tile τ carries the following information: (i) which basic concepts a domain element of the kind τ participates in, as well as (ii) the kind of neighbors such an element has in a model of $\mathcal{K}$. A *mosaic* for $\mathcal{K}$ is a function N that assigns to each tile a multiplicity. Mosaics satisfy certain conditions that ensure it is possible to build a model of $\mathcal{K}$ by taking, for each tile τ, $N(\tau)$ domain elements that fit the description given by τ. Deciding the satisfiability of $\mathcal{K}$ thus amounts to deciding the existence of a mosaic for $\mathcal{K}$. We next formally define these notions.

Definition 3.2.5. *Given an $\mathcal{ALCHOIQ}$ KB $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$, a* tile *for $\mathcal{K}$ is a tuple (T, ρ), where T is a type for $\mathcal{K}$ and ρ is a set of triples (R, T', k) s.t. $R \subseteq \mathsf{N}_R^{\pm}(\mathcal{K})$, T' is a type for $\mathcal{K}$, $k > 0$, and the following conditions hold:*

- *T1. If $(R, T', k) \in \rho$ then $(R, T', k') \in \rho$, for all $0 < k' < k$*
- *T2. For every $(R, T', k) \in \rho$, there exists some $A \sqsubseteq\!= nr.B \in \mathcal{T}$ such that $A \in T$, $B \in T'$ and $r \in R$*
- *T3. If $B_1 \sqcap \cdots \sqcap B_{k-1} \sqsubseteq B_k \sqcup \cdots \sqcup B_m \in \mathcal{T}$ and $\{B_1, \ldots, B_{k-1}\} \subseteq T$, then $\{B_k, \ldots, B_m\} \cap T \neq \emptyset$*
- *T4. If $A \sqsubseteq\!= n\, r.B \in \mathcal{T}$ and $A \in T$, then $|\{(R, T', k) \in \rho : r \in R \text{ and } B \in T'\}| = n$.*
- *T5. For all $(R, T', k) \in \rho$, the following hold:*
  - *(a) If $A \sqsubseteq \forall r.B \in \mathcal{T}$, $A \in T$ and $r \in R$, then $B \in T'$*
  - *(b) If $A \sqsubseteq \forall r.B \in \mathcal{T}$, $A \in T'$ and $r^- \in R$, then $B \in T$*
  - *(c) If $r \sqsubseteq s \in \mathcal{T}$ and $r \in R$, then $s \in R$*
  - *(d) If (T, R, T') is invertible, then $T \neq T'$, and there is no other $(R', T', k') \in \rho$ such that (T, R', T') is invertible,*
  - *(e) If $\neg p(a, b) \in \mathcal{A}$, $\{a\} \in T$, and $p \in R$, then $\{b\} \notin T'$*

(f) If $\neg p(a,b) \in \mathcal{A}$, $\{b\} \in T$, *and* $p^- \in R$, *then* $\{a\} \notin T'$

T6. If $A(c) \in \mathcal{A}$ *and* $\{c\} \in T$ *then* $A \in T$

T7. If $\neg A(c) \in \mathcal{A}$ *and* $\{c\} \in T$, *then* $A \notin T$

T8. If $A \in \Sigma \cap \mathsf{N_C}$ *and* $A \in T$ *then there exists* $c \in \mathsf{N_I}$ *s.t.* $\{c\} \in T$ *and* $A(c) \in \mathcal{A}$

T9. If $r \in \Sigma \cap \mathsf{N_R}$, *then for all* $(R, T', k) \in \rho$ *with* $r \in R$, *there exist* $c, d \in \mathsf{N_I}$ *s.t.* $\{c\} \in T$, $\{d\} \in T'$ *and* $r(c,d) \in \mathcal{A}$.

We denote by Tiles($\mathcal{K}$) the set of all tiles for $\mathcal{K}$.

Consider a tile $\tau = (T, \rho)$ for $\mathcal{K}$, a model $\mathcal{I}$ of $\mathcal{K}$, and a domain element d in $\Delta^{\mathcal{I}}$. If d fits the description provided by τ, we call d an *instance of* τ, which means that d participates in exactly those basic concepts that are listed in T. Moreover, every $(R, T', k) \in \rho$ represents a distinct arc in a graphical representation of $\mathcal{I}$ that has d as its start node, whose label contains all roles in R and whose end node is an element in $\Delta^{\mathcal{I}}$ that participates exactly in the basic concepts given by T'. In other words, every $(R, T', k) \in \rho$ represents a distinct R-neighbor of d of type T'. As d might have multiple neighbors of type T' that are connected to d using the same set of roles R, we use the integer k to denote the k-th neighbor of d of type T' connected to d via the roles from R. Formally, we have the following definition.

Definition 3.2.6. *Given an interpretation* $\mathcal{I}$ *and a tile* $\tau = (T, \rho)$, *we say that* d *is an* instance *of* τ *if:*

- $\mathsf{t}(d) = T$,
- *for all* $(R, T', k) \in \rho$, *there exists* $e \in \Delta^{\mathcal{I}}$, *referred to as a witness of* (R, T', k), *such that* $\mathsf{t}(e) = T'$ *and* $R \subseteq \mathsf{rt}(t, t')$, *and*
- *no two distinct triples in* ρ *have the same witness.*

It is crucial to note that, in general, ρ does not describe all the neighbors that d may have in $\mathcal{I}$, but only those that are necessary to satisfy the counting axioms in $\mathcal{T}$. This kind of encoding allows us to keep the size of ρ independent of the ABox, which plays an important role in defining a polynomial and data-independent Datalog translation and obtaining the desired complexity bounds.

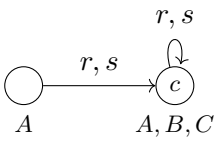
Example 3.2.7. *Consider the following KB* $\mathcal{K} = (\mathcal{T}, \{B\}, \{B(c)\})$, *where* $\mathcal{T} = \{B \sqcap C \sqsubseteq A, A \sqsubseteq {=}\, 1r.B, A \sqsubseteq \forall r.C, s \sqsubseteq r\}$. *Let* τ *be the following tile for* $\mathcal{K}$

$$\tau = (\{\top, A\}, \{(\{r, s\}, \{\top, A, B, C, \{c\}\}, 1)\}).$$

In a model $\mathcal{I}$ *of* $\mathcal{K}$, *a domain element* $d \in \Delta^{\mathcal{I}}$ *that is an instance of* τ *has the following properties:*

- *d has the (unary) type $\{\top, A\}$, i.e., $d \in A^{\mathcal{I}}$ and $d \notin D^{\mathcal{I}}$, for all $D \in \mathsf{N}_C^+(\mathcal{K}) \setminus \{\top, A\}$,*
- *the constant c is an r-successor and an s-successor of d, i.e, $(d, c) \in r^{\mathcal{I}} \cap s^{\mathcal{I}}$, and*
- *c has the (unary) type $\{\top, A, B, C, \{c\}\}$.*

For example, in the model of $\mathcal{K}$ given below, we can say that the element on the left is an instance of τ.



We next briefly explain the intuitions behind the conditions in Definition 3.2.5. Recall that each triple $(R, T', k) \in \rho$ represents a distinct neighbor of the domain element d that is an instance of τ. Notice that, by definition, ρ is a set, which means that it does not contain duplicates. However, it could be the case that we need to encode two distinct neighbors that happen to have the same type and are connected to the d via the same roles. To overcome this issue, we consider triples (R, T', k), where k is an integer that tells us that (R, T', k) encodes the k-th R-neighbor of d of type T'. This is reflected in the first condition. We also already mentioned that the tiles only encode the relevant part of the neighborhood of a domain element, i.e., those neighbors that serve as witnesses for counting axioms in TBox $\mathcal{T}$ of the knowledge base. This is reflected in the condition T2. Note that this condition also places an upper bound on the number of neighbors that we encode in ρ. The intuition behind conditions T3, T4, and T5 (a)-(c) is rather simple – they all relate to the satisfaction of TBox axioms. In particular, the third condition ensures the satisfaction of axioms of the type $B_1 \sqcap \cdots \sqcap B_{k-1} \sqsubseteq B_k \sqcup \cdots \sqcup B_m \in \mathcal{T}$, the condition T4 guarantees that d has n witnesses for every axiom $= nr.B \in \mathcal{T}$ and conditions T5 (a)-(c) ensure that d and its neighbors respect axioms of the type $A \sqsubseteq \forall r.B$ and $r \sqsubseteq s$. Condition T5 (d) makes use of Proposition 3.2.4 which essentially says it is enough to only focus on chromatic models. We will explain why this is needed a little later. We next have the conditions that ensure that the description of d is compatible with the assertions in $\mathcal{A}$ (conditions T5 (e), T5 (f), T6, and T7), and those that ensure that the closed predicates are respected (conditions T8 and T9).

We now move on to defining mosaics for a given knowledge base $\mathcal{K}$, which are functions that tell us, for each tile τ, how many instances of τ we need to build a model of $\mathcal{K}$. Recall that it is a well-known fact that $\mathcal{ALCHOIQ}$ is capable of enforcing infinite models, as shown in the following example.

Example 3.2.8. *Consider the knowledge base $\mathcal{K} = (\{\{a\} \sqsubseteq \neg A, \top \sqsubseteq= 1r.A, \top \sqsubseteq= 1r^-.A\}, \emptyset, \emptyset)$. Every model of $\mathcal{K}$ must contain an infinite r-chain starting at a, and so $\mathcal{K}$ has only infinite models.*

This means that, in order to build a model of $\mathcal{K}$, we might need to instantiate some tiles infinitely many times. Therefore, we consider the set $\mathbb{N}^*$ that extends the set of natural numbers $\mathbb{N}$ with a new value $\aleph_0$, representing infinity. We also extend the usual relation $<$ and operations $\cdot$ and $+$ as follows:

- $n < \aleph_0$, for all $n \in \mathbb{N}$,
- $\aleph_0 \cdot \aleph_0 = \aleph_0 + \aleph_0 = \aleph_0 + 0 = 0 + \aleph_0 = \aleph_0 + n = n + \aleph_0 = \aleph_0 \cdot n = n \cdot \aleph_0 = \aleph_0$, for all $n \in \mathbb{N} \setminus \{0\}$, and
- $0 \cdot \aleph_0 = \aleph_0 \cdot 0 = 0$.

We let $\mathbb{N}^* = \mathbb{N} \cup \{\aleph_0\}$. Furthermore, we say that infinite sets have cardinality $\aleph_0$.

Definition 3.2.9. *Let $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$ be a knowledge base. A* mosaic *for $\mathcal{K}$ is a function $N : Tiles(\mathcal{K}) \to \mathbb{N}^*$ such that:*

M1. For every $\{c\} \in \mathsf{N}_C^+(\mathcal{K})$: $\displaystyle\sum_{\substack{(T,\rho)\in Tiles(\mathcal{K}),\\ \{c\}\in T}} N((T,\rho)) = 1.$

M2. The following inequality is satisfied: $\displaystyle\sum_{\tau\in Tiles(\mathcal{K})} N(\tau) \geq 1.$

M3. For every pair $T, T' \in Types(\mathcal{K})$ and every $R \subseteq \mathsf{N}_R^+(\mathcal{K})$ s.t. (T, R, T') is invertible, the following holds:

$$\sum_{\substack{(T,\rho)\in Tiles(\mathcal{K}),\\ (R,T',k)\in\rho}} N((T,\rho)) = \sum_{\substack{(T',\rho')\in Tiles(\mathcal{K}),\\ (R^-,T,l)\in\rho'}} N((T',\rho')).$$

M4. For every every tile $\tau = (T, \rho)$ and every type T':

$$N(\tau) > 0 \implies \sum_{(T',\rho')\in Tiles(\mathcal{K})} N((T',\rho')) \geq |\{(R,T',k) : (R,T',k) \in \rho\}|$$

M5. For all $\{a\}, \{b\} \in \mathsf{N}_C^+(\mathcal{K})$ and all $A, B \in \mathsf{N}_C(\mathcal{K})$, if for some $p, r \in \mathsf{N}_R^+(\mathcal{K})$ at least one of the following condition holds

(a) $p(a,b) \in \mathcal{A}$, $p \sqsubseteq r \in \mathcal{T}$ and $A \sqsubseteq \forall r.B \in \mathcal{T}$, or

(b) $p(b,a) \in \mathcal{A}$, $p^- \sqsubseteq r \in \mathcal{T}$ and $A \sqsubseteq \forall r.B \in \mathcal{T}$,

then the following implication must hold:

$$\sum_{\substack{(T,\rho)\in Tiles(\mathcal{K}),\\ \{a\},A\in T}} N((T,\rho)) > 0 \text{ implies } \sum_{\substack{(T',\rho')\in Tiles(\mathcal{K}),\\ \{b\},B\in T'}} N((T',\rho')) > 0.$$

M6. For all $\{a\}, \{b\} \in \mathsf{N}_C^+(\mathcal{K})$*, all* $A, B \in \mathsf{N}_C(\mathcal{K})$*, all tiles* $\tau' = (T', \rho')$ *with* $\{\{b\}, B\} \subseteq T'$*, and all roles* $r \in \mathsf{N}_R^+(\mathcal{K})$*, if*

(a) $A \sqsubseteq= nr.B \in \mathcal{T}$*, and* $p(a,b) \in \mathcal{A}$*,* $p \sqsubseteq r \in \mathcal{T}$*, for some role name* p*, or*

(b) $A \sqsubseteq= nr.B \in \mathcal{T}$*, and* $p(b,a) \in \mathcal{A}$*,* $p^- \sqsubseteq r \in \mathcal{T}$*, for some role name* p*,*

then the following implication must hold:

$$N((T', \rho')) > 0 \text{ implies } \sum_{\substack{(T,\rho) \in Tiles(\mathcal{K}),\\ \{\{a\},A\} \subseteq T,\\ |\{(R,T',k) \in \rho : r \in R\}| = 0}} N((T, \rho)) \leq 0.$$

Consider a mosaic N for a knowledge base $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$. Conditions M1-M5 ensure that we can build a model $\mathcal{I}$ of $\mathcal{K}$ such that $\Delta^{\mathcal{I}} = \{d_1^\tau, \ldots, d_{N(\tau)}^\tau : \tau \in \text{Tiles}(\mathcal{K})\}$, where d_i^τ, for $1 \leq i \leq N(\tau)$, is an instance of the tile τ. The condition M1 ensures that, for every constant c occurring in $\mathcal{K}$, there is exactly one element in $\Delta^{\mathcal{I}}$ that participates in the nominal $\{c\}$ – namely the constant c itself. As description logics do not allow interpretations with empty domains, the condition M2 ensures that at least one tile is instantiated, i.e., $\Delta^{\mathcal{I}} \neq \emptyset$. Conditions M3 and M4 together ensure that it is possible to construct the interpretation function $\cdot^{\mathcal{I}}$ such that each domain element $d_i^\tau \in \Delta^{\mathcal{I}}$ has the neighbors prescribed by the tile τ. Recall that if we have a tile $\tau = (T, \rho)$ for $\mathcal{K}$ with $N(\tau) > 0$, this means there is a domain element $d_i^\tau \in \Delta^{\mathcal{I}}$ that is an instance of τ and thus we have to find suitable witnesses for every triple in ρ. The condition M4 makes sure that, for each type T', the total number of elements in $\Delta^{\mathcal{I}}$ that (are instances of tiles that) have T' as the unary type is greater than or equal to the number of triples in ρ that require a witness of type T'. Thus, we can simply use a different domain element of type T' as a witness to each such triple.

Unfortunately, M4 alone is not enough to ensure that a mosaic actually encodes a model, as illustrated by the following example.

Example 3.2.10. *Consider the following knowledge base* $\mathcal{K} = (\mathcal{T}, \emptyset, \emptyset)$*, where* $\mathcal{T} = \{A \sqsubseteq= 2r.B, C \sqsubseteq= 1r^-.A\}$ *and let*

$$\tau_1 = (\{\top, A\}, \{(\{r\}, \{\top, B\}, 1), (\{r\}, \{\top, B, C\}, 1)\})$$
$$\tau_2 = (\{\top, B\}, \{\})$$
$$\tau_3 = (\{\top, B, C\}, \{(\{r^-\}, \{\top, A\}, 1)\}).$$

Let $N : Tiles(\mathcal{K}) \to \mathbb{N}^*$ *such that*

$$N(\tau) = \begin{cases} 2, & \text{if } \tau = \tau_1 \\ 1, & \text{if } \tau = \tau_2 \\ 1, & \text{if } \tau = \tau_3 \\ 0, & \text{otherwise} \end{cases}$$

Observe that N satisfies every condition in Definition 3.2.9 except M3 and let us try to build a model $\mathcal{I}$ of $\mathcal{K}$ according to N. We let $\Delta^{\mathcal{I}} = \{d_1^{\tau_1}, d_2^{\tau_1}, d_1^{\tau_2}, d_1^{\tau_3}\}$. According to the information provided by the tiles, elements $d_1^{\tau_1}$ and $d_2^{\tau_1}$ each need two r-successors participating in the concept name B. As N satisfies M4, it should be possible to follow the approach described above to find r-successors of $d_1^{\tau_1}$ and $d_2^{\tau_1}$. Consider first the element $d_1^{\tau_1}$. Since there are no other options, we use $d_1^{\tau_2}$ as its first r-successor and $d_1^{\tau_3}$ as its second r-successor. We then consider $d_2^{\tau_1}$. The element $d_1^{\tau_2}$ can still serve as the first r-successor to $d_2^{\tau_1}$. However, if we now use $d_1^{\tau_3}$ as the second r-successor of $d_2^{\tau_1}$ we run into a problem. Namely, due to the axiom $C \sqsubseteq= 1r^-.A \in \mathcal{T}$, $d_1^{\tau_3}$ cannot serve as an r-successor to more than one domain element participating in A. Hence, N does not encode a model.

To avoid the issue in the previous example, we must first identify the problematic situations that can occur. This is where the notion of invertibility, as defined in Section 3.2.1, comes into play. Consider a tile $\tau = (T, \rho)$ and assume $(R, T', k) \in \rho$. Furthermore, assume that (T, R, T') is invertible. Obviously, a domain element that is an instance of τ will require a R-neighbor of type T'. However, from the definition of invertibility, it follows that a domain element of type T' can be an R-successor only to a limited number of elements of type T. Note that this is exactly the problematic situation we had in the previous example. One way to deal with such situations is to restrict our attention solely to chromatic models, which we know is possible due to Proposition 3.2.4. The condition T5 (d) in Definition 3.2.5 ensures that the specification of necessary neighbors in tiles respects chromaticity. Focusing only on chromatic models allows us to exploit a very convenient property that they possess and that we describe next. Let T and T' be two types for $\mathcal{K}$, and $R \subseteq \mathsf{N}_{\mathsf{R}}^{+}(\mathcal{K})$ such that (T, R, T') is invertible. Note that this also means that (T', R^-, T) is invertible. It follows from Definition 3.2.3 that, in a chromatic model of $\mathcal{K}$, a domain element of type T has at most one R-successor of type T' and vice versa, a domain element of type T' has at most one R^--successor of type T. This means that, in order to ensure that a (chromatic) model can be built according to some mosaic for $\mathcal{K}$, it suffices to ensure that there is a way to pair up domain elements of type T that require an R-successor of type T' with domain elements of type T' that require an R^--successor of type T. If elements d and d' are paired together, d' serves as the required R-successor to d and d serves as the R^--successor for d'. This is precisely what the condition M3 does – it ensures that, when building a model $\mathcal{I}$ of $\mathcal{K}$ according to some mosaic N for $\mathcal{K}$, the total number of elements in $\Delta^{\mathcal{I}}$ of type T that require an R-successor of type T' is equal to the number of elements in $\Delta^{\mathcal{I}}$ of type T' that require an R^--successor of type T, which means that such a pairing exists.

The intuition behind the condition M5 relates to the following situation. Assume we have $p(a, b) \in \mathcal{A}$ asserting that, in a model of $\mathcal{K}$, the constant a has as a p-successor the constant b and assume a participates in a concept name A. Now assume there are axioms in $\mathcal{T}$ that allow us to infer that all p-neighbors of a must participate in a concept name B. We can conclude that b must participate in B. Conditions M5 (a)-(b) represent different ways a can communicate information to b. Finally, M6 (a)-(b) express

that if we additionally know that b participates in B and there is some counting axiom $A \sqsubseteq= nr.B \in \mathcal{T}$ requiring a to have exactly n r-successors that are B, then a must explicitly record this in the ρ component of the tile that describes it. In other words, it cannot be that a is described by a tile $\tau = (T, \rho)$ in which b is not recorded in ρ as an r-successor. This is rooted in the condition T4 in Definition 3.2.5 that requires that tiles explicitly record information about all the neighbors that are used for satisfying counting axioms.

Before we state the main result of this chapter, we introduce one more auxiliary notion. Notice that the way tiles were designed makes sure that all role inclusions are handled correctly for those connections that are encoded within the tiles. For example, if a tile $\tau = (T, \rho)$ contains some triple $(R, T', k) \in \rho$ asking for an r-neighbor of type T' and we have some RI $r \sqsubseteq s$, then this neighbor also acts as an s-neighbor and we must have $s \in R$. However, recall that tiles do not keep track of all connections that might hold in a model of the KB $\mathcal{K}$ but rather only those that are triggered by counting axioms. In particular, tiles may not encode all the connections that are asserted by the ABox, which means that we have to handle those manually. For this reason, we introduce the notion of KBs that *respect role inclusions*. Intuitively, if $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$ respects role inclusions, then we can close $\mathcal{A}$ under the role inclusions in the TBox and not obtain blatant contradictions (like, e.g., both $p(a, b) \in \mathcal{A}$ and $\neg p(a, b) \in \mathcal{A}$) nor would we violate the closed predicates. Obviously, if this is not possible, then $\mathcal{K}$ has no model.

Definition 3.2.11. *Let $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$ be an $\mathcal{ALCHOIQ}$ KB. We say that $\mathcal{K}$* respects role inclusions*, if the following conditions hold:*

1. *if $r \sqsubseteq s \in \mathcal{T}$ and $\neg s(a, b) \in \mathcal{A}$, then $r(a, b) \notin \mathcal{A}$,*
2. *if $r^- \sqsubseteq s \in \mathcal{T}$ and $\neg s(a, b) \in \mathcal{A}$, then $r(b, a) \notin \mathcal{A}$,*
3. *if $r \in \Sigma \cap \mathsf{N}_R$, $s \sqsubseteq r \in \mathcal{T}$ and $s(a, b) \in \mathcal{A}$, then $r(a, b) \in \mathcal{A}$, and*
4. *if $r \in \Sigma \cap \mathsf{N}_R$, $s^- \sqsubseteq r \in \mathcal{T}$ and $s(a, b) \in \mathcal{A}$, then $r(b, a) \in \mathcal{A}$.*

With the previous definition in mind, we state the result that establishes the connection between mosaic existence and satisfiability.

Theorem 3.2.12. *Let $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$ be an $\mathcal{ALCHOIQ}$ KB. $\mathcal{K}$ is satisfiable if and only if $\mathcal{K}$ respects role inclusions and there exists a mosaic for $\mathcal{K}$.*

3.2.3 Correctness of Theorem 3.2.12

Consider an arbitrary knowledge base $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$. We dedicate this subsection to showing that the result in Theorem 3.2.12 indeed holds. To this end, we first show that if $\mathcal{K}$ respects role inclusions and there is a mosaic N for $\mathcal{K}$, we can construct a model $\mathcal{I}$ of $\mathcal{K}$ from N. We then show that the converse also holds, i.e., if we are given a model $\mathcal{I}$ of $\mathcal{K}$, then $\mathcal{K}$ respects role inclusions and we can easily construct a mosaic for $\mathcal{K}$.

From Mosaic to Model Furthermore, assume $\mathcal{K}$ respects role inclusions and let N be a mosaic for $\mathcal{K}$. We next show how to construct a model $\mathcal{I}$ of $\mathcal{K}$.

We have already explained that each tile can be seen as a description of a domain element and a part of its neighborhood and that a mosaic N tells us how many instances of each tile we need in order to build a model. Therefore, we build our domain $\Delta^{\mathcal{I}}$ by instantiating each tile τ $N(\tau)$ times:

$$\Delta^{\mathcal{I}} = \{(T, \rho)_i : (T, \rho) \in \text{Tiles}(\mathcal{K}), 1 \leq i \leq N((T, \rho))\}.$$

Intuitively, the domain element τ_i corresponds to the i-th instance of tile τ. Recall that, due to the condition M1 in Definition 3.2.9, for each constant $a \in \mathsf{N_I}(\mathcal{K})$, there is exactly one tile $\tau_a = (T_a, \rho_a)$ for which $\{a\} \in T_a$ and $N(\tau_a) > 0$. Moreover, as $N(\tau_a) = 1$ (again due to M1), there is a *single* instance of τ_a in $\Delta^{\mathcal{I}}$ and this instance represents the constant a. For ease of presentation, for the rest of this construction, we use a and $\tau_{a1} = (T_a, \rho_a)_1$ interchangeably, to denote the domain element in $\Delta^{\mathcal{I}}$ that represents the constant a.

Recall that in a tile $\tau = (T, \rho)$, the type T tells us in which basic concepts the instances of τ participate. Keeping this in mind, we next construct the extensions of concept names occurring in $\mathcal{K}$. To this end, for every $B \in \mathsf{N_C}(\mathcal{K})$, we let

$$B^{\mathcal{I}} = \{(T, \rho)_i \in \Delta^{\mathcal{I}} : B \in T, 1 \leq i \leq N((T, \rho))\}.$$

Constructing the extensions of roles is a more complex matter and it is done in multiple steps.

Step 1. We first ensure that $\mathcal{I}$ satisfies the ABox $\mathcal{A}$. For each $p(a, b) \in \mathcal{A}$ and each role r such that $p \sqsubseteq r \in \mathcal{T}$, we set $(a, b) \in r^{\mathcal{I}}$, if r is a role name, and $(b, a) \in (r^-)^{\mathcal{I}}$, otherwise.

Moreover, for each $A \sqsubseteq= nr.B \in \mathcal{T}$ we do the following. If $a \in A^{\mathcal{I}}$, $b \in B^{\mathcal{I}}$, and $p \sqsubseteq r \in \mathcal{T}$, then, due to M6, there is some $(R, T_b, k) \in \rho_a$ with $r \in R$. Then, for each $s \in R$, we let $(a, b) \in s^{\mathcal{I}}$, if s is a role name, and $(b, a) \in s^{\mathcal{I}}$, otherwise. Similarly, if $a \in B^{\mathcal{I}}$, $b \in A^{\mathcal{I}}$ and $p^- \sqsubseteq r \in \mathcal{T}$, there is some $(R, T_a, k) \in \rho_b$ with $r \in R$. For each $s \in R$, we let $(b, a) \in s^{\mathcal{I}}$, if s is a role name, and $(a, b) \in s^{\mathcal{I}}$, otherwise.

Step 2. Next, for all $T, T' \in \text{Types}(\mathcal{K})$ and $R \subseteq \mathsf{N_R^+}(\mathcal{T})$, if (T, R, T') is invertible, we do the following. Due to M3, we know that the following equation is satisfied:

$$\sum_{\substack{(T,\rho)\in\mathsf{Tiles}(\mathcal{K}),\\ (R,T',k)\in\rho}} N((T, \rho)) = \sum_{\substack{(T',\rho')\in\mathsf{Tiles}(\mathcal{K}),\\ (R^-,T,l)\in\rho'}} N((T', \rho'))$$

Let $X_{T,R,T'}$ and $Y_{T,R,T'}$ be the following sets of domain elements

$$X_{T,R,T'} = \{(T,\rho)_i \in \Delta^{\mathcal{I}} : \text{there exists } k \geq 1 \text{ s.t. } (R,T',k) \in \rho \text{ and } 1 \leq i \leq N((T,\rho))\}$$
$$Y_{T,R,T'} = \{(T',\rho')_i \in \Delta^{\mathcal{I}} : \text{there exists } l \geq 1 \text{ s.t. } (R^-,T,l) \in \rho' \text{ and } 1 \leq i \leq N((T',\rho'))\}$$

Intuitively, the set $X_{T,R,T'}$ contains all domain elements with the unary type T that require an R-successor of type T' and the set $Y_{T,R,T'}$ contains all domain elements with the unary type T' that require an R^--successor of type T. Due to M3, these two sets have the same cardinality, i.e., $|X_{T,R,T'}| = |Y_{T,R,T'}|$, and so there exists a bijection between them. We choose one such bijection, denoted by $f_{T,R,T'}$, and we do the following. For every $e \in X_{T,R,T'}$ and every $s \in R$, we set $(e, f_{T,R,T'}(e)) \in s^{\mathcal{I}}$ if s is a role name and $(f_{T,R,T'}(e), e) \in (s^-)^{\mathcal{I}}$ otherwise. Intuitively, this connects via an R-arc, every domain element of type T that requires an R-neighbor of type T' to one domain element of type T' that requires an R^--neighbor of type T.

Observe that if (T,R,T') is invertible then so is (T',R^-,T). To avoid conflict, when choosing the bijections $f_{T,R,T'}$ and $f_{T',R^-,T}$ we ensure that $f_{T',R^-,T} = f^{-1}_{T,R,T'}$.

Further, observe that $f_{T,R,T'}$ has the following property: for constants $a,b \in \Delta^{\mathcal{I}}$ and a role name r s.t. $r \in R$ (resp. $r^- \in R$), $a \in X_{T,R,T'}$ and $b \in Y_{T,R,T'}$, if $(a,b) \in r^{\mathcal{I}}$ (resp. $(b,a) \in r^{\mathcal{I}}$) was constructed in step 1, then $f_{T,R,T'}(a) = b$. To see this, recall that, due to the definition of sets $X_{T,R,T'}$ and $Y_{T,R,T'}$, we have that $T = T_a$ and $T' = T_b$. Due to the condition M1, there is exactly one element in $\Delta^{\mathcal{I}}$ of the type T_a (the constant a) and exactly one element in $\Delta^{\mathcal{I}}$ of the type T_b (the constant b). Therefore, $X_{T,R,T'} = \{a\}$, $Y_{T,R,T'} = \{b\}$, and so it must be that $f_{T,R,T'}(a) = b$.

Step 3. For each domain element $e = (T,\rho)_i \in \Delta^{\mathcal{I}}$ and each type T' we do the following. We say that a counting axiom $A \sqsubseteq\!= nr.B \in \mathcal{T}$ is *relevant (for e and T')* if $A \in T$ and $B \in T'$. Moreover, we say that a counting role r is *relevant (for e and T')*, if it occurs in a relevant axiom for e and T. Further, due to the condition T2, each $(R,T',k) \in \rho$ contains at least one relevant role.

Let $\rho_{T'} = \{(R,T',k) : (R,T',k) \in \rho\}$, $\Delta^{\mathcal{I}}_{T'} = \{d \in \Delta^{\mathcal{I}} : d = (T',\rho')_j\}$, and let $\Delta^{\mathcal{I}}_{e,T'}$ be the set of all domain elements of type T' that are an r-successor of e, for some relevant counting role r:

$$\Delta^{\mathcal{I}}_{e,T'} = \{d \in \Delta^{\mathcal{I}} : d = (T',\rho')_j \text{ and } (e,d) \in r^{\mathcal{I}}, r \text{ is a relevant counting role}\}.$$

Let $w_{e,T'} : \Delta^{\mathcal{I}}_{e,T'} \to \rho_{T'}$ be a total injective function such that for every $d \in \Delta^{\mathcal{I}}_{T'}$ with $w_{e,T'}(d) = (R,T',k)$, $(e,d) \in s^{\mathcal{I}}$, if $s \in R$. We say that d is a witness to (R,T',k) and that (R,T',k) is witnessed if it is in the image of $w_{e,T'}$. We explain why it is possible to find such a function a little later.

Observe that the number of witnessed triples in $\rho_{T'}$ is exactly $|\Delta^{\mathcal{I}}_{e,T'}|$. Due to M4, we have that $|\rho_{T'}| \leq |\Delta^{\mathcal{I}}_{T'}|$, which means that the number of triples in $\rho_{T'}$ still

requiring a witness from $\Delta^{\mathcal{I}}_{T'}$ is smaller than or equal to the number of elements in $\Delta^{\mathcal{I}}_{T'} \setminus \Delta^{\mathcal{I}}_{e,T'}$, i.e., the number of domain elements of type T' that were not yet used as witnesses for any relevant counting roles. To finish the construction, for each triple $(R, T', k) \in \rho_{T'}$ that does not have a witness, we take a fresh $d \in \Delta^{\mathcal{I}} \setminus \Delta^{\mathcal{I}}_{e,T'}$ and for each $s \in R$, we let $(e, d) \in s^{\mathcal{I}}$, if s is a role name, and $(d, e) \in (s^-)^{\mathcal{I}}$, otherwise.

Now, we still need to show that we can find the total function $w_{e,T'}$ as described above. To this end, consider an arbitrary domain element $d = \Delta^{\mathcal{I}}_{e,T'}$ and a relevant counting role r. We make the following case distinction:

- Assume that $(e, d) \in r^{\mathcal{I}}$ was constructed in step 1. In this case, the domain elements e and d represent some constants $a, b \in \mathsf{N_I}(\mathcal{K})$, respectively. Thus, we have $\{a\} \in T$ and $\{b\} \in T'$. Moreover, we have that, for some $p \in \mathsf{N_R}(\mathcal{K})$, either (i) $p(a, b) \in \mathcal{A}$ and $p \sqsubseteq r \in \mathcal{T}$ or (ii) $p(b, a) \in \mathcal{A}$ and $p^- \sqsubseteq r \in \mathcal{T}$. Due to M6 in Definition 3.2.9, there exists an $(R, T', k) \in \rho_{T'}$ such that $r \in R$. Moreover, due to M1, $|\Delta^{\mathcal{I}}_{T'}| = 1$ and so there cannot be more than one triple $\rho_{T'}$, otherwise M4 would be violated. Observe that during step 1, we ensured that for this unique triple $(R, T', k) \in \rho_{T'}$, we have $(e, d) \in s^{\mathcal{I}}$, for all $s \in R$. We let $w_{e,T'}(d) = (R, T', k)$.
- Assume that $(e, d) \in r^{\mathcal{I}}$ was constructed during step 2. In this case, $(e, d) \in r^{\mathcal{I}}$ was constructed when we considered some $R \subseteq \mathsf{N}^+_{\mathsf{R}}(\mathcal{K})$ s.t. (T, R, T') is invertible and we have $e \in X_{T,R,T'}$ and $f_{T,R,T'}(e) = d$. We thus know that there must be some triple $(R, T', k) \in \rho_{T'}$ such that $r \in R$ and (T, R, T') is invertible. Moreover, due to the condition T5 (d) in Definition 3.2.5, there is only one such triple in ρ. Note also that, during this step, we ensured that $(e, d) \in s^{\mathcal{I}}$, for all $s \in R$. We let $w_{e,T'}(d) = (R, T', k)$.
- Assume we constructed $(e, d) \in r^{\mathcal{I}}$ in step 3 as a witness for some previously unwitnessed triple $(R, T, k) \in \rho'_T$ with $r^- \in R$ when considering some element $d = (T', \rho')_j \neq e$ and the unary type T. Recall, once again, that due to the condition T2, R must contain a relevant counting role s for d and T, which makes (T', R, T) is invertible. However, all invertible triples are dealt with in step 2, in particular also (R, T, k), so it cannot be the case that (R, T, k) has no witness. This is a contradiction to $(e, d) \in r^{\mathcal{I}}$ being constructed in step 3 while considering an element different from e, because we only create new connections for unwitnessed triples.

Finally, to see that $w_{e,T'}$ is an injective function, consider some $(R, T', k) \in \rho_{T'}$ and assume that there are two different $d_1, d_2 \in \Delta^{\mathcal{I}}_{e,T'}$ such that $w_{e,T'}(d_1) = w_{e,T'}(d_2)$. Note that as both d_1 and d_2 are of type T', T' cannot be a constant type, otherwise M1 would be violated. Therefore, d_1 and d_2 were both determined to be witnesses for (R, T', k) in step 2, which means that $f_{T,R,T'}(e) = d_1 = d_2$. Thus, d_1 and d_2 are in fact the same domain element.

Observation 3.2.13. *The way in which we construct role extensions ensures that, if* $(e,d) \in r^{\mathcal{I}}$*, for some domain elements* $e = (T,\rho)_i, d = (T',\rho')_j$ *and some role* r*, then at least one of the following must hold:*

(i) $e = a$*,* $d = b$ *for some* $a, b \in \mathsf{N_I}(\mathcal{K})$*, and* $p(a,b) \in A$ *for some role name* p *s.t.* $p \sqsubseteq r \in \mathcal{T}$*,*

(ii) $e = a$*,* $d = b$ *for some* $a, b \in \mathsf{N_I}(\mathcal{K})$*, and* $p(b,a) \in A$ *for some role name* p *s.t.* $p^- \sqsubseteq r \in \mathcal{T}$*,*

(iii) there is some $(R, T', l) \in \rho$ *s.t.* $r \in R$*, or*

(iv) there is some $(R', T, k) \in \rho'$ *s.t.* $r^- \in R'$*.*

We next show that the interpretation we constructed is indeed a model of $\mathcal{K}$.

Satisfaction of $\mathcal{T}$. Consider an arbitrary axiom α in $\mathcal{T}$.

- α is of the shape $B_1 \sqcap \cdots \sqcap B_{k-1} \sqsubseteq B_k \sqcup \cdots \sqcup B_m$: Let $e = (T,\rho)_i \in \Delta^{\mathcal{I}}$ and assume that $e \in (B_1 \sqcap \cdots \sqcap B_{k-1})^{\mathcal{I}}$. This means that $e \in B_j^{\mathcal{I}}$, for all $1 \leq j < k$. By construction of $\mathcal{I}$, this implies that $B_j \in T$, for all $1 \leq j < k$. By condition T3 in the definition of tiles, we have that there exists l, $k \leq l \leq m$, such that $B_l \in T$. Thus, $e \in B_l^{\mathcal{I}}$ and so $e \in (B_k \sqcup \cdots \sqcup B_m)^{\mathcal{I}}$.

- α is of the shape $A \sqsubseteq \forall r.B$: Assume $e = (T,\rho)_i \in \Delta^{\mathcal{I}}$, $d = (T',\rho')_j \in \Delta^{\mathcal{I}}$, $(e,d) \in r^{\mathcal{I}}$, $e \in A^{\mathcal{I}}$, and $d \notin B^{\mathcal{I}}$. By construction, this means that $A \in T$ and $B \notin T'$. Notice that due to conditions T5 (a) and T5 (b) in the definition of tiles, there can be no $(R, T', k) \in \rho$ s.t. $r \in R$ and no $(R', T, l) \in \rho'$ s.t. $r^- \in R$. In view of Observation 3.2.13, this means that $e = a$ and $d = b$, for some constants $a, b \in \mathsf{N_I}(\mathcal{K})$ and either (i) $p(a,b) \in \mathcal{A}$ and $p \sqsubseteq r \in \mathcal{T}$ or (ii) $p(b,a)$ and $p^- \sqsubseteq r \in \mathcal{T}$, for some role name p. However, this contradicts the conditions M5 and M1, and so $B \in T'$.

- α is of the shape $r \sqsubseteq s$: This axiom is satisfied due to Observation 3.2.13 and the condition T5 (c) in the definition of tiles as well as the fact that in step 1 of the construction of role extensions, whenever we add $(a,b) \in s^{\mathcal{I}}$, we do the same for all roles $r \sqsubseteq s \in \mathcal{T}$.

- α is of the shape $A \sqsubseteq\!= nr.B$: Let $e = (T,\rho)_i$ be an arbitrary element in $\Delta^{\mathcal{I}}$ and assume $e \in A^{\mathcal{I}}$, i.e. $A \in T$. We need to prove that $|\{d \in \Delta^{\mathcal{I}} : (e,d) \in r^{\mathcal{I}}, d \in B^{\mathcal{I}}\}| = n$. Recall that, due to the condition T4 in Definition 3.2.5, we have $|\{(R, T', k) \in \rho : r \in R \text{ and } B \in T'\}| = n$.

By construction of $B^{\mathcal{I}}$, we have that

$$|\{d \in \Delta^{\mathcal{I}} : (e,d) \in r^{\mathcal{I}}, d \in B^{\mathcal{I}}\}| = |\{d = (T',\rho')_j \in \Delta^{\mathcal{I}} : (e,d) \in r^{\mathcal{I}}, B \in T'\}|.$$

Let T' be an arbitrary unary type with $B \in T'$. Recall that we use $\Delta^{\mathcal{I}}_{T'}$ to denote the set of elements of $\Delta^{\mathcal{I}}$ that have the unary type T', i.e., that are of the form $(T',\rho')_j$, and $\rho_{T'}$ to denote the set of triples in ρ that mention the unary type T', i.e., are of the form (R,T',k). In step 3 of the construction of role extensions, we show that each domain element $d \in \Delta^{\mathcal{I}}_{T'}$ with $(e,d) \in r^{\mathcal{I}}$ is a witness to exactly one $(R,T',k) \in \rho_{T'}$ with $r \in R$. Moreover, we also show that each such triple $(R,T',k) \in \rho_{T'}$ with $r \in R$ has exactly one associated witness. Hence,

$$|\{d \in \Delta^{\mathcal{I}}_{T'} : (e,d) \in r^{\mathcal{I}}\}| = |\{(R,T',k) \in \rho_{T'} : r \in R\}|,$$

Summing up over all unary types that contain B we get:

$$\begin{aligned}|\{d \in \Delta^{\mathcal{I}} : d \in B^{\mathcal{I}}, (e,d) \in r^{\mathcal{I}}\}| &= |\{d \in \Delta^{\mathcal{I}}_{T'} : T' \in \mathrm{Types}(\mathcal{K}), B \in T', (e,d) \in r^{\mathcal{I}}\}| \\ &= |\{(R,T',k) \in \rho : r \in R, B \in T'\}| \\ &= n.\end{aligned}$$

Satisfaction of $\mathcal{A}$. Let $A(c) \in \mathcal{A}$ and recall that $c = (T_c,\rho_c)_1 \in \Delta^{\mathcal{I}}$. As $(T_c,\rho_c)_1$ is a tile, $A \in T_c$ due to T6 in Definition 3.2.5. By construction of $A^{\mathcal{I}}$, we have that $c \in A^{\mathcal{I}}$ and so $\mathcal{I}$ satisfies $A(c)$. Let $\neg A(c) \in \mathcal{A}$. Then, by condition T7 in Definition 3.2.5, $A \notin T_c$ and so $c \notin A^{\mathcal{I}}$. Hence, $\mathcal{I}$ satisfies $\neg A(c)$.

Let $p(a,b) \in \mathcal{A}$, where $p \in \mathsf{N_R}$. Step 1 in our construction of $\mathcal{I}$ involved adding $(a,b) \in r^{\mathcal{I}}$ for each assertion $p(a,b) \in \mathcal{A}$ and role $r \sqsubseteq p \in \mathcal{T}$. Due to the closure assumption for the TBox, we have $p \sqsubseteq p \in \mathcal{T}$ and thus $(a,b) \in p^{\mathcal{I}}$. Hence, $\mathcal{I}$ satisfies $p(a,b)$.

Let $\neg p(a,b) \in \mathcal{A}$, where $p \in \mathsf{N_R}$ and assume towards a contradiction that $(a,b) \in p^{\mathcal{I}}$. Due to our assumption that $\mathcal{K}$ respects role inclusions, $(a,b) \in p^{\mathcal{I}}$ could not have been constructed due to conditions (i) or (ii) in Observation 3.2.13. Further, due to condition T5 (e) in Definition 3.2.5, there is no $(R,T_b,k) \in \rho_a$ such that $p \in R$ and, due to condition T5 (f), there is no $(R,T_a,k) \in \rho_b$ such that $p^- \in R$. This is a contradiction to Observation 3.2.13 and $(a,b) \in p^{\mathcal{I}}$ could not have been constructed in the first place.

Satisfaction of the closed predicates. Let $A \in \mathsf{N_C} \cap \Sigma$ and $e = (T,\rho)_i$ be an arbitrary element in $A^{\mathcal{I}}$. By construction of $A^{\mathcal{I}}$, we have that $A \in T$. As $A \in \Sigma$, by condition T8 in the definition of tiles for $\mathcal{K}$, there exists some $c \in \mathsf{N_I}$ such that $\{c\} \in T$ and $A(c) \in \mathcal{A}$. Thus, it must be the case that $e = c$. As $A(c) \in \mathcal{A}$, $\mathcal{I}$ respects closed concepts.

Let $r \in \mathsf{N_R} \cap \Sigma$ and let $e_1 = (T_1,\rho_1)_i$ and $e_2 = (T_2,\rho_2)_j$ arbitrary elements of $\Delta^{\mathcal{I}}$ for which $(e_1,e_2) \in r^{\mathcal{I}}$ holds. We now make a case distinction based on at which point in the construction of $\mathcal{I}$ we set $(e_1,e_2) \in r^{\mathcal{I}}$.

- We set $(e_1, e_2) \in r^{\mathcal{I}}$ in step 1 of the construction, in order to satisfy the ABox. In this case, $e_1 = a$ and $e_2 = b$, for some $a, b \in \mathsf{N_I}(\mathcal{K})$ for which either (i) $p(a, b) \in \mathcal{A}$ and $p \sqsubseteq r \in \mathcal{T}$ or (ii) $p(b, a) \in \mathcal{A}$ and $p^- \sqsubseteq r \in \mathcal{T}$. Due to the assumption that $\mathcal{K}$ respects role inclusions and therefore, for all $r \in \Sigma \cap \mathsf{N_R}$, if $p \sqsubseteq r \in \mathcal{T}$ (resp. $p^- \sqsubseteq r \in \mathcal{T}$) and $p(a, b) \in \mathcal{A}$ (resp. $p(b, a)$), then $r(a, b) \in \mathcal{A}$, we can conclude that $r(a, b) \in \mathcal{A}$ and thus the closed role is not violated.

- We set $(e_1, e_2) \in r^{\mathcal{I}}$ in any of the other cases. In this case, in view of Observation 3.2.13, we can see that a prerequisite to setting $(e_1, e_2) \in r^{\mathcal{I}}$ is that there is some $(R, T_2, k) \in \rho_1$ such that $r \in R$. Then, due to condition T9 in the Definition 3.2.5, we have that there exists $c \in \mathsf{N_I}(\mathcal{K})$ occurring in $\mathcal{A}$ such that $\{c\} \in T_1$. Hence, $e_1 = c$. Further, also due to the same condition, we have that there is some $d \in \mathsf{N_I}(\mathcal{K})$ such that $\{d\} \in T_2$ and therefore $e_2 = d$. The final part of the condition T9 requires states that $r(c, d) \in \mathcal{A}$. Hence $(e_1, e_2) \in r^{\mathcal{I}}$ does not violate the closed role.

From Model to Mosaic Recall that $\mathcal{K}$ has a model if and only if it has a chromatic model and let $\mathcal{I}$ be a chromatic model of $\mathcal{K}$. Obviously, as $\mathcal{I}$ is a model of $\mathcal{K}$, $\mathcal{K}$ must respect role inclusions. We next show that we can construct a mosaic N for $\mathcal{K}$ from $\mathcal{I}$.

Consider an element $e \in \Delta^{\mathcal{I}}$. We first extract a tile $\tau_e = (T_e, \rho_e)$ such that e is an instance of τ_e. To this end, we let $T_e = \mathsf{t}(e)$. Further, we define ρ_e as follows. Since $\mathcal{I}$ is a model of $\mathcal{K}$, $\mathcal{I}$ satisfies every counting axiom in $\mathcal{T}$. This means that for every axiom $\alpha \in \mathcal{T}$ that is of the type $A \sqsubseteq\!= nr.B$, if $A \in \mathsf{t}(e)$, then there exist exactly n elements $e_1^\alpha, \ldots, e_n^\alpha \in \Delta^{\mathcal{I}}$ such that $(e, e_i^\alpha) \in r^{\mathcal{I}}$ and $e_i^\alpha \in B^{\mathcal{I}}$, for $i = 1, \ldots, n$. We let

$$E(e) = \{e_i^{A \sqsubseteq\!= nr.B} \in \Delta^{\mathcal{I}} : A \sqsubseteq\!= nr.B \in \mathcal{T}, A \in \mathsf{t}(e), 1 \leq i \leq n\},$$
$$\rho_e = \{(R, T', i) : T' \in \mathrm{Types}(\mathcal{T}),\ R \subseteq \mathsf{N_R^+}(\mathcal{T}),\ 1 \leq i \leq |\{e' \in E(e) : \mathsf{rt}(e, e') = R, t(e') = T'\}|\}.$$

It is easy to verify that τ_e is indeed a tile for $\mathcal{K}$. We note that condition T5 (d) is satisfied due to chromaticity of $\mathcal{I}$.

We next define a function $N : \mathrm{Tiles}(\mathcal{T}) \to \mathbb{N}^*$ with

$$N(\tau) = |\{e \in \Delta^{\mathcal{I}} : \tau_e = \tau\}|.$$

Finally, in order to show that N is indeed a mosaic for $\mathcal{K}$, we need to show that N satisfies conditions M1-M6 in Definition 3.2.9.

M1. Let a be an arbitrary nominal in $\mathsf{N}_\mathsf{C}^+(\mathcal{T})$. Due to SNA, the constant a is the only element in $\Delta^\mathcal{I}$ that participates in $\{a\}$, i.e., the only element that has $\{a\}$ in its type. This means that there can be no other element $e \in \Delta^\mathcal{I}$ such that $\tau_a = \tau_e$, and so $N(\tau_a) = 1$. Further, as τ_a is the only tile with $N(\tau_a) > 1$ that contains $\{a\}$ in its unary type, we have $\sum_{(T,\rho)\in \mathrm{Tiles}(\mathcal{T}),\{a\}\in T} N((T,\rho)) = N(\tau_a) = 1$.

M2. As we do not allow interpretations with empty domains, there must be at at least one element $e \in \Delta^\mathcal{I}$. Then, $N(\tau_e) \geq 1$ and M2 is satisfied.

M3. Let T, T' be two arbitrary types for $\mathcal{K}$, let R be an arbitrary subset of $\mathsf{N}_\mathsf{R}^+(\mathcal{K})$ s.t. (T, R, T') is invertible. Since (T, R, T') is invertible, by definition, there exists an axiom $\alpha = A \sqsubseteq= nr.B \in \mathcal{T}$ s.t. $A \in T$, $B \in T'$, and $r \in R$. Consider an arbitrary domain element $d \in \Delta^\mathcal{I}$ with $\mathsf{t}(d) = T$. We know by construction of $E(d)$, that $d' \in E(d)$, for every $d' \in \Delta^\mathcal{I}$ with $\mathsf{t}(d') = T'$ and $\mathsf{rt}(d, d') = R$. Further, by construction of ρ_d, we have the following observation:

Observation 3.2.14. *For each $d \in \Delta^\mathcal{I}$ with $t(d) = T$, there is a one-to-one correspondence between the triples in ρ_d of the form (R, T', l) and the elements $d' \in \Delta^\mathcal{I}$ with $t(d') = T'$ and $rt(d, d') = R$.*

Moreover, as (T, R, T') is invertible and $\mathcal{I}$ is chromatic, we know that there cannot be two distinct elements $d', d'' \in \Delta^\mathcal{I}$ s.t. $\mathsf{t}(d') = \mathsf{t}(d'') = T'$ and $\mathsf{rt}(d, d') = \mathsf{rt}(d, d'') = R$. This leads us to the following observation:

Observation 3.2.15. *For each $d \in \Delta^\mathcal{I}$ with $t(d) = T$, $|\{d' \in \Delta^\mathcal{I} : t(d') = T', rt(d, d') = R\}| \leq 1$.*

Further, as (T', R^-, T) is also invertible, analogously we reach the following two observations:

Observation 3.2.16. *For each $d' \in \Delta^\mathcal{I}$ with $t(d') = T'$, there is a one-to-one correspondence between the triples in $\rho_{d'}$ of the form (R^-, T, l) and the elements $d \in \Delta^\mathcal{I}$ with $t(d) = T$ and $rt(d', d) = R^-$.*

Observation 3.2.17. *For each $d' \in \Delta^\mathcal{I}$ with $t(d') = T'$, $|\{d \in \Delta^\mathcal{I} : t(d) = T, rt(d', d) = R^-\}| \leq 1$.*

We are now ready to show that M3 holds:

$$\sum_{\substack{(T,\rho)\in \mathrm{Tiles}(\mathcal{K}),\\ (R,T',k)\in\rho}} N((T,\rho)) = \sum_{\substack{(T,\rho)\in \mathrm{Tiles}(\mathcal{K}),\\ (R,T',k)\in\rho}} |\{d \in \Delta^{\mathcal{I}} : T_d = T \text{ and } \rho_d = \rho\}| =$$

$$|\{d \in \Delta^{\mathcal{I}} : T_d = T \text{ and } (R,T',k) \in \rho_d, \text{ for some } k \geq 1\}| \overset{Obs.\ 3.2.14}{=}$$

$$|\{d \in \Delta^{\mathcal{I}} : T_d = T \text{ and } \mathsf{rt}(d,d') = R, T_{d'} = T', \text{ for some } d' \in \Delta^{\mathcal{I}}\}| \overset{Obs.\ 3.2.15}{=}$$

$$|\{(d,d') \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} : T_d = T, T_{d'} = T', \mathsf{rt}(d,d') = R\}| =$$

$$|\{(d',d) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} : T_d = T, T_{d'} = T', \mathsf{rt}(d',d) = R^-\}| \overset{Obs.\ 3.2.17}{=}$$

$$|\{d' \in \Delta^{\mathcal{I}} : T_{d'} = T' \text{ and } \mathsf{rt}(d',d) = R^-, T_d = T, \text{ for some } d \in \Delta^{\mathcal{I}}\}| \overset{Obs.\ 3.2.16}{=}$$

$$|\{d' \in \Delta^{\mathcal{I}} : T_{d'} = T' \text{ and } (R^-,T,l) \in \rho_{d'}, \text{ for some } l \geq 1\}| =$$

$$\sum_{\substack{(T',\rho')\in \mathrm{Tiles}(\mathcal{K}),\\ (R^-,T,l)\in\rho'}} |\{d' \in \Delta^{\mathcal{I}} : T_{d'} = T' \text{ and } \rho_{d'} = \rho'\}| = \sum_{\substack{(T',\rho')\in \mathrm{Tiles}(\mathcal{K}),\\ (R^-,T,l)\in\rho'}} N((T',\rho'))$$

M4. Let (T,ρ) be an arbitrary tile and T' be an arbitrary type for $\mathcal{K}$. Assume that $N((T,\rho)) > 0$. This means that there is an element $d \in \Delta^{\mathcal{I}}$ s.t. $T_d = T$ and $\rho_d = \rho$. By construction of ρ_d, it is easy to see that:

$$\begin{aligned} |\{(R,T',k) : (R,T',k) \in \rho_d\}| &= |\{d' \in \Delta^{\mathcal{I}} : d' \in E(d), T_{d'} = T'\}| \\ &\leq |\{d' \in \Delta^{\mathcal{I}} : T_{d'} = T'\}|. \end{aligned}$$

Finally, by construction of N, we have that

$$|\{d' \in \Delta^{\mathcal{I}} : T_{d'} = T'\}| = \sum_{(T',\rho')\in \mathrm{Tiles}(\mathcal{K})} N((T',\rho')).$$

Thus, we get that

$$\sum_{(T',\rho')\in \mathrm{Tiles}(\mathcal{K})} N((T',\rho')) \geq |\{(R,T',k) : (R,T',k) \in \rho\}|.$$

M5. Let $\{a\}$ and $\{b\}$ be two nominals in $\mathsf{N}_\mathsf{C}^+(\mathcal{K})$, $A, B \in \mathsf{N}_\mathsf{C}(\mathcal{K})$, and assume at least one of the conditions (a)-(b) is satisfied. In this case, it is easy to see that $a \in A^{\mathcal{I}}$ implies $b \in B^{\mathcal{I}}$. Further, assume that $\sum_{\substack{(T,\rho)\in \mathrm{Tiles}(\mathcal{K}),\\ \{a\}\in T, A\in T}} N((T,\rho)) > 0$. This means that there is a tile $\tau = (T,\rho)$ s.t. $\{a\}, A \in T$. As N satisfies M1, there can only be one such tile, namely the tile τ_a obtained from the domain element a. By construction of tiles, this means that $a \in A^{\mathcal{I}}$. Thus, $b \in B^{\mathcal{I}}$ and so $B \in T_b$. As $N(\tau_b) > 0$, we get that M5 is satisfied.

M6. Let $\{a\}$ and $\{b\}$ be two nominals in $\mathsf{N}_\mathsf{C}^+(\mathcal{K})$, $A, B \in \mathsf{N}_\mathsf{C}(\mathcal{K})$, $r \in \mathsf{N}_\mathsf{R}^+(\mathcal{K})$, $\tau' = (T',\rho')$ be a tile for $\mathcal{K}$ s.t. $\{\{b\}, B\} \subseteq T'$. Further assume that at least one of the conditions a-b is satisfied, which means that we can conclude that $(a,b) \in r^{\mathcal{I}}$.

Now, assume that $N(\tau') \geq 1$. In this case, we know that $b \in B^{\mathcal{I}}$. Let $\tau = (T, \rho)$ be a tile with $\{\{a\}, A\} \subseteq T$ and $|\{(R, T', k) \in \rho : r \in R\}| \leq 0$, and assume that $N(\tau) \geq 1$. Then, we have that $a \in A^{\mathcal{I}}$, and so (due to the condition (a)/(b)) a must have exactly n r-neighbors of type B, with b being one of them. Recall that, by construction of tiles, this information is recorded in ρ, i.e., there is a triple $(R, T', k) \in \rho$ s.t. $R = \mathsf{rt}(a, b)$, which is a contradiction to $|\{(R, T', k) \in \rho : r \in R\}| \leq 0$. Thus, it must be that $N(\tau) \leq 0$.

3.2.4 Enriched Systems of Integer Linear Inequalities

So far, we have shown that we can reduce knowledge base satisfiability to deciding the existence of mosaics. To decide the latter, we next show how to build a system of integer linear inequalities with implications whose solutions over $\mathbb{N}^*$ correspond to the mosaics, for some given knowledge base $\mathcal{K}$.

We begin by formally introducing the notion *enriched systems of integer linear inequalities*.

Definition 3.2.18. *An* enriched system (of integer linear inequalities) *is a tuple* $(V, \mathcal{E}, I)$, *where:*

- V *is a set of variables,*
- $\mathcal{E}$ *is a set of inequalities of the form*

$$a_1 \cdot x_1 + \cdots + a_n \cdot x_n + c \leq b_1 \cdot y_1 + \cdots + b_m \cdot y_m + d, \tag{3.1}$$

  where $a_1, \ldots, a_n, b_1, \ldots, b_m$ *are non-negative integers called* variable coefficients, c, d *are non-negative integers called* constant coefficients, *and* $x_1, \ldots, x_n, y_1, \ldots, y_m \in V$, *and*

- I *is the set of implications of the form*

$$\alpha \Rightarrow \beta, \tag{3.2}$$

  where α *and* β *are inequalities of the form (3.1).*

Notice that if $I = \emptyset$, the enriched system $(V, \mathcal{E}, I)$ (or simply $(V, \mathcal{E})$) is a system of integer linear inequalities in the ordinary sense. We call such systems *ordinary systems (of integer linear inequalities)*.

Solutions to enriched systems can be defined over any domain $\mathbb{D}$ of numbers.

Definition 3.2.19. *A* solution S *over* $\mathbb{D}$ *to an enriched system* $(V, \mathcal{E}, I)$ *is a function* $S : V \to \mathbb{D}$ *that assigns to each variable* $x \in V$ *a value* $S(x)$ *over* $\mathbb{D}$ *such that all inequalities and implications are satisfied.*

In this thesis, we are interested in solutions of enriched settings over the aforementioned set $\mathbb{N}^*$ of non-negative integers with a special value $\aleph_0$ that represents infinity. We next state some useful results on the complexity of deciding whether an enriched system has a solution over $\mathbb{N}^*$.

We begin by recalling some well-known results about solving ordinary systems. It has been proven many times that deciding whether an ordinary system has a non-negative integer solution is possible in NP (see, e.g., [Pap81, KM78] and references therein). The proof in [KM78] hinges on showing that if an ordinary system has a non-negative integral solution, then it has one in which all values are bounded by a single exponential function of the size of the system. This means that we can use only polynomially many bits to encode such values, so we can devise a guess-and-check procedure for checking the existence of solutions. We formally state these results below.

Proposition 3.2.20. *Given a finite ordinary system of integer linear inequalities $\mathcal{S} = (V, \mathcal{E})$ whose coefficients are in $\{0, \pm 1, \ldots, \pm a\}$. If $\mathcal{S}$ has a solution over $\mathbb{N}$, then $\mathcal{S}$ has a solution over $\mathbb{N}$ where all values are bounded by $(|V| + |\mathcal{E}|) \cdot ((|\mathcal{E}|) \cdot a)^{2|\mathcal{E}|+1}$.*

Proposition 3.2.21. *[KM78] Given an ordinary system of integer linear inequalities $\mathcal{S}$, deciding whether $\mathcal{S}$ has a solution over $\mathbb{N}$ is possible in NP.*

It has also been shown that the same bounds hold when it comes to considering solutions over $\mathbb{N}^*$. It is easy to see that an ordinary system $\mathcal{S}$ has a solution over $\mathbb{N}^*$ if and only if there is a way to assign $\aleph_0$ to certain variables in a way that ensures that (i) all inequalities involving these infinite values are satisfied and (ii) removing all inequalities from $\mathcal{S}$ that involve variables that have been assigned to infinity results in a system that has a solution over $\mathbb{N}$. In view of Proposition 3.2.20, this observation already reveals that if $\mathcal{S}$ has a solution over $\mathbb{N}$, it also has one in which all finite values are bounded by a single exponential function in the size of $\mathcal{S}$. As before, this implies that checking whether $\mathcal{S}$ has a solution over $\mathbb{N}^*$ is possible in NP. This has been observed many times in the literature, e.g., Lemma 18 in [PH05]. We show that the results stated above also generalize to enriched systems, which was also observed in Theorem 13 in [GGBIG$^+$19] (the extended version of [GGI$^+$20]).

Proposition 3.2.22. *Let $\mathcal{S} = (V, \mathcal{E}, I)$ be an enriched system of integer linear inequalities in which all coefficients are in $\{0, \pm 1, \ldots, \pm a\}$. We have that the following hold:*

1. *If $(V, \mathcal{E}, I)$ has a solution over $\mathbb{N}$, then it also has a solution over $\mathbb{N}$ where all values are bounded by $(|V| + |I| + |\mathcal{E}|) \cdot ((|\mathcal{E}| + |I|) \cdot a)^{2(|\mathcal{E}|+|I|)+1}$.*
2. *If $(V, \mathcal{E}, I)$ has a solution over $\mathbb{N}^*$, then it also has a solution over $\mathbb{N}^*$ where all finite values are bounded by $(|V| + |I| + |\mathcal{E}|) \cdot ((|\mathcal{E}| + |I|) \cdot a)^{2(|\mathcal{E}|+|I|)+1}$.*
3. *Deciding whether $\mathcal{S}$ has a solution over $\mathbb{N}^*$ (resp. $\mathbb{N}$) is in NP.*

Proof. We show this result in a couple of steps. The first step is to remove all implications from $\mathcal{S}$.

For an inequality $\alpha = a_1 \cdot x_1 + \cdots + a_n \cdot x_n + c \leq b_1 \cdot y_1 + \cdots + b_m \cdot y_m + d$, let $\overline{\alpha}$ denote the negation of α, i.e., $\overline{\alpha} = b_1 \cdot y_1 + \cdots + b_m \cdot y_m + d + 1 \leq a_1 \cdot x_1 + \cdots + a_n \cdot x_n + c$.

It is easy to see that $\mathcal{S}$ has a solution over some domain of numbers $\mathbb{D}$ if and only if there is some set of inequalities $\mathcal{X}$ such that $(V, \mathcal{E} \cup \mathcal{X})$ has a solution over $\mathbb{D}$ and:

- $|\mathcal{X}| = |I|$, and
- for every implication $\alpha \implies \beta \in I$, either $\beta \in \mathcal{X}$ or $\overline{\alpha} \in \mathcal{X}$.

Indeed, assume S is a solution of $\mathcal{S}$ over $\mathbb{D}$ and let $\mathcal{X}$ be the following set of inequalities:

$$\mathcal{X} = \{\beta : \alpha \implies \beta \in I \text{ s.t. } \beta \text{ is satisfied by } S\}$$
$$\{\overline{\alpha} : \alpha \implies \beta \in I \text{ s.t. } \beta \text{ is not satisfied by } S\},$$

Note that, by definition of solutions, if S does not satisfy β, then S must not satisfy α, and thus S satisfies $\overline{\alpha}$, for each $\alpha \implies \beta \in I$. Thus, S is obviously a solution to $(V, \mathcal{E} \cup \mathcal{X})$.

Conversely, let $\mathcal{X}'$ be a set of inequalities as described above such that $(\mathcal{V}, \mathcal{E} \cup \mathcal{X})$ has a solution S' over $\mathbb{D}$. As $\mathcal{X}'$ contains either $\overline{\alpha}$ or β, for each $\alpha \implies \beta \in I$, S' is also a solution to $(\mathcal{V}, \mathcal{E}, I)$.

It is now straightforward to show that the claims of the theorem hold:

1. Assume that $\mathcal{S}$ has a solution over $\mathbb{N}$. Then, there is a set of inequalities $\mathcal{X}$ s.t. (i) $|\mathcal{X}| = |I|$, (ii) $\mathcal{S}' = (\mathcal{V}, \mathcal{E} \cup \mathcal{X})$ has a solution over $\mathbb{N}$ and (iii) every solution of $\mathcal{S}'$ over $\mathbb{N}$ is also a solution to $\mathcal{S}$.

   Due to Proposition 3.2.20, if $\mathcal{S}'$ has a solution over $\mathbb{N}$, then it has one in which every value is bounded by $(|V| + |\mathcal{X} \cup \mathcal{E}|) \cdot ((|\mathcal{E} \cup \mathcal{X}|) \cdot a)^{2(|\mathcal{E} \cup \mathcal{X}|)+1}$. The claim follows from $|\mathcal{X}| = |I|$.

2. As in the item above, if we assume that $\mathcal{S}$ has a solution over $\mathbb{N}^*$, then there is a set of inequalities $\mathcal{X}$ s.t. (i) $|\mathcal{X}| = |I|$, (ii) $\mathcal{S}' = (\mathcal{V}, \mathcal{E} \cup \mathcal{X})$ has a solution over $\mathbb{N}^*$ and (iii) every solution of $\mathcal{S}'$ over $\mathbb{N}^*$ is also a solution to $\mathcal{S}$. It was shown in [PH05] that if $\mathcal{S}'$ has a solution over $\mathbb{N}^*$, then it has one in which all finite values are bounded by $(|V| + |\mathcal{X} \cup \mathcal{E}|) \cdot ((|\mathcal{E} \cup \mathcal{X}|) \cdot a)^{2(|\mathcal{E} \cup \mathcal{X}|)+1}$. The claim once again follows from $|\mathcal{X}| = |I|$.

3. To show that we can decide whether $\mathcal{S}$ has a solution over $\mathbb{N}^*$ in NP, we can non-deterministically guess $\mathcal{X}$, the subset of variables that are assigned to $\aleph_0$ as well as the bit encoding of the remainder of the finite values (recall we only need polynomially many bits for this!) and then check in polynomial time whether our guess is valid.

4. Analogous to the item above.

□

3.2.5 Existence of Mosaics via Enriched Systems

Consider an $\mathcal{ALCHOIQ}$ knowledge base $\mathcal{K}$ with closed predicates. We show how to obtain an enriched system $\mathcal{S}_\mathcal{K} = (V, \mathcal{E}, I)$ from $\mathcal{K}$ such that there is a one-to-one correspondence between the solutions of $\mathcal{S}_\mathcal{K}$ over $\mathbb{N}^*$ and the mosaics for $\mathcal{K}$.

Definition 3.2.23. *Let $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$ be an $\mathcal{ALCHOIQ}$ KB with closed predicates. We denote by $\mathcal{S}_\mathcal{K}$ the enriched system of integer linear inequalities obtained from $\mathcal{K}$ as follows:*

- *We associate a variable x_τ to every tile $\tau \in Tiles(\mathcal{K})$ and we let $V = \{x_\tau : \tau \in Tiles(\mathcal{K})\}$.*
- *We obtain the set of inequalities $\mathcal{E}$ from the conditions M1-M3 by replacing every occurrence of $N(\tau)$ by x_τ, for every $\tau \in Tiles(\mathcal{K})$. Note that we treat the equations $\alpha = \beta$ in M1 and M3 as two inequalities $\alpha \leq \beta$ and $\beta \leq \alpha$.*
- *We obtain the set of implications I from the conditions M4-M5 in Definition 3.2.9 by, once again, replacing every occurrence of $N(\tau)$ by x_τ, for every $\tau \in Tiles(\mathcal{K})$.*

Proposition 3.2.24. *Let $\mathcal{K}$ be an $\mathcal{ALCHOIQ}$ knowledge base with closed predicates. For each solution S over $\mathbb{N}^*$ of $\mathcal{S}_\mathcal{K}$ there exists a mosaic N for $\mathcal{K}$ such that $S(x_\tau) = N(\tau)$, for every $\tau \in Tiles(\mathcal{K})$, and vice versa.*

Although implicit, we can see from the construction in the proof of Theorem 3.2.12 and the proposition above, that if a KB $\mathcal{K}$ has a model $\mathcal{I}$ then the enriched system $\mathcal{S}_\mathcal{K}$ has a solution in which the sum of all the variables representing tiles for $\mathcal{K}$ with A in their unary type is equal to the number of domain elements in the extension of A in $\mathcal{I}$, for all concept names $\mathcal{A} \in \mathsf{N_C}(\mathcal{K})$. Moreover, the converse also holds. We next make this explicit.

Observation 3.2.25. *If $\mathcal{K}$ has a model $\mathcal{I}$, then $\mathcal{S}$ has a solution S in which $|A^\mathcal{I}| = \sum_{(T,\rho)\in Tiles(\mathcal{K}),A\in T} S(x_{(T,\rho)})$, for all $A \in \mathsf{N_C}(\mathcal{T})$. Moreover, if $\mathcal{S}_\mathcal{K}$ has a solution S then $\mathcal{K}$ has a model $\mathcal{I}$ in which $|A^\mathcal{I}| = \sum_{(T,\rho)\in Tiles(\mathcal{K}),A\in T} S(x_{(T,\rho)})$, for all $A \in \mathsf{N_C}(\mathcal{T})$.*

Size of the enriched system and complexity results. We can now analyze the size of the obtained enriched system $\mathcal{S}_\mathcal{K} = (V, \mathcal{E}, I)$ relative to the size of the knowledge base $\mathcal{K}$.

Recall that every variable in V corresponds to a tile in Tiles($\mathcal{K}$), so we start by computing the number of distinct tiles for $\mathcal{K}$. First off, each unary type T for $\mathcal{K}$ consists of a set of concept names that is a subset of $\mathsf{N_C}(\mathcal{T})$ together with at most one nominal $\{a\}$, where $a \in \mathsf{N_I}(\mathcal{K})$. This means that we have at most $2^{|\mathsf{N_C}(\mathcal{T})|} \cdot (|\mathsf{N_I}(\mathcal{K})| + 1)$ different unary types in Types($\mathcal{K}$). In other words, the number of different unary types is exponential in the

size of $\mathcal{K}$, but only polynomial if we consider $\mathcal{T}$ to be fixed. Similarly, each role type R is a subset of $\mathsf{N}_\mathsf{R}^+(\mathcal{T})$, so we have at most $2^{|\mathsf{N}_\mathsf{R}^+(\mathcal{T})|}$ different role types for $\mathcal{K}$. Every tile τ consists of two components: a unary type T and a set ρ of triples of the form (R, T', k), where R is a role type, T' is a type and k is an integer. Moreover, due to the condition T2, ρ contains at most $m_\mathcal{T} \cdot c_\mathcal{T}$ triples. Also, observe that due to T1, the choice of k is not free. It is now easy to see that the number of different tiles in Tiles($\mathcal{K}$) (and therefore also the set of variables V) is also exponential in the size of $\mathcal{K}$ and polynomial if $\mathcal{T}$ is fixed. More precisely, we have the following:

Observation 3.2.26.

$$|V| \leq (2^{|\mathsf{N}_C(\mathcal{T})|})^{m_\mathcal{T} \cdot c_\mathcal{T}+1} \cdot (|\mathsf{N}_I(\mathcal{K})| + 1)^{m_\mathcal{T} \cdot c_\mathcal{T}+1} \cdot (2^{|\mathsf{N}_R^+(\mathcal{T})|})^{m_\mathcal{T} \cdot c_\mathcal{T}}$$

We next see that similar bounds apply to the size of $\mathcal{E}$ and I.

Conditions M1 and M2 only introduce polynomially many inequalities in the size of $\mathcal{K}$. Indeed, M1 introduces two inequalities for every individual occurring in $\mathcal{K}$ and M2 introduces a single inequality. The condition M3 introduces at most two inequalities for every invertible triple (T, R, T'), where T and T' are unary types and R is a role type. In the worst case, this means that M3 introduces exponentially many inequalities in the size of $\mathcal{K}$, however, as before, this number is polynomial if $\mathcal{T}$ is considered fixed. Regarding the size needed to encode a single inequality $q \in \mathcal{E}$, we note that q contains every variable $x \in V$ at most twice (once on the LHS and once on the RHS). Moreover, all integer constants occurring in q are bounded by $c_\mathcal{T}$. Therefore, q is also exponential in the size of $\mathcal{K}$ and polynomial for a fixed $\mathcal{T}$.

Observation 3.2.27.

$$|\mathcal{E}| \leq 2|\mathsf{N}_I(\mathcal{K})| + 1 + 2(2^{|\mathsf{N}_C(\mathcal{T})|} \cdot (|\mathsf{N}_I(\mathcal{K})| + 1)^2 \cdot 2^{|\mathsf{N}_C(\mathcal{T})|} \cdot 2^{|\mathsf{N}_R^+(\mathcal{T})|})$$

Moving on to I, condition M5 introduces polynomially many implications in the size of $\mathcal{K}$ – at most one implication for each pair of individuals and each pair of concept names in $\mathcal{K}$. Conditions M4 and M6 introduce exponentially many implications in the size of $\mathcal{K}$ (polynomial if $\mathcal{T}$ is fixed). Indeed, in the worst case, M4 introduces one implication per pair (τ, T'), where τ is a tile and T' is a unary type for $\mathcal{K}$, and M6 introduces one implication for every pair a, b of individuals occurring in $\mathcal{K}$, tile τ and role $r \in \mathsf{N}_\mathsf{R}^+(\mathcal{K})$. Moreover, each of these implications consists of two inequalities whose size is once again at most exponential in $\mathcal{K}$ (polynomial if $\mathcal{T}$ is fixed). Thus the same bounds apply to all of I.

Observation 3.2.28.

$$|I| \leq |\mathsf{N}_I(\mathcal{K})|^2 \cdot |\mathsf{N}_C(\mathcal{K})|^2 + |V| \cdot 2^{|\mathsf{N}_C(\mathcal{K})|} \cdot (|\mathsf{N}_I(\mathcal{K})| + 1) + |\mathsf{N}_I(\mathcal{K})|^2 \cdot |V| \cdot |\mathsf{N}_R^+(\mathcal{K})|.$$

Finally, we can see from the construction of $\mathcal{S}_\mathcal{K}$ that the coefficients in this enriched system do not exceed $\max\{1, c_\mathcal{T}\}$.

Observation 3.2.29. *If c is a coefficient in $\mathcal{S}_\mathcal{K}$, then $c \in \{0, \pm 1, \ldots, \pm c_\mathcal{T}\}$.*

Summing up the discussion above, we have the following result for the size of $\mathcal{S}_\mathcal{K}$.

Proposition 3.2.30. *Let $\mathcal{K}$ be an $\mathcal{ALCHOIQ}$ KB with closed predicates. The size of the system $\mathcal{S}_\mathcal{K}$ is exponential in the size of $\mathcal{K}$ and polynomial in the size of $\mathcal{K}$, if $\mathcal{T}$ is fixed.*

Data complexity of $\mathcal{ALCHOIQ}$ with closed predicates Finally, we show that the problem of deciding whether an $\mathcal{ALCHOIQ}$ KB with closed predicates admits a model is NP-complete in data complexity by relying on the previously introduced characterization together with the well-known results from the realm of integer programming.

We begin by recalling a well-known result on the data complexity of plain $\mathcal{ALC}$.

Proposition 3.2.31. *[Sch93] Deciding whether an $\mathcal{ALC}$ KB has a model is NP-hard in data complexity.*

The proposition above provides the lower data complexity bound for reasoning in $\mathcal{ALCHOIQ}$ with closed predicates. The matching upper bound comes from Proposition 3.2.22 and the fact that we can decide in NP whether an enriched system has a solution over $\mathbb{N}^*$.

Theorem 3.2.32. *Let $\mathcal{K}$ be an $\mathcal{ALCHOIQ}$ KB with closed predicates. Deciding whether $\mathcal{K}$ has a model is NP-complete in data complexity.*

Proof. Checking whether $\mathcal{K}$ respects role inclusions can easily be done in polynomial time. According to Theorem 3.2.12 and Proposition 3.2.24, deciding whether $\mathcal{K}$ that respects role inclusions is satisfiable amounts to deciding whether $\mathcal{S}_\mathcal{K}$ has a solution over $\mathbb{N}^*$. The latter is possible in NP in the size of the input system (Proposition 3.2.22). As $\mathcal{S}_\mathcal{K}$ is polynomial in the size of the data (Proposition 3.2.30, we get that the KB satisfiability in $\mathcal{ALCHOIQ}$ with closed predicates is in NP. The matching lower bound comes from Theorem 3.2.31. □

3.3 The "simpler" $\mathcal{ALCHOIF}$

$\mathcal{ALCHOIF}$ is a sublogic of $\mathcal{ALCHOIQ}$ – indeed, we can replace $\mathsf{func}(r)$ by $\top \sqsubseteq \leq 1 r.\top$ and yield an equivalent $\mathcal{ALCHOIQ}$ KB. As such, the upper bound on the data complexity of the knowledge satisfiability problem from the previous section is directly transferred over to $\mathcal{ALCHOIF}$ with closed predicates. Moreover, due to Theorem 3.2.31, this bound is tight so reasoning in $\mathcal{ALCHOIF}$ is not easier. Nevertheless, we present a simplified characterization of the satisfiability problem for $\mathcal{ALCHOIF}$ with closed predicates in terms of mosaics that will come in useful in the later chapters. The intuition behind the tiles and the mosaics remains the same as before, but the conditions placed on them were

specifically tailored for functionality axioms instead of the general number restrictions. More importantly, having a characterization specifically tailored for $\mathcal{ALCHOIF}$ will be useful in Chapter 5, where we extend this logic as well as the notion tiles and mosaics to provide capturing results for the complexity class coNP.

Similarly to before, we assume w.l.o.g. that all KBs in this section are given in normal form. To this end, we use the $\mathcal{ALCHOIF}$ normal form introduced in Definition 3.1.13. Consider an $\mathcal{ALCHOIF}$ KB $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$. Much like in the case of $\mathcal{ALCHOIQ}$, a tile $\tau = (T, \rho)$ for $\mathcal{K}$ consists of a unary type T and a component ρ that stores the relevant part of the neighborhood of the instances of τ. However, instead of triples of the form (R, T', k), ρ now consists of pairs (R, T'), where R is a role type and T' is a unary type for $\mathcal{K}$. The reason for this is that the only type of "at least" number restrictions in $\mathcal{ALCHOIF}$ KBs are axioms of the type $A \sqsubseteq \exists r.B$ (i.e., $A \sqsubseteq \geq 1 r.B$)). This means that we need only *one* witness in ρ per existential axiom in $\mathcal{T}$, so there is no need to keep track of multiple different copies of the same type of neighbor. We next give a formal definition of tiles.

Definition 3.3.1. *Given an $\mathcal{ALCHOIF}$ KB $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$, a* tile *for $\mathcal{K}$ is a tuple $\tau = (T, \rho)$, where $T \in \mathit{Types}(\mathcal{K})$ and ρ is a set of pairs (R, T'), where $R \subseteq \mathsf{N}_R^{\pm}(\mathcal{K})$, $T' \in \mathit{Types}(\mathcal{K})$ and the following conditions are satisfied:*

TF1. $|\rho| \leq |\mathcal{T}|$

TF2. *If* $B_1 \sqcap \cdots \sqcap B_{k-1} \sqsubseteq B_k \sqcup \cdots \sqcup B_m \in \mathcal{T}$ *and* $\{B_1, \ldots, B_{k-1}\} \subseteq T$, *then* $\{B_k, \ldots, B_m\} \cap T \neq \emptyset$

TF3. *If* $A \sqsubseteq \exists r.B \in \mathcal{T}$ *and* $A \in T$, *then there is* $(R, T') \in \rho$ *such that* $r \in R$ *and* $B \in T'$

TF4. *For all* $(R, T') \in \rho$, *the following hold:*

- *(a)* *If* $A \sqsubseteq \forall r.B \in \mathcal{T}$, $A \in T$ *and* $r \in R$, *then* $B \in T'$
- *(b)* *If* $A \sqsubseteq \forall r.B \in \mathcal{T}$, $A \in T'$ *and* $r^- \in R$, *then* $B \in T$
- *(c)* *If* $r \sqsubseteq s \in \mathcal{T}$ *and* $r \in R$, *then* $s \in R$

TF5. *If* $\mathsf{func}(r) \in \mathcal{T}$, *then* $|\{(R, T') \in \rho : r \in R\}| \leq 1$

TF6. *If* $A(b) \in \mathcal{A}$ *and* $\{b\} \in T$, *then* $A \in T$

TF7. *If* $\neg A(b) \in \mathcal{A}$ *and* $\{b\} \in T$, *then* $A \notin T$

TF8. *For all* $(R, T') \in \rho$, *the following hold:*

- *(a)* *If* $p(a, b) \in \mathcal{A}$, $\{p \sqsubseteq r, \mathsf{func}(r)\} \subseteq \mathcal{T}$, $\{a\} \in T$ *and* $r \in R$, *then* $\{b\} \in T'$
- *(b)* *If* $p(a, b) \in \mathcal{A}$, $\{p \sqsubseteq r, \mathsf{func}(r^-)\} \subseteq \mathcal{T}$, $\{b\} \in T$, *and* $r^- \in R$, *then* $\{a\} \in T'$
- *(c)* *If* $\neg p(a, b) \in \mathcal{A}$, $r \sqsubseteq p \in \mathcal{T}$, $\{a\} \in T$, *and* $r \in R$, *then* $\{b\} \notin T'$

(d) If $\neg p(a,b) \in \mathcal{A}$, $r \sqsubseteq p^- \in \mathcal{T}$, $\{b\} \in T$, *and* $r \in R$, *then* $\{a\} \notin T'$

TF9. If $A \in \Sigma \cap \mathsf{N_C}$ *and* $A \in T$, *then there exists* $c \in \mathsf{N_I}$ *such that* $\{c\} \in T$ *and* $A(c) \in \mathcal{A}$

TF10. If $r \in \Sigma \cap \mathsf{N_R}$, *then for all* $(R, T') \in \rho$ *with* $r \in R$, *there exist* $c, d \in \mathsf{N_I}$ *such that* $\{c\} \in T$, $\{d\} \in T'$ *and* $r(c,d) \in \mathcal{A}$.

We briefly explain the intuitions behind the conditions. Consider some tile τ that describes some domain element d. Conditions TF1-TF4 are inherited from [GGI+20] and ensure that the description of d is consistent with the statements in $\mathcal{T}$, from the perspective of d. Condition TF2 ensures the satisfaction of axioms of the type $B_1 \sqcap \cdots \sqcap B_{k-1} \sqsubseteq B_k \sqcup \cdots \sqcup B_m \in \mathcal{T}$, condition TF3 guarantees that d has a witness for every axiom $\exists R.B \in \mathcal{T}$, conditions TF4 (a)-(c) ensure that d and its neighbors respect axioms of the type $A \sqsubseteq \forall r.B$ and $r \sqsubseteq s$, and condition TF5 ensures that the functionality assertions in $\mathcal{T}$ are not violated. We further add the conditions that ensure that the description of d is compatible with the assertions in $\mathcal{A}$ (conditions TF6-TF8), and those that ensure that the closed predicates are respected (conditions TF9-TF10).

We next define mosaics for $\mathcal{ALCHOIF}$ KBs.

Definition 3.3.2. *Let* $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$ *be an* $\mathcal{ALCHOIF}$ *KB. A* mosaic *for* $\mathcal{K}$ *is a function* $N : \mathit{Tiles}(\mathcal{K}) \to \mathbb{N}^*$ *such that:*

MF1. For every $\{c\} \in \mathsf{N_C^+}(\mathcal{K})$: $\displaystyle\sum_{\substack{(T,\rho)\in \mathit{Tiles}(\mathcal{K}),\\ \{c\}\in T}} N((T,\rho)) = 1$

MF2. The following inequality is satisfied: $\displaystyle\sum_{\tau \in \mathit{Tiles}(\mathcal{K})} N(\tau) \geq 1$

MF3. For every pair $T, T' \in \mathit{Types}(\mathcal{K})$ *and every* $R \subseteq \mathsf{N_R^+}(\mathcal{K})$ *with* $r \in R$ *and* $\mathsf{func}(r^-) \in \mathcal{T}$, *the following holds:*

$$\sum_{\substack{(T,\rho)\in \mathit{Tiles}(\mathcal{K}),\\ (R,T')\in\rho}} N((T,\rho)) \leq \sum_{\substack{(T',\rho')\in \mathit{Tiles}(\mathcal{K}),\\ (R^-,T)\in\rho'}} N((T',\rho'))$$

MF4. For all $(T,\rho) \in \mathit{Tiles}(\mathcal{K})$ *and* $(R,T') \in \rho$ *the following holds: if* $N((T,\rho)) > 0$, *then there exists* ρ' *such that* $(T',\rho') \in \mathit{Tiles}(K)$ *and* $N((T',\rho')) > 0$.

MF5. For all $\{a\}, \{b\} \in \mathsf{N_C^+}(\mathcal{K})$ *and all* $A, B \in \mathsf{N_C}(\mathcal{K})$, *if there exist* $p, r \in \mathsf{N_R^+}(\mathcal{K})$ *for which any of the following conditions hold:*

(a) $p(a,b) \in \mathcal{A}$, $p \sqsubseteq r \in \mathcal{T}$ *and* $A \sqsubseteq \forall r.B \in \mathcal{T}$,

(b) $p(b,a) \in \mathcal{A}$, $p \sqsubseteq r^- \in \mathcal{T}$ *and* $A \sqsubseteq \forall r.B \in \mathcal{T}$,

(c) $p(a,b) \in \mathcal{A}$, $p \sqsubseteq r \in \mathcal{T}$ *and* $A \sqsubseteq \exists r.B \in \mathcal{T}$ *and* $\mathsf{func}(r) \in \mathcal{T}$, *or*

(d) $p(b,a) \in \mathcal{A}$, $p \sqsubseteq r^- \in \mathcal{T}$ *and* $A \sqsubseteq \exists r.B \in \mathcal{T}$ *and* $\mathsf{func}(r) \in \mathcal{T}$,

we have that the following implication holds:

$$\sum_{\substack{(T,\rho)\in \mathit{Tiles}(\mathcal{K}),\\ \{a\}\in T, A\in T}} N((T,\rho)) > 0 \implies \sum_{\substack{(T',\rho')\in \mathit{Tiles}(\mathcal{K}),\\ \{b\}\in T', B\in T'}} N((T',\rho')) > 0$$

Conditions MF1 and MF2 are the same as the conditions M1 and M2 for $\mathcal{ALCHOIQ}$ mosaics. The condition MF3 ensures that we have enough domain elements to satisfy the functionality assertions in $\mathcal{T}$. More precisely, assume we are given types T, T' and $R \subseteq \mathsf{N}_\mathsf{R}^+(\mathcal{K})$, where a role r whose inverse is functional occurs in R. Let n be the number of domain elements in a model of $\mathcal{K}$ that participate in basic concepts in T and have an outgoing arc labeled by R to some neighbor of type T'. Since r^- is functional, each domain element can "accept" at most one incoming arc labeled by R, or equivalently, has at most one outgoing arc labeled by R^-. Thus, to build a model of $\mathcal{K}$, there must be n or more elements of type T' that have an outgoing arc to an element of type T that is labeled by R^-. The condition MF4 says that if we obtain a domain element d by instantiating a tile (T,ρ) and ρ asserts the existence of some neighbors of d, we can also instantiate tiles to provide suitable neighbors for d. Condition MF5 relates to the following situation. Assume we have $p(a,b) \in \mathcal{A}$ asserting that, in a model of $\mathcal{K}$, a has as a p-neighbor the constant b and assume a participates in a concept name A. Now assume there are axioms in $\mathcal{T}$ that allow us to infer that all p-neighbors of a must participate in a concept name B. We can conclude that b must participate in B. Constant a can "send" such a message to b either via universal axioms in $\mathcal{T}$ or via existential axioms and functionality assertions in $\mathcal{T}$. Conditions MF5 (a)-(d) represent different ways a can communicate information to b.

The following result establishes the connection between the existence of mosaics and the satisfiability of $\mathcal{ALCHOIF}$ KBs with closed predicates. Similarly to Definition 3.2.11, we first define what it means for an $\mathcal{ALCHOIF}$ KB with closed predicates to respect role inclusions.

Definition 3.3.3. *Let* $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$ *be an* $\mathcal{ALCHOIF}$ *KB with closed predicates. We say that* $\mathcal{K}$ respects role inclusions *if it satisfies the following conditions:*

1. *for all* $r \in \Sigma \cap \mathsf{N_R}$ *and* $s \sqsubseteq r \in \mathcal{T}$ *(resp.* $s^- \sqsubseteq r$*): if* $s(a,b) \in \mathcal{A}$, *then* $r(a,b) \in \mathcal{A}$ *(resp.* $r(b,a)$*), and*
2. *for all* $r \in \mathsf{N}_\mathsf{R}^+(\mathcal{A})$ *with* $\mathsf{func}(r) \in \mathcal{T}$ *and all* $a \in \mathsf{N_I}(\mathcal{A})$, *the set*

$$\{b : p(a,b) \in \mathcal{A}, p \sqsubseteq r \in \mathcal{T}\} \cup \{b : p(b,a) \in \mathcal{A}, p^- \sqsubseteq r \in \mathcal{T}\}$$

has at most one element.

Intuitively, these two conditions ensure that if we were to compute the closure of $\mathcal{A}$ under the role inclusions from $\mathcal{T}$ we would not violate any closed predicates or functionality assertions. The first condition states that for each role $r \in \Sigma$, if $p(a,b) \in \mathcal{A}$ and $p \sqsubseteq r \in \mathcal{T}$ then also $r(a,b) \in \mathcal{A}$ must hold. If this is not the case, every interpretation satisfying $p \sqsubseteq r$ would violate the closed predicates, and as such $\mathcal{K}$ would not have a model. The second condition checks for each constant a in $\mathcal{A}$ and each functional role r, whether $\mathcal{A}$ asserts the existence of more than one $r-$neighbor of a. If that is the case, then $\mathcal{K}$ also does not have a model.

Theorem 3.3.4. *Let $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$ be an $\mathcal{ALCHOIF}$ KB with closed predicates. $\mathcal{K}$ is satisfiable if and only if it respects role inclusions and there exists a mosaic for $\mathcal{K}$.*

As the proof of Theorem 3.3.4 is very similar to the proof of Theorem 3.2.12, we delegate it to the appendix to this thesis.

In view of the previous theorem, deciding whether $\mathcal{K}$ is satisfiable amounts to checking whether it respects role inclusions and whether there exists a mosaic for $\mathcal{K}$. The former can be easily done in polynomial time. To decide the latter, we can once again build an enriched system of linear inequalities whose solutions over $\mathbb{N}^*$ correspond to the mosaics for $\mathcal{K}$, and then use regular integer programming techniques to test for the existence of a solution.

3.4 Discussion

In this chapter, we presented a characterization of the knowledge base satisfiability problem in $\mathcal{ALCHOIQ}$ with closed predicates as a system of integer linear inequalities enriched with implications and we obtained a tight coNP data complexity bound. Our technique was inspired by several other works that use integer programming to characterize different reasoning problems in various DLs. Amongst them, the work of Pratt-Hartmann [PH09] stands out, showing the same data complexity bound for a fragment of first-order logic that subsumes many expressive DLs, in particular also standard $\mathcal{ALCHOIQ}$. However, our result is novel and it does not follow from previous results as we are in the setting with closed predicates that were not handled in [PH09]. Furthermore, the technique from Pratt-Hartmann was introduced for a fragment of FO that supports additional features that are not available in $\mathcal{ALCHOIQ}$. One of our aims was also to simplify the approach, defining a characterization specifically designed for the logic at hand that clearly reflects its (restricted) syntax of $\mathcal{ALCHOIQ}$ ontologies and semantics. Although our definitions of tiles and mosaics involve many conditions, all of them are quite intuitive and relatively easy to implement in Datalog$^\neg$, as we will see in Chapter 4. Unfortunately, one disadvantage of our method is that our tiles need to explicitly store all relevant neighbors of some domain element, the number of which depends on the actual numerical values in the TBox. This means that the size needed to encode a single tile is polynomial in the size of the given knowledge base only if we assume the unary encoding of the numbers occurring in the TBox. While this does not

affect the data complexity bound obtained in Theorem 3.2.32, it does have an effect on the succinctness of our Datalog$^{\neg}$ encoding, as discussed in the next chapter. Investigating how our approach can be adapted to also obtain results in the case of binary encoding is left as future work.

CHAPTER 4

Datalog Rewritability and Data Complexity of OMQs with Closed Predicates

In the previous chapter, we investigated the data complexity of deciding whether an arbitrary $\mathcal{ALCHOIQ}$ knowledge base with closed predicates admits a model and we showed that it is unaffected by the addition of the closed predicates. In this chapter, we shift our focus to the query answering problem in the presence of closed predicates. In this setting, an OMQ Q consists of a TBox $\mathcal{T}$, a set of predicates Σ whose extensions are considered to be complete, and a database query q. The certain answers to Q over an ABox $\mathcal{A}$ are then defined as those tuples of KB individuals that are the answers to q in every model of $\mathcal{T}$ and $\mathcal{A}$ that obeys the closed predicates from Σ. In this chapter, we are primarily interested in understanding the *relative expressiveness* of OMQs expressed in $\mathcal{ALCHOIQ}$ with closed predicates compared to more standard query languages like Datalog and its extensions. More precisely, we show the existence of a succinct *rewriting* (or a *translation*) of OMQs with closed predicates into Datalog$^{\neg}$, which is a procedure that takes as input an OMQ Q in the source language and produces a query q in the target language such that the certain answers to Q over $\mathcal{A}$ coincide with the answers to q over $\mathcal{A}$, for any ABox $\mathcal{A}$.

Contributions and publications. The key result presented in this chapter is a polynomial-time rewriting of instance (i.e., atomic) queries as well as "safe" first-order queries mediated by $\mathcal{ALCHOIQ}$ ontologies with closed predicates into Datalog$^{\neg}$. This translation is based on the satisfiability characterization of $\mathcal{ALCHOIQ}$ with closed predicates presented in the previous chapter, and it is done in two steps:

- Given an $\mathcal{ALCHOIQ}$ TBox $\mathcal{T}$ and a set Σ of closed predicates, we first show

how to construct a Datalog$^{\neg}$ program $\mathcal{P}_{sat}^{\mathcal{T},\Sigma}$ with the following property: $\mathcal{P}_{sat}^{\mathcal{T},\Sigma}$ together with some input data $\mathcal{A}$ has a stable model if and only if $(\mathcal{T},\Sigma,\mathcal{A})$ is satisfiable. This program is modular and it consists of two components that do the following. Based on the characterization from the previous chapter, the first component computes the relational representation of the system of integer linear inequalities and implications for $(\mathcal{T},\Sigma,\mathcal{A})$, and the second component solves the computed system. Moreover, $\mathcal{P}_{sat}^{\mathcal{T},\Sigma}$ is polynomial in the size of $(\mathcal{T},\Sigma)$.

- In the second step, we modify $\mathcal{P}_{sat}^{\mathcal{T},\Sigma}$ to preserve atomic consequences of $(\mathcal{T},\Sigma)$ over input data, which gives us a way to answer instance queries mediated by $\mathcal{ALCHOIQ}$ ontologies with closed predicates. We also introduce a large class of so-called *safe-range OMQs*, which support first-order queries where quantification is "guarded" by closed predicates and we show that they are Datalog$^{\neg}$-rewritable.

As an important consequence of our results, we get that the considered class of safe-range OMQs is co-NP-complete in data complexity, which follows from the complexity of Datalog with negation under the stable model semantics. This is a positive result, as it shows that closed predicates in $\mathcal{ALCHOIQ}$ do not increase the data complexity. We note that this is not obvious, as closed predicates are known to increase the data complexity in some cases, e.g., for ontology languages based on existential rules [BBtCP16]. As a final remark, we note that for expressive DLs that simultaneously support nominals, inverses, and number restrictions, the computational complexity of answering even very simple FO queries consisting only of existential quantification and conjunction (i.e., conjunctive queries) is unknown. It was shown in [GKL11] that conjunctive queries mediated by $\mathcal{ALCOIF}$ ontologies are co-N2ExpTime-hard, but so far there is no upper bound on the same problem, apart from decidability [RG10]. It is thus beneficial to consider fragments of FO queries for which one can pinpoint the complexity, especially those that can be answered at no additional cost.

The results presented in this chapter have been published in:

> [LOŠ24] **Sanja Lukumbuzya**, Magdalena Ortiz, Mantas Šimkus. "Datalog Rewritability and Data Complexity of $\mathcal{ALCHOIQ}$ with closed predicates". Artificial Intelligence (2024): 104099.

Organization In Section 4.1, we show how to construct a Datalog program with negation under the stable model semantics for deciding the satisfiability problem for $\mathcal{ALCHOIQ}$ KBs with closed predicates. In Section 4.2, we explain how this program can be augmented for answering certain types of OMQs and we present our complexity results. Conclusions and a discussion about potential future work are given in Section 4.3.

4.1 KB Satisfiability via Datalog$^\neg$

As a first step towards our goal of providing a rewriting of $\mathcal{ALCHOIQ}$ OMQs with closed predicates into Datalog, we show that, given an $\mathcal{ALCHOIQ}$ TBox $\mathcal{T}$ and a set of closed predicates Σ, we can construct a Datalog$^\neg$ program $\mathcal{P}_{sat}^{\mathcal{T},\Sigma}$ with the following properties:

- the size of $\mathcal{P}_{sat}^{\mathcal{T},\Sigma}$ is polynomial in the size of $\mathcal{T}$ and Σ, and
- the program $(\mathcal{P}_{sat}^{\mathcal{T},\Sigma}, \hat{A})$ has a stable model if and only if the KB $(\mathcal{T}, \Sigma, \mathcal{A})$ has a model. Notice that in this case, $\mathcal{A}$ is considered to be an external database, i.e., an input to $\mathcal{P}_{sat}^{\mathcal{T},\Sigma}$.

Since the considered variant of Datalog does not provide support for strong negation, we resort to a common technical trick to accommodate negative assertions in $\mathcal{A}$, i.e., assertions of the form $\neg q(\vec{a})$ (cf. the discussion about supporting strong negation in Datalog in Section 2.4). For each predicate q occurring in some TBox, we assume a predicate $\overline{q}$ that does not occur in any TBox. For an ABox $\mathcal{A}$, we denote by $\hat{\mathcal{A}}$ the set of atoms obtained from $\mathcal{A}$ by replacing all assertions of the form $\neg q(\vec{a})$ by $\overline{q}(\vec{a})$. Note that if $\mathcal{A}$ contains no negative assertions, $\mathcal{A}$ and $\hat{A}$ coincide.

We base our translation on the characterization of the KB satisfiability problem for $\mathcal{ALCHOIQ}$ with closed predicates via the existence of mosaics, as described in the previous chapter. Relying on Theorem 3.2.12, we construct the program $\mathcal{P}_{sat}^{\mathcal{T},\Sigma}$ consisting of two components, $\mathcal{P}_{sys}^{\mathcal{T},\Sigma}$ and $\mathcal{P}_{sol}^{\mathcal{T},\Sigma}$. These components communicate through a shared part of the signature summarized below, which is used for representing enriched systems of linear inequalities in a relational way. Assuming that, given an enriched system $\mathcal{S}$, every variable, every implication of the form (3.2), and every inequality of the form (3.1) (also including those occurring within implications) in $\mathcal{S}$ can be assigned a unique identifier (ID) encoded as a string over certain constants, we define the following predicates for encoding enriched systems of linear inequalities:

- A unary relation Cst storing constants, including 0 and 1.
- A binary relation LEQ defining a linear order over the constants in Cst, where 0 is the least constant w.r.t. LEQ.
- Relation Int storing integers relevant to the system using standard binary encoding.
- Relations Var, Im, and Iq, storing IDs of variables, implications, and inequalities (either stand-alone or occurring within implications) in the enriched system
- Relation Iq* storing IDs of inequalities in the enriched system that must be satisfied
- Relations $\mathsf{Iq}_\mathsf{L}^\mathsf{v}$ and $\mathsf{Iq}_\mathsf{R}^\mathsf{v}$ storing a pair $(\vec{q}, \vec{v})$, for each inequality ID $\vec{q}$ and a variable ID $\vec{v}$ for which the variable identified by $\vec{v}$ occurs on the left-hand side (LHS) (resp. right-hand side (RHS)) of the inequality identified by $\vec{q}$.

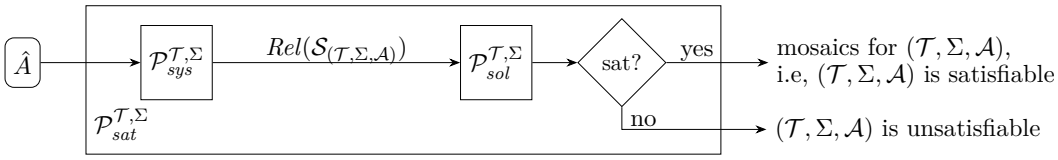


Figure 4.1: $\mathcal{P}^{\Sigma,\mathcal{T}}_{sat}$ and its components.

- Relations $\mathsf{lq}^{\mathsf{int}}_{\mathsf{L}}$ and $\mathsf{lq}^{\mathsf{int}}_{\mathsf{R}}$ storing a pair $(\vec{q},\vec{n})$, for each inequality ID $\vec{q}$ and an integer n (in binary encoding) for which n occurs on the LHS (resp. RHS) of the inequality identified by $\vec{q}$.

- Relations lm_{L} and lm_{R} storing a pair $(\vec{m},\vec{q})$, for each implication ID $\vec{m}$ and an inequality ID $\vec{q}$ for which the inequality identified by $\vec{q}$ occurs on the LHS (resp. RHS) of the implication identified by $\vec{m}$.

For ease of presentation, here we focus on the intuition behind the predicates, omitting technicalities like, e.g., their arities. These will become clear in the remainder of this chapter. We use $Rel(\mathcal{S})$ to denote some relational encoding of the enriched system $\mathcal{S}$ using the aforementioned predicates.

We now briefly explain what each component of $\mathcal{P}^{\mathcal{T},\Sigma}_{sat}$ does. For an input ABox $\mathcal{A}$, the program $(\mathcal{P}^{\mathcal{T},\Sigma}_{sys},\hat{\mathcal{A}})$ checks whether $(\mathcal{T},\Sigma,\mathcal{A})$ respects role inclusions as given in Definition 3.2.11, and, if that is the case, it computes $Rel(\mathcal{S}_{(\mathcal{T},\Sigma,\mathcal{A})})$, for the enriched system $\mathcal{S}_{(\mathcal{T},\Sigma,\mathcal{A})}$ defined in Section 3.2.5. The program $\mathcal{P}^{\mathcal{T},\Sigma}_{sol}$ then operates on the computed relational representation of $\mathcal{S}_{(\mathcal{T},\Sigma,\mathcal{A})}$ and checks whether $\mathcal{S}_{(\mathcal{T},\Sigma,\mathcal{A})}$ has solutions over $\mathbb{N}^*$. The two components together thus check whether $(\mathcal{T},\Sigma,\mathcal{A})$ is satisfiable. This is depicted in Fig. 4.1. It is worth noting that our constructed $\mathsf{Datalog}^{\neg}$ program $\mathcal{P}^{\mathcal{T},\Sigma}_{sat}$ is modular in the sense that $\mathcal{P}^{\mathcal{T},\Sigma}_{sys}$ computes the extensions of the shared predicates, which are then only used as an input (i.e., EDB predicates) to $\mathcal{P}^{\mathcal{T},\Sigma}_{sol}$. Thus, in view of Proposition 2.4.12 we can compute the answer sets of $\mathcal{P}^{\mathcal{T},\Sigma}_{sol}$ by first computing the answer sets to $\mathcal{P}^{\mathcal{T},\Sigma}_{sys}$ and then using them as inputs to $\mathcal{P}^{\mathcal{T},\Sigma}_{sys}$. Finally, we remark that the program $\mathcal{P}^{\mathcal{T},\Sigma}_{sol}$ depends on Σ and $\mathcal{T}$ only in terms of the arities of the shared predicates and can otherwise be used to solve arbitrary enriched systems of linear inequalities as long as they can be represented using the provided signature.

4.1.1 Generating Linear Inequalities

Consider an $\mathcal{ALCHOIQ}$ TBox $\mathcal{T}$ and a set of closed predicates Σ. For ease of presentation, we will assume that every predicate from Σ also occurs in $\mathcal{T}$. Indeed, if there is some predicate $p \in \Sigma$ that does not occur in $\mathcal{T}$, we can simply add $p \sqsubseteq p$ to obtain a TBox equivalent to $\mathcal{T}$ s.t. p occurs in it. We next show how to obtain in polynomial time the program $\mathcal{P}^{\mathcal{T},\Sigma}_{sys}$ that has the following properties:

- for every ABox $\mathcal{A}$ over the signature of $\mathcal{T}$, the program $(\mathcal{P}_{sys}^{\mathcal{T},\Sigma}, \hat{\mathcal{A}})$ has a stable model if and only if the KB $(\mathcal{T}, \Sigma, \mathcal{A})$ respects role inclusions, and
- the stable models of $(\mathcal{P}_{sys}^{\mathcal{T},\Sigma}, \hat{\mathcal{A}})$ correspond to $Rel(\mathcal{S}_{(\mathcal{T},\Sigma,\mathcal{A})})$, differing only in terms of which IDs they use for the variables, inequalities, and implications in $\mathcal{S}_{(\mathcal{T},\Sigma,\mathcal{A})}$.

We now sketch the construction of $\mathcal{P}_{sys}^{\mathcal{T},\Sigma}$. This program is modular and consists of four distinct components that, given an input ABox $\mathcal{A}$, do the following:

1. Compute all possible candidate tiles for $(\mathcal{T}, \Sigma, \mathcal{A})$.
2. Eliminate the candidates that do not satisfy the conditions in Definition 3.2.5 leaving behind only proper tiles – these are the variables of $\mathcal{S}_{(\mathcal{T},\Sigma,\mathcal{A})}$.
3. Compute the inequalities and implications of $\mathcal{S}_{\mathcal{K}}$.
4. Check whether $(\mathcal{T}, \Sigma, \mathcal{A})$ respects role inclusions.

We note that all constants used by $\mathcal{P}_{sys}^{\mathcal{T},\Sigma}$ (and also $\mathcal{P}_{sol}^{\mathcal{T},\Sigma}$) are introduced right at the beginning, and each of the components uses the relations computed by the previous components only as EDB predicates. Thus, in view of Proposition 2.4.12, the answer sets of $\mathcal{P}_{sys}^{\mathcal{T},\Sigma}$ can be computed by computing the answer sets of the first component extended with the facts from the input ABox, then feeding these answer sets, one by one, as facts into the second component, and repeating this procedure until all components have been considered. Relevant dependencies among the predicates used to define $\mathcal{P}_{sys}^{\mathcal{T},\Sigma}$ are depicted in Figure 4.2 and the complete overview of the signature is listed in Table 4.1. For ease of presentation, we also assume the following convention: all symbols occurring in the predicates within the rules of our program represent variables (or variable vectors) unless they are $0, 1, *$, or it is specifically stated otherwise. Furthermore, we use the notation $\vec{0}$ to refer to the vector consisting of only zeroes. We note that the length of $\vec{0}$ might vary from rule to rule. We are now ready to begin with our construction.

Let $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$, for some input ABox $\mathcal{A}$. We first present the rules that compute the basic building blocks for computing our relational representation of relations $\mathcal{S}_{\mathcal{K}}$, namely the relations Bin, Int, Adom and Cst. We begin with the relation Bin, that simply stores constants 0 and 1 and is computed using the following two facts:

$$\mathsf{Bin}(0). \qquad \mathsf{Bin}(1).$$

Building on Bin, the relation Int stores all relevant integers for computing the system $\mathcal{S}_{\mathcal{K}}$. From T1 and T2 in Definition 3.2.5, we can conclude that for any tile (T, ρ), if $(R, T', k) \in \rho$, then $1 \leq k \leq c_{\mathcal{T}}$. Further, by closely inspecting Definition 3.2.9, we can see that if an integer k occurs in $\mathcal{S}_{\mathcal{K}}$ as a constant, then $0 \leq k \leq \max(c_{\mathcal{T}} \cdot m_{\mathcal{T}}, 1)$. This means that we can encode all relevant integers as strings of length $\log(\max(c_{\mathcal{T}} \cdot m_{\mathcal{T}}, 1))$

using the usual binary encoding. These strings are stored in the relation Int that is computed using the following rule:

$$\mathsf{Int}(x_1, \ldots, x_{\log(\max(c_\mathcal{T} \cdot m_\mathcal{T}, 1))}) \leftarrow \mathsf{Bin}(x_1), \ldots, \mathsf{Bin}(x_{\log(\max(c_\mathcal{T} \cdot m_\mathcal{T}, 1))}).$$

Further, we need access to the constants that occur in $\mathcal{K}$, i.e., the active domain of $\mathcal{K}$. For this, we define the relation Adom and add the following facts:

$$\mathsf{Adom}(c),$$

for all constants c occurring in $\mathcal{T}$. Note that this adds polynomially many facts to $\mathcal{P}^{\mathcal{T},\Sigma}_{sys}$ in the size of $\mathcal{T}$. To gain access to the constants that occur in the ABox A while keeping the size of $\mathcal{P}^{\mathcal{T},\Sigma}_{sys}$ independent of $\mathcal{A}$, we add the following generic rules that collect all the constants occurring in $\mathcal{A}$:

$$\begin{array}{ll} \mathsf{Adom}(x) \leftarrow A(x), & \mathsf{Adom}(x) \leftarrow \overline{A}(x), \\ \mathsf{Adom}(x) \leftarrow r(x,y), & \mathsf{Adom}(x) \leftarrow r(y,x), \\ \mathsf{Adom}(x) \leftarrow \overline{r}(x,y), & \mathsf{Adom}(x) \leftarrow \overline{r}(y,x), \end{array}$$

for each concept name $A \in \mathsf{N_C}(\mathcal{T})$ and role name $r \in \mathsf{N_R}(\mathcal{T})$.

We also define another relation Adom^* that contains all of the constants from Adom as well as a special constant $*$ whose significance will be explained later. We add:

$$\mathsf{Adom}^*(*), \qquad \mathsf{Adom}^*(x) \leftarrow \mathsf{Adom}(x).$$

Finally, we use the relation Cst to store all the constants from Bin and Adom:

$$\mathsf{Cst}(x) \leftarrow \mathsf{Bin}(x), \qquad \mathsf{Cst}(x) \leftarrow \mathsf{Adom}^*(x).$$

Computing candidate tiles We are now ready to define the rules that compute the relation CandT, storing a relational representation of all candidate tiles for $\mathcal{K}$. Similarly to a proper tile, a candidate tile for $\mathcal{K}$ consists of a type T for $\mathcal{K}$ and a set ρ of triples (R, T', k), where T' is a type for $\mathcal{K}$, R is a role type for $\mathcal{K}$ and $1 \leq k \leq c_\mathcal{T}$. However, the difference between tiles and candidate tiles for $\mathcal{K}$ is that the latter need not satisfy all the conditions in Def. 3.2.5. As types and role types are integral components of tiles, we first define the rules that compute and store all types and role types for $\mathcal{K}$ using relations Type and RType, respectively.

Let us first focus on role types, as they are easier to explain. Recall that a role type is simply a subset of $\mathsf{N}^+_\mathsf{R}(\mathcal{T})$, which means that there are $2^{k_\mathcal{T}}$ different role types for $\mathcal{K}$, where $k_\mathcal{T}$ is the number of roles in $\mathsf{N}^+_\mathsf{R}(\mathcal{T})$ (see Table 3.1). Hence, we need exponentially many different constants to represent every role type, which is a problem, as we want to obtain a polynomially-sized program. To overcome this, we resort to a common trick and

encode role types as strings of length $k_\mathcal{T}$ over constants 0 and 1, where each binary string represents a different role type. To this end, we fix an (arbitrary) enumeration $r_1, \dots, r_{k_\mathcal{T}}$ of the roles in $\mathsf{N}_\mathsf{R}^+(\mathcal{T})$ and associate to every role type R a binary string of length $k_\mathcal{T}$ that acts as an identifier for R and indicates which roles occur in R. More precisely, let l be a binary string of length $k_\mathcal{T}$ and let $l[i]$, $1 \leq i \leq k_\mathcal{T}$ denote the constant at the i-th position in l. Then, l is the identifier for the role type R for which the following holds: $r_i \in R$ if and only if $l[i] = 1$. For example, the string $11100\dots0$ uniquely identifies the role type $\{r_1, r_2, r_3\}$. Strings that are used as identifiers for role types are stored in the relation RType and are computed using the following simple rule:

$$\mathsf{RType}(x_1, \dots, x_{k_\mathcal{T}}) \leftarrow \mathsf{Bin}(x_1), \dots, \mathsf{Bin}(x_{k_\mathcal{T}}).$$

Next, we show how to encode types. Recall that a type T for $\mathcal{K}$ is a subset of $\mathsf{N}_\mathsf{C}^+(\mathcal{K})$, with the caveat that T contains at most one nominal. Naturally, one might want to employ the same approach as the one used for encoding role types, i.e., fix an enumeration of all concept names and nominals in $\mathsf{N}_\mathsf{C}^+(\mathcal{K})$ and encode types as binary strings of length $|\mathsf{N}_\mathsf{C}^+(\mathcal{K})|$. However, as the number of nominals in $\mathsf{N}_\mathsf{C}^+(\mathcal{K})$ depends on the ABox $\mathcal{A}$, this would make our translation data-dependent, as the arity of the Type relation would depend on the number of constants in $\mathcal{A}$. Relying on the fact that there can be at most one nominal in a type, we overcome this issue as follows. Similarly to before, we fix an enumeration $B_1, \dots, B_{n_\mathcal{T}}$ of the concept names in $\mathsf{N}_\mathsf{C}(\mathcal{T})$. We assign to every type T a string of length $n_\mathcal{T} + 1$, where the first $n_\mathcal{T}$ positions are either 0 or 1, indicating which concept names occur in T. The last position in the string indicates which nominal (if any) occurs in T and is filled by either a constant from $\mathcal{K}$, denoting a specific nominal, or a special constant $*$, denoting the lack of nominals in T. More precisely, each type T is encoded by the string l of length $n_\mathcal{T} + 1$ such that, for $1 \leq i \leq n_\mathcal{T}$, $l[i] = 1$ if $B_i \in T$, otherwise $l[i] = 0$, and if there is a nominal $\{a\} \in T$, then $l[n_\mathcal{T} + 1] = a$, otherwise $l[n_\mathcal{T} + 1] = *$. For example, the string $11100\dots0a$ represents the type $\{B_1, B_2, B_3, \{a\}\}$, whereas the string $11100\dots0*$ represents the type $\{B_1, B_2, B_3\}$. These strings are stored in the relation Type and are computed using the rule:

$$\mathsf{Type}(x_1, \dots, x_{n_\mathcal{T}+1}) \leftarrow \mathsf{Bin}(x_1), \dots, \mathsf{Bin}(x_k), \mathsf{Adom}^*(x_{n_\mathcal{T}+1}).$$

Recall once again that a candidate tile consists of a type T and a set of triples ρ of the form (R, T', k), where R is a role type, T' is a type and k is an integer with $1 \leq k \leq c_\mathcal{T} \cdot m_\mathcal{T}$. We introduce a new relation Triple that stores all such triples and is computed by the following rule:

$$\mathsf{Triple}(\vec{R}, \vec{T}, \vec{k}) \leftarrow \mathsf{Type}(\vec{T}), \mathsf{RType}(\vec{R}), \mathsf{Int}(\vec{k}).$$

We can now make use of the relations Type and Triple to compute the relation CandT that stores all candidate tiles for $\mathcal{K}$. Once again, our goal is to encode candidate tiles as strings over 0, 1, and the constants from $\mathcal{K}$ of fixed length that is polynomial in the size of $\mathcal{T}$ and Σ and constant in the size of $\mathcal{A}$.

The first thing that we need to decide is exactly how long these strings should be. Conditions T2 and T4 in Definition 3.2.5 together tell us that, for each (candidate) tile $\tau = (T, \rho)$ for $\mathcal{K}$, $|\rho| \leq c_{\mathcal{T}} \cdot m_{\mathcal{T}}$. However, it should be noted that $c_{\mathcal{T}} \cdot m_{\mathcal{T}}$ is only an upper bound, meaning that τ is not required to have exactly $c_{\mathcal{T}} \cdot m_{\mathcal{T}}$ elements in ρ. This makes encoding tiles via strings of fixed length a little tricky. To overcome this issue, if $|\rho| < c_{\mathcal{T}} \cdot m_{\mathcal{T}}$ for some candidate tile (T, ρ), during the encoding we pad ρ to the desired size by allowing duplicates in ρ. Thus, we can encode (candidate) tiles as strings of length $n_{\mathcal{T}} + 1 + c_{\mathcal{T}} \cdot m_{\mathcal{T}} \cdot (n_{\mathcal{T}} + 1 + k_{\mathcal{T}} + \log(\max(c_{\mathcal{T}} \cdot m_{\mathcal{T}}, 1))$. From now on, unless stated otherwise, let $n = c_{\mathcal{T}} \cdot m_{\mathcal{T}}$.

Now, a natural way of computing the relation CandT would be by adding the following in $\mathcal{P}_{sys}^{\mathcal{T},\Sigma}$:

$$\mathsf{CandT}(\vec{T}, \vec{R}_1, \vec{T}_1, \vec{k}_1, \ldots, \vec{R}_n, \vec{T}_n, \vec{k}_n) \leftarrow \mathsf{Type}(\vec{T}), \mathsf{Triple}(\vec{R}_1, \vec{T}_1, \vec{k}_1), \ldots, \mathsf{Triple}(\vec{R}_n, \vec{T}_n, \vec{k}_n).$$

However this approach is too naive for the following reason. Recall that ρ is a set, which means that duplicates and the order in which the triples occur in ρ are ignored. Therefore, if we compute CandT using the rule given above, we will inevitably have multiple different tuples in CandT encoding the same tile. We deal with this as follows. First, we guess a linear order over the constants used to encode the triples (i.e., the constants stored in Cst) using a binary predicate LEQ, where 0 is the least constant with respect to LEQ. This is done using the following rules:

$$\begin{aligned}
&\mathsf{LEQ}(0, x) \leftarrow \mathsf{Cst}(x),\\
&\mathsf{LEQ}(x, x) \leftarrow \mathsf{Cst}(x),\\
&\mathsf{LEQ}(x, y) \leftarrow \mathsf{Cst}(x), \mathsf{Cst}(y), \mathit{not}\ \mathsf{LEQ}(y, x),\\
&\mathsf{LEQ}(x, z) \leftarrow \mathsf{LEQ}(x, y), \mathsf{LEQ}(y, z).
\end{aligned}$$

We use the standard approach to lift this linear order to strings of length $(n_{\mathcal{T}} + 1 + k_{\mathcal{T}} + \log(\max(n, 1)))$ (see e.g., [DEGV01]), using the $2(n_{\mathcal{T}} + 1 + k_{\mathcal{T}} + \log(\max(n, 1)))$-ary relation $\mathsf{LEQ}^{n_{\mathcal{T}}+1+k_{\mathcal{T}}+\log(\max(n,1))}$ which allows us to compare strings stored in relation Triple.

We assume the following convention for encoding (candidate) tiles. The only tuples that are valid encodings of candidate tiles are tuples of the form $(\vec{T}, \vec{R}_1, \vec{T}_1, \vec{k}_1, \ldots, \vec{R}_n, \vec{T}_n, \vec{k}_n)$, where $\vec{T}, \vec{T}_1, \ldots, \vec{T}_n$ are in Type, $(\vec{R}_i, \vec{T}_i, \vec{k}_i) \in \mathsf{Triple}$, for $1 \leq i \leq n$, and the following holds:

(i) $(\vec{R}_i, \vec{T}_i, \vec{k}_i, \vec{R}_j, \vec{T}_j, \vec{k}_j)$ is in $\mathsf{LEQ}^{n_{\mathcal{T}}+1+k_{\mathcal{T}}+\log(\max(n,1))}$, for all $1 \leq i < j \leq n$, and

(ii) if $(\vec{R}_i, \vec{T}_i, \vec{k}_i) = (\vec{R}_j, \vec{T}_j, \vec{k}_j)$, then $\vec{R}_i = (0, \ldots, 0)$, $\vec{T}_i = (0, \ldots, 0, *)$ and $\vec{k}_i = (0, \ldots, 0, 1)$, for all $1 \leq i, j \leq n$.

Intuitively, this condition assumes that given a tile (T, ρ), ρ is encoded by the string that lists the triples of ρ in the ascending order with respect to the guessed linear order, and, in case ρ contains less than n triples, pads it at the beginning with empty triples.

Tuples that have the correct form but violate (i) or (ii) are stored in a separate relation $\mathsf{InvCandT}$ that is computed as follows:

$$\mathsf{InvCandT}(\vec{T},\vec{R}_1,\vec{T}_1,\vec{k}_1,\ldots,\vec{R}_n,\vec{T}_n,\vec{k}_n) \leftarrow \mathsf{Type}(\vec{T}),\mathsf{Triple}(\vec{R}_1,\vec{T}_1,\vec{k}_1),\ldots,\mathsf{Triple}(\vec{R}_n,\vec{T}_n,\vec{k}_n),$$
$$\mathit{not}\ \mathsf{LEQ}^{n_\tau+1+k_\tau+\log(\max(n,1))}(\vec{R}_i,\vec{T}_i,\vec{k}_i,\vec{R}_j,\vec{T}_j,\vec{k}_j),$$

$$\mathsf{InvCandT}(\vec{T},\vec{R}_1,\vec{T}_1,\vec{k}_1,\ldots,\vec{R}_n,\vec{T}_n,\vec{k}_n) \leftarrow \mathsf{Type}(\vec{T}),\mathsf{Triple}(\vec{R}_1,\vec{T}_1,\vec{k}_1),\ldots,\mathsf{Triple}(\vec{R}_n,\vec{T}_n,\vec{k}_n),$$
$$(\vec{R}_i,\vec{T}_i,\vec{k}_i)=(\vec{R}_j,\vec{T}_j,\vec{k}_j),\vec{R}_i\neq(0,\ldots,0),$$

$$\mathsf{InvCandT}(\vec{T},\vec{R}_1,\vec{T}_1,\vec{k}_1,\ldots,\vec{R}_n,\vec{T}_n,\vec{k}_n) \leftarrow \mathsf{Type}(\vec{T}),\mathsf{Triple}(\vec{R}_1,\vec{T}_1,\vec{k}_1),\ldots,\mathsf{Triple}(\vec{R}_n,\vec{T}_n,\vec{k}_n),$$
$$(\vec{R}_i,\vec{T}_i,\vec{k}_i)=(\vec{R}_j,\vec{T}_j,\vec{k}_j),\vec{T}_i\neq(0,\ldots,0,*),$$

$$\mathsf{InvCandT}(\vec{T},\vec{R}_1,\vec{T}_1,\vec{k}_1,\ldots,\vec{R}_n,\vec{T}_n,\vec{k}_n) \leftarrow \mathsf{Type}(\vec{T}),\mathsf{Triple}(\vec{R}_1,\vec{T}_1,\vec{k}_1),\ldots,\mathsf{Triple}(\vec{R}_n,\vec{T}_n,\vec{k}_n),$$
$$(\vec{R}_i,\vec{T}_i,\vec{k}_i)=(\vec{R}_j,\vec{T}_j,\vec{k}_j),\vec{k}_i\neq(0,\ldots,0,1),$$

for all for $1\leq i,j\leq n$. Note that the first of the rules above stores in $\mathsf{InvCandT}$ the potential encodings for a tile that violates the condition (i), while the other three rules together store the ones that violate the condition (ii).

Finally, we can compute the relation CandT with the help of the following rule:

$$\mathsf{CandT}(\vec{T},\vec{R}_1,\vec{T}_1,\vec{k}_1,\ldots,\vec{R}_n,\vec{T}_n,\vec{k}_n) \leftarrow \mathsf{Type}(\vec{T}),\mathsf{Triple}(\vec{R}_1,\vec{T}_1,\vec{k}_1),\ldots,\mathsf{Triple}(\vec{T}_n,\vec{R}_n,\vec{k}_n),$$
$$\mathit{not}\ \mathsf{InvCandT}(\vec{T},\vec{R}_1,\vec{T}_1,\vec{k}_1,\ldots,\vec{R}_n,\vec{T}_n,\vec{k}_n).$$

This concludes the construction of the first component of $\mathcal{P}^{\mathcal{T},\Sigma}_{sys}$. It is easy to verify that the answer sets of this component in conjunction with the ABox, contain exactly one tuple in CandT per candidate tile τ of $\mathcal{K}$ that encodes τ as explained above.

Eliminating bad candidate tiles The second component, takes the computed relation CandT and eliminates the ones that do not encode a valid tile. To this end, we introduce a new relation $\mathsf{BadTile}$ storing bad candidate tiles, i.e., those that violate one of the conditions in Definition 3.2.5. From now on, unless stated otherwise, we assume that $\vec{T},\vec{T}'$ and $\vec{T}_i$, are variable vectors of length $n_\tau+1$, $\vec{R}$ and $\vec{R}_i$ are variable vectors of length $k_\mathcal{T}$, $\vec{k}$ and $\vec{k}_i$ are variable vectors of length $\log(\max(n,1))$, and $\vec{t},\vec{t}'$, and $\vec{t}_i$ are variable vectors of length $n_\mathcal{T}+1+n\cdot(n_\mathcal{T}+1+k_\mathcal{T}+\log(\max(n,1)))$, for all $i\geq 1$.

We begin by introducing a couple of auxiliary relations. Recall that we assume enumerations $B_1,\ldots,B_{n_\mathcal{T}}$ concept names in $\mathsf{N_C}(\mathcal{T})$ and $r_1,\ldots,r_{k_\mathcal{T}}$ of roles in $\mathsf{N^+_R}(\mathcal{T})$. For every concept name B_i, $1\leq i\leq n_\mathcal{T}$, we define an auxiliary $n_\mathcal{T}+1$-ary relation In_{B_i} that stores all types containing B_i occurs, i.e., whose i-th position is set to 1. The relation is computed by the following rule:

$$\mathsf{In}_{B_i}(x_1,\ldots,x_{n_\mathcal{T}},x_{n_\mathcal{T}+1}) \leftarrow \mathsf{Type}(x_1,\ldots,x_{n_\mathcal{T}},x_{n_\mathcal{T}+1}),x_i=1.$$

Similarly, for each role r_j, $1\leq j\leq k_\mathcal{T}$, we define an $k_\mathcal{T}$-ary relation In_{r_j} that stores all role types containing r_j, i.e., whose j-th position is set to 1. This relation is computed by

$$\mathsf{In}_{r_j}(x_1,\ldots,x_{k_\mathcal{T}}) \leftarrow \mathsf{RType}(x_1,\ldots,x_{k_\mathcal{T}}),x_{k_\mathcal{T}}=1.$$

We now go through the conditions of Definition 3.2.5 and add the corresponding rules that "invalidate" candidate tiles that do not satisfy them, i.e., that store the tuple representing this candidate tile in the relation $\mathsf{BadTile}$:

T1. In order to properly capture T1, we need to find a way to talk about predecessors and successors of integers stored in the relation Int. To this end, we compute the relation $\mathsf{LEQ}^{\log(\max(n,1)}$ that is obtained by lifting the linear order LEQ to strings of length $\log(\max(n,1))$. We then use this relation to extract the successor relation on the integers in Int as follows:

$$\begin{aligned}\overline{\mathsf{Succ}_{\mathsf{int}}}(\vec{x},\vec{y}) \leftarrow\ & \mathsf{Int}(\vec{x}),\mathsf{Int}(\vec{z}),\mathsf{Int}(\vec{y}),\mathsf{LEQ}^{\log(\max(c_{\mathcal{T}}\cdot m_{\mathcal{T}},1))}(\vec{x},\vec{z}),\\ & \mathsf{LEQ}^{\log(\max(n,1))}(\vec{z},\vec{y}),\vec{x}\neq\vec{y},\vec{x}\neq\vec{z},\vec{y}\neq\vec{z},\\ \mathsf{Succ}_{\mathsf{int}}(\vec{x},\vec{y}) \leftarrow\ & \mathsf{Int}(\vec{x}),\mathsf{Int}(\vec{y}),\mathsf{LEQ}^{\log(\max(n,1))}(\vec{x},\vec{y}),\vec{x}\neq\vec{y},\mathit{not}\ \overline{\mathsf{Succ}_{\mathsf{int}}}(\vec{x},\vec{y}).\end{aligned}$$

The first rule stores all pairs $(\vec{x},\vec{y})$ in $\overline{\mathsf{Succ}_{\mathsf{int}}}$ for which the integer encoded by $\vec{y}$ is *not* the successor of the integer encoded by $\vec{x}$, i.e., if there is some integer encoded by $\vec{z}$, different from the ones encoded by $\vec{x}$ and $\vec{y}$, which is greater than $\vec{x}$ and smaller than $\vec{y}$. The second rule then stores pairs $(\vec{x},\vec{y})$ in $\mathsf{Succ}_{\mathsf{int}}$, if $\vec{y}$ encodes an integer that is greater than the one encoded by $\vec{x}$ and for which there is no other integer that is in between them. Thus, $\mathsf{Succ}_{\mathsf{int}}$ encodes the successor relation over available integers.

Next, for all $1\leq i\leq n$, we define an auxiliary relation OK_i^{T1}, for all $1\leq i\leq n$, that stores candidate tiles (T,ρ) whose i-th triple in ρ satisfies T1 in Definition 3.2.5. This is done using two rules – the first one "accepts" the i-th triple $(R_i,T_i,k_i)\in\rho$ if $k_i=1$, as it trivially satisfies T1, and the second one "accepts" (R_i,T_i,k_i) if we can find some other triple $(R_j,T_j,k_j)\in\rho$ such that k_j is the predecessor of k_i:

$$\begin{aligned}\mathsf{OK}_i^{T1}(\vec{T},\vec{R}_1,\vec{T}_1,\vec{k}_1,\ldots,\vec{R}_n,\vec{T}_n,\vec{k}_n) \leftarrow\ & \mathsf{CandT}(\vec{T},\vec{R}_1,\vec{T}_1,\vec{k}_1,\ldots,\vec{R}_n,\vec{T}_n,\vec{k}_n),\\ & \vec{k}_i=(0,\ldots,0,1),\\ \mathsf{OK}_i^{T1}(\vec{T},\vec{R}_1,\vec{T}_1,\vec{k}_1,\ldots,\vec{R}_n,\vec{T}_n,\vec{k}_n) \leftarrow\ & \mathsf{CandT}(\vec{T},\vec{R}_1,\vec{T}_1,\vec{k}_1,\ldots,\vec{R}_n,\vec{T}_n,\vec{k}_n),\\ & \vec{R}_j=\vec{R}_i,\vec{T}_j=\vec{T}_i,\mathsf{Succ}_{\mathsf{int}}(\vec{k}_j,\vec{k}_i),\\ & \text{for } 1\leq j\leq n, j\neq i.\end{aligned}$$

Finally, if there is a triple in a candidate tile that does not satisfy T1, then this candidate is a "bad tile". Thus, we add the following rule, for all $1\leq i\leq n$:

$$\mathsf{BadTile}(\vec{t}) \leftarrow \mathsf{CandT}(\vec{t}),\mathit{not}\ \mathsf{OK}_i^{T1}(\vec{t}).$$

T2. For all $1\leq i\leq n$, we define an auxiliary relation OK_i^{T2} that collects all candidate tiles (T,ρ) whose i-th triple in ρ fulfills T2 in Definition 3.2.5. The relation OK_i^{T2}

is then computed using the following rules:

$$\begin{aligned}\mathsf{OK}_i^{T2}(\vec{T},\vec{R}_1,\vec{T}_1,\vec{k}_1,\ldots,\vec{R}_n,\vec{T}_n,\vec{k}_n) \leftarrow\ & \mathsf{CandT}(\vec{T},\vec{R}_1,\vec{T}_1,\vec{k}_1,\ldots,\vec{R}_n,\vec{T}_n,\vec{k}_n),\\ & \vec{R}_i=(0,\ldots,0),\vec{T}_i=(0,\ldots,0,*),\\ \mathsf{OK}_i^{T2}(\vec{T},\vec{R}_1,\vec{T}_1,\vec{k}_1,\ldots,\vec{R}_n,\vec{T}_n,\vec{k}_n) \leftarrow\ & \mathsf{CandT}(\vec{T},\vec{R}_1,\vec{T}_1,\vec{k}_1,\ldots,\vec{R}_n,\vec{T}_n,\vec{k}_n),\\ & \mathsf{In}_{B_{i_1}}(\vec{T}),\mathsf{In}_{B_{i_2}}(\vec{T}_i),\mathsf{In}_{r_h}(\vec{R}_i),\end{aligned}$$

for every axiom $B_{i_1} \sqsubseteq= m\ r_h.B_{i_2} \in \mathcal{T}$. Let (R_i,T_i,k_i) denote the i-th triple in ρ. The first rule "accepts" (R_i,T_i,k_i) if both R_i and T_i are empty. Recall that such triples were used for padding ρ to the desired size and they do not represent any real connections. The second rule actually checks whether there is an axiom $B_{i_1} \sqsubseteq= m\ r_h.B_{i_2} \in \mathcal{T}$ such that $B_{i_1} \in T$, $B_{i_2} \in T_i$ and $m \in R_i$, and if this is the case, it "accepts" (R_i,T_i,k_i).

As before, we eliminate all tiles for which this does not hold:

$$\mathsf{BadTile}(\vec{t}) \leftarrow\ \mathsf{CandT}(\vec{t}), \mathit{not}\ \mathsf{OK}_i^{T2}(\vec{t}),\ \text{for } 1 \le i \le n.$$

T3. The rule for T3 is rather straightforward. For every $B_{i_1} \sqcap \cdots \sqcap B_{i_{m-1}} \sqsubseteq B_{i_m} \sqcup \cdots \sqcup B_{i_t} \in \mathcal{T}$ we add:

$$\begin{aligned}\mathsf{BadTile}(\vec{T},\vec{r}) \leftarrow\ & \mathsf{CandT}(\vec{T},\vec{r}),\mathsf{In}_{B_{i_1}}(\vec{T}),\ldots,\mathsf{In}_{B_{i_{m-1}}}(\vec{T}),\\ & \mathit{not}\ \mathsf{In}_{B_{i_m}}(\vec{T}),\ldots,\mathit{not}\ \mathsf{In}_{B_{i_t}}(\vec{T}).\end{aligned}$$

T4. Eliminating the candidate tiles that violate T4 is done in mulitple steps. First, given an axiom $B_{i_1} \sqsubseteq= m\ r_h.B_{i_2} \in \mathcal{T}$ and a candidate tile (T,ρ) with $B_{i_1} \in T$, we need a way of determining how many triples (R,T',k) there are in ρ, for which $B_{i_2} \in T'$ and $r_h \in R$. To this end, for every $1 \le j \le n$ and every axiom $\alpha = B_{i_1} \sqsubseteq= m\ r_h.B_{i_2} \in \mathcal{T}$, we introduce a new auxiliary relation $\mathsf{Upto}_{j,B_{i_1}\sqsubseteq=m\ r_h.B_{i_2}}$ whose role is the following. Assume that $\vec{t} \in \mathsf{CandT}$ represents the candidate tile (T,ρ) and let $\vec{k} \in \mathsf{Int}$ be a binary representation of some integer $k \le n$. The relation $\mathsf{Upto}_{j,B_{i_1}\sqsubseteq=m\ r_h.B_{i_2}}$ stores $(\vec{t},\vec{k})$ if and only if $B_{i_1} \in T$ and among the first j triples of ρ, there are exactly k triples (R,T',k) such that $B_{i_2} \in T'$ and $r_h \in R$. This relation is computed by adding the following rules, for all $j = 1,\ldots,n$:

$$\mathsf{Upto}_{0,\alpha}(\vec{T},\vec{r},\vec{0}) \leftarrow \mathsf{CandT}(\vec{T},\vec{r}),\mathsf{In}_{B_{i_1}}(\vec{T}),\mathsf{Int}(\vec{0}),$$

$$\mathsf{Upto}_{j,\alpha}(\vec{T},\vec{r},\vec{k}) \leftarrow \mathsf{Upto}_{j-1,\alpha}(\vec{T},\vec{r},\vec{k}'),\mathsf{Succ}_{\mathsf{int}}(\vec{k}',\vec{k}),\mathsf{In}_{B_{i_2}}(\vec{T}_j),\mathsf{In}_{r_h}(\vec{R}_j),$$

$$\mathsf{Upto}_{j,\alpha}(\vec{T},\vec{r},\vec{k}) \leftarrow \mathsf{Upto}_{j-1,\alpha}(\vec{T},\vec{r},\vec{k}),\mathit{not}\ \mathsf{In}_{B_{i_2}}(\vec{T}_j),$$

$$\mathsf{Upto}_{j,\alpha}(\vec{T},\vec{r},\vec{k}) \leftarrow \mathsf{Upto}_{j-1,\alpha}(\vec{T},\vec{r},\vec{k}),\mathit{not}\ \mathsf{In}_{r_h}(\vec{R}_j),$$

where $\vec{r} = \vec{R}_1, \vec{T}_1, \vec{k}_1, \ldots, \vec{R}_n, \vec{T}_n, \vec{k}_n$. The first of the rules given above initializes the sum to 0, if B_{i_1} is in T. The rest of the rules iterate through ρ and do the following. The second rule increases the previously-computed sum stored in $\mathsf{Upto}_{j-1,\alpha}$ by one, if in the j-th triple $(R_j, T_j, k_j) \in \rho$, $r_h \in R$ and $B_{i_2} \in T$. If this is not the case, the previous result is simply copied using the third and the fourth rule.

We can now make use of these auxiliary relations to compute candidate tiles that violate T4. To this end, for each axiom $B_{i_1} \sqsubseteq= m\ r_h.B_{i_2} \in \mathcal{T}$, we add the following rule

$$\mathsf{BadTile}(\vec{t}) \leftarrow \mathsf{CandT}(\vec{t}), \mathit{not}\ \mathsf{Upto}_{n,B_{i_1} \sqsubseteq= m\ r_h.B_{i_2}}(\vec{t}, \vec{b}),$$

where $\vec{b}$ is the vector of constants stored in Int corresponding to the binary representation of m.

T5. For each axiom $B_{i_1} \sqsubseteq \forall r_h.B_{i_2} \in \mathcal{T}$, assume $r_{h'} = r_h^-$. We add the following rules:

$$\begin{aligned}\mathsf{BadTile}(\vec{T}, \vec{R}_1, \vec{T}_1, \vec{k}_1, \ldots, \vec{R}_n, \vec{T}_n, \vec{k}_n) \leftarrow\ & \mathsf{CandT}(\vec{T}, \vec{R}_1, \vec{T}_1, \vec{k}_1, \ldots, \vec{R}_n, \vec{T}_n, \vec{k}_n),\\ & \mathsf{In}_{B_{i_1}}(\vec{T}), \mathsf{In}_{r_h}(\vec{R}_j), \mathit{not}\ \mathsf{In}_{B_{i_2}}(\vec{T}_j),\\ \mathsf{BadTile}(\vec{T}, \vec{R}_1, \vec{T}_1, \vec{k}_1, \ldots, \vec{R}_n, \vec{T}_n, \vec{k}_n) \leftarrow\ & \mathsf{CandT}(\vec{T}, \vec{R}_1, \vec{T}_1, \vec{k}_1, \ldots, \vec{R}_n, \vec{T}_n, \vec{k}_n),\\ & \mathsf{In}_{B_{i_1}}(\vec{T}_j), \mathsf{In}_{r_{h'}}(\vec{R}_j), \mathit{not}\ \mathsf{In}_{B_{i_2}}(\vec{T}),\end{aligned}$$

for all $1 \leq j \leq n$. This makes sure to eliminate the tiles that do not satisfy T5a and T5b.

To satisfy the T5c we do the following. For $1 \leq j \leq n$ and each $r_h \sqsubseteq r_g \in \mathcal{T}$, we add the following rule:

$$\begin{aligned}\mathsf{BadTile}(\vec{T}, \vec{R}_1, \vec{T}_1, \vec{k}_1, \ldots, \vec{R}_n, \vec{T}_n, \vec{k}_n) \leftarrow\ & \mathsf{CandT}(\vec{T}, \vec{R}_1, \vec{T}_1, \vec{k}_1, \ldots, \vec{R}_n, \vec{T}_n, \vec{k}_n),\\ & \mathsf{In}_{r_h}(\vec{R}_j), \mathit{not}\ \mathsf{In}_{r_g}(\vec{R}_j).\end{aligned}$$

The rules for T5a-T5c are rather simple – they are direct translations of the conditions in Definition 3.2.5.

To deal with T5d, we first need a way to represent invertible triples (see Definition 3.2.2). To this end, we introduce a new relation Invrt that stores such triples and is computed as follows. For each pair of axioms $B_{i_1} \sqsubseteq= m_1 r_h.B_{i_2}$ and $B_{j_1} \sqsubseteq= m_2 r_g.B_{j_2}$ occurring in $\mathcal{T}$, we add the following rule:

$$\begin{aligned}\mathsf{Invrt}(\vec{T}_1, \vec{R}, \vec{T}_2) \leftarrow\ & \mathsf{Type}(\vec{T}_1), \mathsf{RType}(\vec{R}), \mathsf{Type}(\vec{T}_2), \mathsf{In}_{B_{i_1}}(\vec{T}_1), \mathsf{In}_{B_{j_2}}(\vec{T}_1),\\ & \mathsf{In}_{B_{i_2}}(\vec{T}_2), \mathsf{In}_{B_{j_1}}(\vec{T}_2), \mathsf{In}_{r_h}(\vec{R}), \mathsf{In}_{r_{g'}}(\vec{R}),\end{aligned}$$

where $r_{g'}$ denotes the role r_g^-.

Then, adding the rules that "invalidate" candidate tiles that violate T5d is once again straightforward:

$$\mathsf{BadTile}(\vec{T}, \vec{R}_1, \vec{T}_1, \vec{k}_1, \ldots, \vec{R}_n, \vec{T}_n, \vec{k}_n) \leftarrow \mathsf{CandT}(\vec{T}, \vec{R}_1, \vec{T}_1, \vec{k}_1, \ldots, \vec{R}_n, \vec{T}_n, \vec{k}_n), \mathsf{Invrt}(\vec{T}, \vec{R}_i, \vec{T}_i), \vec{T} = \vec{T}_i,$$

$$\mathsf{BadTile}(\vec{T}, \vec{R}_1, \vec{T}_1, \vec{k}_1, \ldots, \vec{R}_n, \vec{T}_n, \vec{k}_n) \leftarrow \mathsf{CandT}(\vec{T}, \vec{R}_1, \vec{T}_1, \vec{k}_1, \ldots, \vec{R}_n, \vec{T}_n, \vec{k}_n), \mathsf{Invrt}(\vec{T}, \vec{R}_i, \vec{T}_i), \mathsf{Invrt}(\vec{T}, \vec{R}_j, \vec{T}_j), \vec{T}_i = \vec{T}_j,$$

for all $1 \leq i, j \leq n$, $i \neq j$.

Finally, we also translate the conditions T5e and T5f as follows. For each role name $r_h \in \mathsf{N_R}(\mathcal{T})$, and each $i = 1, \ldots, n$, we add:

$$\mathsf{BadTile}(\vec{T}, \vec{R}_1, \vec{T}_1, \vec{k}_1, \ldots, \vec{R}_n, \vec{T}_n, \vec{k}_n) \leftarrow \mathsf{CandT}(\vec{T}, \vec{R}_1, \vec{T}_1, \vec{k}_1, \ldots, \vec{R}_n, \vec{T}_n, \vec{k}_n), \overline{r_h}(x, x'), \mathsf{In}_{r_h}(\vec{y}_i), w_i = x',$$

$$\mathsf{BadTile}(\vec{T}, \vec{R}_1, \vec{T}_1, \vec{k}_1, \ldots, \vec{R}_n, \vec{T}_n, \vec{k}_n) \leftarrow \mathsf{CandT}(\vec{T}, \vec{R}_1, \vec{T}_1, \vec{k}_1, \ldots, \vec{R}_n, \vec{T}_n, \vec{k}_n), \overline{r_h}(x', x), \mathsf{In}_{r_{h'}}(\vec{R}_i), w_i = x',$$

where $r_{h'} = r_h^-$, $\vec{T} = x_1, \ldots, x_{n_\mathcal{T}}, w$, and $\vec{T}_i = x_{i_1}, \ldots, x_{i_{n_\mathcal{T}}}, w_i$.

The rules that invalidate the tiles that violate one of T6-T9 are once again straightforward. In what follows, we assume $\vec{T} = x_1, \ldots, x_{n_\mathcal{T}}, w$ and $\vec{T}_i = x_{i_1}, \ldots, x_{i_{n_\mathcal{T}}}, w_i$, for all $1 \leq i \leq n$.

T6. For each $B_i \in \mathsf{N_C}(\mathcal{T})$ we add:

$$\mathsf{BadTile}(\vec{T}, \vec{r}) \leftarrow \mathsf{CandT}(\vec{T}, \vec{r}), B_i(w), \mathit{not}\ \mathsf{In}_{B_i}(\vec{T}).$$

T7. For each $B_i \in \mathsf{N_C}(\mathcal{T})$ we add:

$$\mathsf{BadTile}(\vec{T}, \vec{r}) \leftarrow \mathsf{CandT}(\vec{T}, \vec{r}), \overline{B_i}(w), \mathsf{In}_{B_i}(\vec{T}).$$

T8. For each $B_i \in \Sigma \cap \mathsf{N_C}(\mathcal{T})$,

$$\mathsf{BadTile}(\vec{T}, \vec{r}) \leftarrow \mathsf{CandT}(\vec{T}, \vec{r}), \mathsf{In}_{B_i}(\vec{T}), \mathit{not}\ B_i(w).$$

T9. For each $r_h \in \Sigma \cap \mathsf{N_R}(\mathcal{T})$ and each $i = 1, \ldots, n$, we add:

$$\mathsf{BadTile}(\vec{T}, \vec{R}_1, \vec{T}_1, \vec{k}_1, \ldots, \vec{R}_n, \vec{T}_n, \vec{k}_n) \leftarrow \mathsf{CandT}(\vec{T}, \vec{R}_1, \vec{T}_1, \vec{k}_1, \ldots, \vec{R}_n, \vec{T}_n, \vec{k}_n), \mathsf{In}_{r_h}(\vec{R}_i), \mathit{not}\ r_h(w, w_i).$$

At this point, the relation $\mathsf{BadTile}$ contains all candidate tiles that violate one of the tile conditions. We now add the following rule that filters out bad candidate tiles and leaves behind only true tiles for $\mathcal{K}$:

$$\mathsf{Tile}(\vec{t}) \leftarrow \mathsf{CandT}(\vec{t}), \mathit{not}\ \mathsf{BadTile}(\vec{t}).$$

Recall that the answer sets of the first component enriched with the facts from the ABox $\mathcal{A}$ contain precisely one tuple $\vec{t}$ in CandT, for each candidate tile τ encoded by $\vec{t}$. It is then not difficult to see that every answer set of the program combining $\mathcal{A}$ and the first two components has the following property: it contains precisely one tuple in Tile encoding τ, for each $\tau \in \mathrm{Tiles}(\mathcal{K})$.

Building the enriched system We now present the rules that build the enriched system $\mathcal{S}_\mathcal{K}$. As each tile represents a variable in the system, we store tiles for $\mathcal{K}$ in the relation Var by adding the rule

$$\mathsf{Var}(\vec{t}) \leftarrow \mathsf{Tile}(\vec{t}).$$

Next, we compute the relations that encode the inequalities and implications in $\mathcal{S}_\mathcal{K}$. To this end, we agree on the convention, summarized in Table 4.2, that assigns to each implication of the form (3.2), and inequality of the form (3.1) occurring in $\mathcal{S}_\mathcal{K}$ a string of length l_{id} over the available constants that uniquely identifies it.

The IDs of inequalities and implications are stored in relations Iq and Im, respectively. Additionally, we use the relation Iq^* to store the IDs of true inequalities, i.e., those that must be satisfied in the system. Note that $\mathsf{Iq}^* \subseteq \mathsf{Iq}$. Therefore, we add the following rule:

$$\mathsf{Iq}(\vec{x}) \leftarrow \mathsf{Iq}^*(\vec{x}).$$

Once we agree on how IDs are assigned to the implications and inequalities, we go through the conditions in Definition 3.2.9 and add the rules that compute the relevant relations storing implications and inequalities.

M1. As Table 4.2 suggests, for each knowledge base constant c, M1 gives rise to two inequalities identified by $(0, c, \vec{0})$ and $(1, c, \vec{0})$. We add the rules that store these IDs in Iq^*:

$$\mathsf{Iq}^*(0, x, \vec{0}) \leftarrow \mathsf{Adom}(x), \qquad \mathsf{Iq}^*(1, x, \vec{0}) \leftarrow \mathsf{Adom}(x).$$

We next need to relate these inequalities with the variables and integers that occur in them using relations $\mathsf{Iq}^{\mathsf{v}}_{\mathsf{L}}$, $\mathsf{Iq}^{\mathsf{v}}_{\mathsf{R}}$, $\mathsf{Iq}^{\mathsf{int}}_{\mathsf{L}}$, and $\mathsf{Iq}^{\mathsf{int}}_{\mathsf{R}}$.

On the LHS of the inequality identified by $(0, c, \vec{0})$ as well as the RHS of the inequality identified by $(1, c, \vec{0})$, where c is, once again, a concrete knowledge base

constant, we have 1) no integers and 2) all tiles (T, ρ) for which $\{c\} \in T$. Thus, we add:

$$\mathsf{Iq}^{\mathsf{v}}_{\mathsf{L}}(0, x, \vec{0}, \vec{T}, \vec{r}) \leftarrow \mathsf{Iq}(0, x, \vec{0}), \mathsf{Adom}(x), \mathsf{Var}(\vec{T}, \vec{r}), w = x,$$
$$\mathsf{Iq}^{\mathsf{v}}_{\mathsf{R}}(1, x, \vec{0}, \vec{T}, \vec{r}) \leftarrow \mathsf{Iq}(1, x, \vec{0}), \mathsf{Adom}(x), \mathsf{Var}(\vec{T}, \vec{r}), w = x,$$

where $\vec{T} = x_1, \ldots, x_{n_\mathcal{T}}, w$.

On the RHS of the inequality identified by $(0, c, \vec{0})$ as well as the LHS of the inequality identified by $(1, c, \vec{0})$ is equal to 1. We encode this using the following:

$$\mathsf{Iq}^{\mathsf{int}}_{\mathsf{R}}(0, x, \vec{0}, 1) \leftarrow \mathsf{Adom}(x),$$
$$\mathsf{Iq}^{\mathsf{int}}_{\mathsf{L}}(1, x, \vec{0}, 1) \leftarrow \mathsf{Adom}(x).$$

M2. The inequality in M2 is simply identified by $\vec{0}$ of length l_{id}. Thus, we add the fact $\mathsf{Iq}^*(\vec{0})$. On the RHS of this inequality we have all variables and on the LHS we have the constant 1. Thus, we add:

$$\mathsf{Iq}^{\mathsf{v}}_{\mathsf{R}}(\vec{0}, \vec{t}) \leftarrow \mathsf{Var}(\vec{t}) \qquad \text{and} \qquad \mathsf{Iq}^{\mathsf{int}}_{\mathsf{L}}(\vec{0}, 1).$$

M3. For all types $T, T' \in \mathrm{Types}(\mathcal{K})$ and a role type $R \subseteq \mathsf{N}^+_{\mathsf{R}}(\mathcal{K})$ such that (T, R, T') is invertible, the condition M3 introduces two inequalities identified by $(0, \vec{T}, \vec{T'}, \vec{R}, \vec{0})$ and $(1, \vec{T}, \vec{T'}, \vec{R}, \vec{0})$, where $\vec{T}$ and $\vec{T'}$ identify T and T', respectively, and $\vec{R}$ identifies R. We thus add the following rules to store these IDs in Iq^* using the following rules:

$$\mathsf{Iq}^*(0, \vec{T}, \vec{T'}, \vec{R}, \vec{0}) \leftarrow \mathsf{Type}(\vec{T}), \mathsf{Type}(\vec{T'}), \mathsf{RType}(\vec{R}), \mathsf{Invrt}(\vec{T}, \vec{R}, \vec{T'}),$$
$$\mathsf{Iq}^*(1, \vec{T}, \vec{T'}, \vec{R}, \vec{0}) \leftarrow \mathsf{Type}(\vec{T}), \mathsf{Type}(\vec{T'}), \mathsf{RType}(\vec{R}), \mathsf{Invrt}(\vec{T}, \vec{R}, \vec{T'}).$$

On the LHS of the inequality identified by $(0, \vec{T}, \vec{T'}, \vec{R}, \vec{0})$ and the RHS of the inequality identified by $(1, \vec{T}, \vec{T'}, \vec{R}, \vec{0})$, we have all tiles (T, ρ) with $(R, T', k) \in \rho$, for some k. Thus, we add:

$$\mathsf{Iq}^{\mathsf{v}}_{\mathsf{L}}(0, \vec{T}, \vec{T'}, \vec{R}, \vec{0}, \vec{t}) \leftarrow \mathsf{Iq}^*(0, \vec{T}, \vec{T'}, \vec{R}, \vec{0}), \mathsf{Var}(\vec{t}), \vec{R}_i = \vec{R}, \vec{T}_i = \vec{T'},$$
$$\mathsf{Iq}^{\mathsf{v}}_{\mathsf{R}}(1, \vec{T}, \vec{T'}, \vec{R}, \vec{0}, \vec{t}) \leftarrow \mathsf{Iq}^*(1, \vec{T}, \vec{T'}, \vec{R}, \vec{0}), \mathsf{Var}(\vec{t}), \vec{R}_i = \vec{R}, \vec{T}_i = \vec{T'},$$

for all $1 \leq i \leq n$, where $\vec{t} = \vec{T}, \vec{R}_1, \vec{T}_1, \vec{k}_1, \ldots, \vec{R}_n, \vec{T}_n, \vec{k}_n$.

Further, on the RHS of the inequality identified by $(0, \vec{T}, \vec{T'}, \vec{R}, \vec{0})$ and the LHS of the inequality identified by $(1, \vec{T}, \vec{T'}, \vec{R}, \vec{0})$, we have all tiles (T', ρ') with $(R^-, T, k) \in \rho'$, for some k. We therefore add:

$$\mathsf{Iq}^{\mathsf{v}}_{\mathsf{R}}(0, \vec{T}, \vec{T'}, \vec{R}, \vec{0}, \vec{t}) \leftarrow \mathsf{Iq}^*(0, \vec{T}, \vec{T'}, \vec{R}, \vec{0}), \mathsf{Var}(\vec{t}), \mathsf{RType}^-(\vec{R}, \vec{R'}), \vec{R}_i = \vec{R'}, \vec{T}_i = \vec{T},$$
$$\mathsf{Iq}^{\mathsf{v}}_{\mathsf{L}}(1, \vec{T}, \vec{T'}, \vec{R}, \vec{0}, \vec{t}) \leftarrow \mathsf{Iq}^*(1, \vec{T}, \vec{T'}, \vec{R}, \vec{0}), \mathsf{Var}(\vec{t}), \mathsf{RType}^-(\vec{R}, \vec{R'}), \vec{R}_i = \vec{R'}, \vec{T}_i = \vec{T},$$

for all $1 \leq i \leq n$, where $\vec{t} = \vec{T'}, \vec{R}_1, \vec{T}_1, \vec{k}_1, \ldots, \vec{R}_n, \vec{T}_n, \vec{k}_n$.

The rules above make use of an auxiliary relation RType^-, that stores $(\vec{R}, \vec{R}')$ if $\vec{R}$ encodes a role type R and $\vec{R}'$ encodes R^-. To compute RType^-, for each role pair of roles $r_g, r_{g'} \in \mathsf{N}_\mathsf{R}^+(\mathcal{K})$ such that $r_{g'} = r_g^-$, we add the following:

$$\overline{\mathsf{RType}^-}(\vec{R}, \vec{R}') \leftarrow \mathsf{RType}(\vec{R}), \mathsf{RType}(\vec{R}'), \mathsf{In}_{r_g}(\vec{R}), \mathit{not}\ \mathsf{In}_{r_{g'}}(\vec{R}'),$$
$$\overline{\mathsf{RType}^-}(\vec{R}, \vec{R}') \leftarrow \mathsf{RType}(\vec{R}), \mathsf{RType}(\vec{R}'), \mathsf{In}_{r_g}(\vec{R}'), \mathit{not}\ \mathsf{In}_{r_{g'}}(\vec{R}),$$
$$\mathsf{RType}^-(\vec{R}, \vec{R}') \leftarrow \mathsf{RType}(\vec{R}), \mathsf{RType}(\vec{R}'), \mathit{not}\ \overline{\mathsf{RType}^-}(\vec{R}, \vec{R}').$$

The first two rules tell us that for two role types R and R', $R' \neq R^-$, if there is some role r such that (i) $r \in R$ but $r^- \notin R'$ or (ii) $r \in R'$ but $r^- \notin R$. The third rule lets us infer that if we cannot find such r, then $R' = R$, and thus $(\vec{R}, \vec{R}') \in \mathsf{RType}^-$.

M4. For each tile $\tau = (T, \rho) \in \mathrm{Tiles}(\mathcal{K})$ and each type $T' \in \mathrm{Types}(\mathcal{K})$, the condition M4 introduces an implication with the ID $(\vec{t}, \vec{T}', \vec{0})$, which in turn consists of two inequalities identified by $(\vec{t}, \vec{0})$ and $(\vec{t}, \vec{T}', \vec{0})$, where $\vec{t}$ encodes τ and $\vec{T}'$ encodes T'. We define rules that store these IDs in Im and Iq:

$$\mathsf{Im}(\vec{t}, \vec{T}', \vec{0}) \leftarrow \mathsf{Tile}(\vec{t}), \mathsf{Type}(\vec{T}'),$$
$$\mathsf{Iq}(\vec{t}, \vec{0}) \leftarrow \mathsf{Tile}(\vec{t}),$$
$$\mathsf{Iq}(\vec{t}, \vec{T}', \vec{0}) \leftarrow \mathsf{Tile}(\vec{t}), \mathsf{Type}(\vec{T}').$$

We also add the rules that relate the implications with the inequalities they consist of:

$$\mathsf{Im}_\mathsf{L}(\vec{t}, \vec{T}', \vec{0}_1, \vec{t}, \vec{0}_2) \leftarrow \mathsf{Im}(\vec{t}, \vec{T}', \vec{0}_1), \mathsf{Iq}(\vec{t}, \vec{0}_2), \mathsf{Tile}(\vec{t}), \mathsf{Type}(\vec{T}'),$$
$$\mathsf{Im}_\mathsf{R}(\vec{t}, \vec{T}', \vec{0}_1, \vec{t}, \vec{T}', \vec{0}_2) \leftarrow \mathsf{Im}(\vec{t}, \vec{T}', \vec{0}_1), \mathsf{Iq}(\vec{t}, \vec{T}', \vec{0}_2), \mathsf{Tile}(\vec{t}), \mathsf{Type}(\vec{T}').$$

Further, we need to relate the newly introduced inequalities with the corresponding variables and integers. Observe that the LHS of the inequality with the ID $(\vec{t}, \vec{0})$ is equal to 1 and the RHS simply consists of the tile encoded by $\vec{t}$. Thus, we have:

$$\mathsf{Iq}^\mathsf{v}_\mathsf{R}(\vec{t}, \vec{0}, \vec{t}) \leftarrow \mathsf{Iq}(\vec{t}, \vec{0}), \mathsf{Tile}(\vec{t}),$$
$$\mathsf{Iq}^\mathsf{int}_\mathsf{L}(\vec{t}, \vec{0}, 1) \leftarrow \mathsf{Iq}(\vec{t}, \vec{0}), \mathsf{Tile}(\vec{t}),$$

On the LHS of the inequality with the ID $(\vec{t}, \vec{T}', \vec{0})$ we have all tiles $\tau' = (T', \rho')$ such that T' is encoded by $\vec{T}'$, while on the RHS we have the number of triples in ρ that contains the type encoded by $\vec{T}'$. We add the following rules:

$$\mathsf{Iq}^\mathsf{int}_\mathsf{L}(\vec{t}, \vec{T}', \vec{0}, \vec{k}) \leftarrow \mathsf{Iq}(\vec{t}, \vec{T}', \vec{0}), \mathsf{Tile}(\vec{t}), \mathsf{Type}(\vec{T}'), \mathsf{Upto}_n(\vec{t}, \vec{T}', \vec{k}),$$
$$\mathsf{Iq}^\mathsf{v}_\mathsf{R}(\vec{t}, \vec{T}', \vec{0}, \vec{t}') \leftarrow \mathsf{Iq}(\vec{t}, \vec{T}', \vec{0}), \mathsf{Tile}(\vec{t}), \mathsf{Type}(\vec{T}'), \mathsf{Tile}(\vec{t}'),$$

where $\vec{t} = \vec{T}, \vec{R}_1, \vec{T}_1, \vec{k}_1, \ldots, \vec{R}_n, \vec{T}_n, \vec{k}_n$ and $\vec{t}' = \vec{T}', \vec{R}'_1, \vec{T}'_1, \vec{k}'_1, \ldots, \vec{R}'_n, \vec{T}'_n, \vec{k}'_n$.

The relation Upto_i, for all $1 \leq i \leq n$, is an auxiliary relation with the following meaning. Let (T, ρ) be a tile identified by $\vec{t}$, T' be a type identified by $\vec{T'}$ and k be an integer whose binary representation $\vec{k}$ is stored in Int. The relation Upto_i stores the tuple $(\vec{t}, \vec{T'}, \vec{k})$ if and only if among the first i triples in ρ, there are exactly k triples of the form (R, T', l), and is computed by following rules:

$$\begin{aligned}
&\mathsf{Upto}_0(\vec{t}, \vec{T'}, \vec{0}) \leftarrow \mathsf{Tile}(\vec{t}), \mathsf{Type}(\vec{T'}),\\
&\mathsf{Upto}_i(\vec{t}, \vec{T'}, \vec{k}) \leftarrow \mathsf{Upto}_{i-1}(\vec{t}, \vec{T'}, \vec{k'}), \mathsf{Succ}_{\mathsf{int}}(\vec{k'}, \vec{k}), \vec{T_i} = \vec{T'},\\
&\mathsf{Upto}_i(\vec{t}, \vec{T'}, \vec{k}) \leftarrow \mathsf{Upto}_{i-1}(\vec{t}, \vec{T'}, \vec{k}), \mathit{not}\ \vec{T_i} \neq \vec{T'},
\end{aligned}$$

for each $1 \leq i \leq n$, where $\vec{t} = \vec{T}, \vec{R}_1, \vec{T}_1, \vec{k}_1, \ldots, \vec{R}_n, \vec{T}_n, \vec{k}_n$.

For the next two conditions, let $\vec{b}_i$ be the vector of constants that corresponds to the ID of the type for $\mathcal{K}$ that contains only B_i, for each concept name B_i, $1 \leq i \leq n_\mathcal{T}$, and let $\vec{r}_j$ be the vector of constants that corresponds to the ID of the role type for $\mathcal{K}$ that contains only r_j, for each role r_j, $1 \leq j \leq k_\mathcal{T}$.

M5. For all knowledge base constants c_1, c_2 and all concept names $B_i, B_j \in \mathsf{N_C}(\mathcal{K})$ fulfilling certain conditions (see Definition 3.2.9), M5 introduces an implication with the ID $(c_1, c_2, \vec{b}_i, \vec{b}_j, \vec{0})$. This implication, in turn, consists of two inequalities with the IDs $(c_1, \vec{b}_i, \vec{0})$ and $(c_2, \vec{b}_j, \vec{0})$.

We first deal with the condition (a). For every pair of axioms $B_{i_1} \sqsubseteq \forall r_h.B_{i_2} \in \mathcal{T}$ and $r_g \sqsubseteq r_h$, we add:

$$\begin{aligned}
\mathsf{Im}(x, y, \vec{b}_{i_1}, \vec{b}_{i_2}, \vec{0}) &\leftarrow r_g(x, y),\\
\mathsf{Iq}(x, \vec{b}_{i_1}, \vec{0}) &\leftarrow r_g(x, y),\\
\mathsf{Iq}(y, \vec{b}_{i_2}, \vec{0}) &\leftarrow r_g(x, y).
\end{aligned}$$

Similarly, we deal with the condition (b) as follows. For every pair of axioms $B_{i_1} \sqsubseteq \forall r_h.B_{i_2} \in \mathcal{T}$ and $r_g^- \sqsubseteq r_h$, we add:

$$\begin{aligned}
\mathsf{Im}(x, y, \vec{b}_{i_1}, \vec{b}_{i_2}, \vec{0}) &\leftarrow r_g(y, x),\\
\mathsf{Iq}(y, \vec{b}_{i_1}, \vec{0}) &\leftarrow r_g(x, y),\\
\mathsf{Iq}(x, \vec{b}_{i_2}, \vec{0}) &\leftarrow r_g(x, y).
\end{aligned}$$

To relate the implications with the corresponding inequalities, we do the following:

$$\begin{aligned}
\mathsf{Im_L}(x, y, \vec{T}, \vec{T'}, \vec{0}, x, \vec{T}, \vec{0}) &\leftarrow \mathsf{Im}(x, y, \vec{T}, \vec{T'}, \vec{0}), \mathsf{Iq}(x, \vec{T}, \vec{0}), \mathsf{Adom}^*(x), \mathsf{Adom}^*(y),\\
\mathsf{Im_R}(x, y, \vec{T}, \vec{T'}, \vec{0}, y, \vec{T'}, \vec{0}) &\leftarrow \mathsf{Im}(x, y, \vec{T}, \vec{T'}, \vec{0}), \mathsf{Iq}(y, \vec{T'}, \vec{0}), \mathsf{Adom}^*(x), \mathsf{Adom}^*(y).
\end{aligned}$$

The last step is to compute the LHS and RHS of the introduced inequalities. The LHS of the inequality with the ID $(c, \vec{b}_i, \vec{0})$ is equal to 1, while on the RHS we have all tiles (T, ρ) such that $\{\{c\}, B_i\} \subseteq T$. Thus, for all $B_i \in \mathsf{N_C}(\mathcal{K})$, we add:

$$\mathsf{Iq}^{\mathsf{int}}_{\mathsf{L}}(x, \vec{b}_i, \vec{0}, 1) \leftarrow \mathsf{Iq}(x, \vec{b}_i, \vec{0}), \mathsf{Adom}^*(x),$$
$$\mathsf{Iq}^{\mathsf{v}}_{\mathsf{R}}(x, \vec{b}_i, \vec{0}, \vec{t}) \leftarrow \mathsf{Iq}(x, \vec{b}_i, \vec{0}), \mathsf{Adom}^*(x), \mathsf{Tile}(\vec{t}), \mathsf{In}_{B_i}(\vec{T}), w = x,$$

where $\vec{t} = \vec{T}, \vec{r}$ and $\vec{T} = x_1, \ldots, x_{n_\mathcal{T}}, w$.

M6. Finally, for every tile $\tau' = (T', \rho') \in \mathrm{Tiles}(\mathcal{K})$, every knowledge base constant c, every $B_i \in \mathsf{N_C}(K)$ and every $r_j \in \mathsf{N}^+_{\mathsf{R}}(\mathcal{K})$ that satisfy some conditions (see Definition 3.2.9), M6 gives rise to an implication with the ID $(c, \vec{t}', \vec{b}_i, \vec{r}_j, \vec{0})$, which in turn consists of two inequalities with the IDs $(c, \vec{T}', \vec{b}_i, \vec{r}_j, \vec{0})$ and $(\vec{t}', 0)$, where $\vec{t}'$ encodes τ' and $\vec{T}'$ encodes T'.

Note that the inequalities with the ID of the form $(\vec{t}, 0)$, where $\vec{t}$ identifies a tile, were already computed in M4.

We first deal with (a) and for all pairs of axioms $p \sqsubseteq r_h$ and $B_{i_1} \sqsubseteq= m\ r_h.B_{i_2}$ in $\mathcal{T}$, we add:

$$\mathsf{Im}(x, \vec{t}, \vec{b}_{i_1}, \vec{r}_h, \vec{0}) \leftarrow \mathsf{Tile}(\vec{t}), p(x, y), \mathsf{In}_{B_{i_2}}(\vec{T}), w = y,$$
$$\mathsf{Iq}(x, \vec{T}, \vec{b}_{i_1}, \vec{r}_h, \vec{0}) \leftarrow \mathsf{Tile}(\vec{t}), p(x, y), \mathsf{In}_{B_{i_2}}(\vec{T}), w = y,$$

where $\vec{t} = \vec{T}, \vec{r}$ and $\vec{T} = x_1, \ldots, x_{n_\mathcal{T}}, w$.

The condition(b) is dealt with in a similar manner. Namely, for all pairs of axioms $p^- \sqsubseteq r_h$ and $B_{i_1} \sqsubseteq= m\ r_h.B_{i_2}$ in $\mathcal{T}$, we add the following rule

$$\mathsf{Im}(x, \vec{t}, \vec{b}_{i_1}, \vec{r}_h, \vec{0}) \leftarrow \mathsf{Tile}(\vec{t}), p(y, x), \mathsf{In}_{B_{i_2}}(\vec{T}), w = y,$$

where $\vec{t} = \vec{T}, \vec{r}$ and $\vec{T} = x_1, \ldots, x_{n_\mathcal{T}}, w$.

We then relate inequalities to the corresponding implications:

$$\mathsf{Im}_{\mathsf{L}}(x, \vec{t}, \vec{b}_{i_1}, \vec{r}_h, \vec{0}, \vec{t}, \vec{0}) \leftarrow \mathsf{Im}(x, \vec{t}, \vec{b}_{i_1}, \vec{r}_h, \vec{0}), \mathsf{Iq}(\vec{t}, \vec{0}), \mathsf{Adom}^*(x), \mathsf{Tile}(\vec{t}),$$
$$\mathsf{Im}_{\mathsf{R}}(x, \vec{t}, \vec{b}_{i_1}, \vec{r}_h, \vec{0}, x, \vec{T}, \vec{b}_{i_1}, \vec{r}_h, \vec{0}) \leftarrow \mathsf{Im}(x, \vec{t}, \vec{b}_{i_1}, \vec{r}_h, \vec{0}), \mathsf{Iq}(x, \vec{T}, \vec{b}_{i_1}, \vec{r}_h, \vec{0}), \mathsf{Adom}^*(x), \mathsf{Tile}(\vec{t}),$$

where $\vec{t} = \vec{T}, \vec{r}$ and $\vec{T} = x_1, \ldots, x_{n_\mathcal{T}}, w$.

Finally, we compute the LHS and RHS of the newly introduced inequalities. The RHS of the inequality with the ID $(c, \vec{T}', \vec{b}_i, \vec{r}_h, \vec{0})$ is equal to 0, while the LHS contains all tiles $\tau = (T, \rho)$ such that $\{\{c\}, B_i\} \in T$ and there is no $(R, T', k) \in \rho$

such that $r_h \in R$. Thus, for all $B_i \in \mathsf{N_C}(\mathcal{K})$ and $r_h \in \mathsf{N_R^+}(\mathcal{K})$, we add:

$$\mathsf{Iq}^{\mathsf{int}}_{\mathsf{R}}(x, \vec{T}', \vec{b}_i, \vec{r}_h, \vec{0}, \vec{k}) \leftarrow \mathsf{Iq}(x, \vec{T}', \vec{b}_i, \vec{r}_h, \vec{0}), \mathsf{Adom}^*(x), \mathsf{Int}(\vec{k}), \vec{k} = (0, \ldots, 0),$$

$$\mathsf{Iq}^{\mathsf{v}}_{\mathsf{L}}(x, \vec{T}', \vec{b}_i, \vec{r}_h, \vec{0}, \vec{t}) \leftarrow \mathsf{Iq}(x, \vec{T}', \vec{b}_i, \vec{r}_h, \vec{0}), \mathsf{Tile}(\vec{t}), \mathsf{Adom}^*(x), \\ w = x, \mathsf{In}_{B_{i_1}}(\vec{T}), \mathsf{Upto}_{n,r_h}(\vec{t}, \vec{T}, 0, \ldots, 0),$$

for all $r_h \in \mathsf{N_R^+}(\mathcal{K})$, where $\vec{t} = \vec{T}, \vec{r}$ and $\vec{T} = x_1, \ldots, x_{n_\mathcal{T}}, w$.

Similar to the previously introduced relation Upto_i, the relation Upto_{i,r_h}, for all $1 \leq i \leq n$ is an auxiliary relation with the following meaning. Let (T, ρ) be a tile identified by $\vec{t}$, T' be a type identified by $\vec{T}'$, r_h be a role in $\mathsf{N_R^+}(\mathcal{K})$, and k be an integer whose binary representation $\vec{k}$ is stored in Int. The relation Upto_{i,r_h} stores the tuple $(\vec{t}, \vec{T}', \vec{k})$ if and only if among the first i triples in ρ, there are exactly k triples of the form (R, T', l), where $r_h \in R$, and is computed by following rules:

$$\mathsf{Upto}_{0,r_h}(\vec{t}, \vec{T}', \vec{0}) \leftarrow \mathsf{Tile}(\vec{t}), \mathsf{Type}(\vec{T}')$$

$$\mathsf{Upto}_{i,r_h}(\vec{t}, \vec{T}', \vec{k}) \leftarrow \mathsf{Upto}_{i-1,r_h}(\vec{t}, \vec{T}', \vec{k}'), \mathsf{Succ}_{\mathsf{int}}(\vec{k}', \vec{k}), \vec{T}_i = \vec{T}', \mathsf{In}_{r_h}(\vec{R}_i)$$

$$\mathsf{Upto}_{i,r_h}(\vec{t}, \vec{T}', \vec{k}) \leftarrow \mathsf{Upto}_{i-1,r_h}(\vec{t}, \vec{T}', \vec{k}), \mathit{not}\ \vec{T}_i \neq \vec{T}'$$

$$\mathsf{Upto}_{i,r_h}(\vec{t}, \vec{T}', \vec{k}) \leftarrow \mathsf{Upto}_{i-1,r_h}(\vec{t}, \vec{T}', \vec{k}), \mathit{not}\ \mathsf{In}_{r_h}(\vec{R}_i),$$

for each $1 \leq i \leq n$ and counting role $r_h \in \mathsf{N_R^+}(\mathcal{K})$, where $\vec{T} = x_1, \ldots, x_{n_\mathcal{T}}, w$ and $\vec{t} = \vec{T}, \vec{R}_1, \vec{T}_1, \vec{k}_1, \ldots, \vec{R}_n, \vec{T}_n, \vec{k}_n$.

At this point in the construction, each answer set of the program that combines an input ABox $\mathcal{A}$ with the three components described above corresponds to $\mathit{Rel}(\mathcal{S}_\mathcal{K})$, up to the renaming of IDs of the variables, implications and inequalities.

Checking that $\mathcal{A}$ respects role inclusions To complete our construction of $\mathcal{P}^{\mathcal{T},\Sigma}_{sys}$, we add the rules that ensure that $\mathcal{K}$ respects role inclusions, i.e., that it satisfies the conditions listed in Definition 3.2.11. This is achieved via constraints (i.e., rules with empty bodies) and is rather straightforward:

1. For all $r \sqsubseteq s \in \mathcal{T}$, we add
$$\leftarrow \overline{s}(x, y), r(x, y).$$
2. For all $r^- \sqsubseteq s \in \mathcal{T}$, we add
$$\leftarrow \overline{s}(x, y), r(y, x).$$
3. For all $r \in \Sigma \cap \mathsf{N_R}$ and all $s \sqsubseteq r \in \mathcal{T}$, we add
$$\leftarrow s(x, y), \mathit{not}\ r(x, y).$$

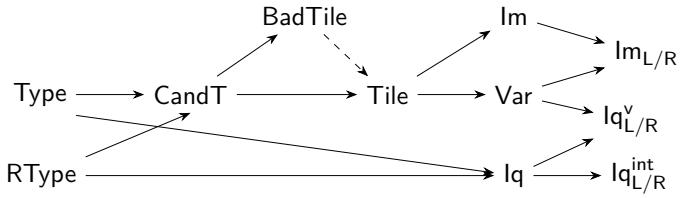


Figure 4.2: Partial dependency graph of $\mathcal{P}_{sat}^{\mathcal{T},\Sigma}$ (negation represented via dashed arcs).

4. For all $r \in \Sigma \cap \mathsf{N_R}$ and all $s^- \sqsubseteq r \in \mathcal{T}$, we add

$$\leftarrow s(x,y), \mathit{not}\ r(y,x).$$

We now make a couple of observations about the constructed program $\mathcal{P}_{sys}^{\mathcal{T},\Sigma}$. First, the arities of the predicates occurring in $\mathcal{P}_{sys}^{\mathcal{T},\Sigma}$ are polynomial in the size of $\mathcal{T}$ and Σ and do not depend on $\mathcal{A}$ (see Table 4.1). Moreover, looking at the encoding above, we can easily verify that the number of non-ground rules in $\mathcal{P}_{sys}^{\mathcal{T},\Sigma}$ is polynomial in the size of $\mathcal{T}$ and Σ and does not depend on $\mathcal{A}$. Further, we already argued during the construction that the answer sets of the first three components together extended with A correspond to the $\mathit{Rel}(\mathcal{S}_{\mathcal{K}})$. The fourth component consists of the constraints that simply "kill" all the answer sets if $\mathcal{K}$ violates one of the conditions 1-4 in Definition 3.2.11. This leads us to the following statement.

Proposition 4.1.1. *Given a TBox $\mathcal{T}$ and a set Σ of closed predicates, $\mathcal{P}_{sys}^{\mathcal{T},\Sigma}$ has the following properties:*

- *$\mathcal{P}_{sys}^{\mathcal{T},\Sigma}$ is polynomial in the size of $\mathcal{T}$ and Σ and it does not depend on the size of the input ABox,*
- *for any ABox $\mathcal{A}$ over the signature of $\mathcal{T}$, $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$ does not respect role inclusions if and only if the program $(\mathcal{P}_{sys}^{\mathcal{T},\Sigma}, \hat{\mathcal{A}})$ has no answer sets,*
- *each answer set of $(\mathcal{P}_{sys}^{\mathcal{T},\Sigma}, \hat{\mathcal{A}})$ corresponds to $\mathit{Rel}(\mathcal{S}_{\mathcal{K}})$, up to the renaming of IDs of the variables, implications and inequalities.*

4.1.2 Solving Linear Inequalities

We next discuss the construction of the program $\mathcal{P}_{sol}^{\mathcal{T},\Sigma}$ that, given a relational representation of an enriched system $\mathcal{S}$ encoded using the previously-described signature, decides whether there exists a solution over $\mathbb{N}^*$ to $\mathcal{S}$. Running this program on a system generated by the program $(\mathcal{P}_{sys}^{\mathcal{T},\Sigma}, \hat{A})$ thus decides whether there is a mosaic for $(\mathcal{T}, \Sigma, \mathcal{A})$. We note that $\mathcal{P}_{sol}^{\mathcal{T},\Sigma}$ depends on $\mathcal{T}$ and Σ only in terms of the arity of the predicates that

are shared with $\mathcal{P}_{sys}^{\mathcal{T},\Sigma}$ and can handle arbitrary enriched systems, as long as they are encoded using the provided signature.

The intuition behind $\mathcal{P}_{sol}^{\mathcal{T},\Sigma}$ is relatively simple – it is a 'guess-and-check' procedure that non-deterministically guesses the values of the variables of the input enriched system and then checks whether the guess is valid, i.e., whether all implications and inequalities of the system are satisfied. However, for this approach to work we need to ensure that the search space that we have to explore in order to determine whether an enriched system has a solution is finite. To this end, we recall the result from Proposition 3.2.22 of the previous chapter stating that the existence of a solution over $\mathbb{N}^*$ implies the existence of a solution over $\mathbb{N}^*$ in which variables are assigned values that are at most exponential in the size of the system. This result indeed ensures that $\mathcal{P}_{sol}^{\mathcal{T},\Sigma}$ has to consider only finitely many different options for the values of the variables. However, those values are extremely large. Moreover, recall that, for a given KB $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$, the enriched system $\mathcal{S}_\mathcal{K}$ is already exponential in the size of $\mathcal{T}$ and Σ, which means that the values that we have to consider for determining whether $\mathcal{S}_\mathcal{K}$ has a solution are, in fact, doubly exponential in the size of $\mathcal{T}$ and Σ. As we would like to keep $\mathcal{P}_{sol}^{\mathcal{T},\Sigma}$ polynomial in the size of $\mathcal{T}$ and Σ, the first and most crucial step in our construction is to come up with a *succinct representation of solutions*. To this end, consider an enriched system $\mathcal{S}$. Let d denote the number of constants in Cst and l_v, l_e, l_i, and l_a denote the arity of Var, Iq, Im, and Int in the relational representation of $\mathcal{S}$, respectively. Since variables, inequalities, and implications are encoded as strings over Cst of length l_v, l_e and l_i, respectively, we know that there are at most d^{l_v} variables, at most d^{l_e} inequalities and at most d^{l_i} implications. Further, the maximum integer occurring in $\mathcal{S}$ is bounded by d^{l_a}. Let $l = (l_v + l_e + l_i + l_a)$. In view of Proposition 3.2.22, for deciding whether $\mathcal{S}$ has a solution it is sufficient to consider only those solutions whose finite values do not exceed $2^{d^{2l}}$. Thus, the maximum finite value that our program must be able to handle is bounded by $2^{d^{3l}}$. A naive way of encoding a solution S is to associate to each variable whose value in S is finite, a binary string of length d^{3l} that encodes this value. However, this makes the translation both exponential as well as dependent on the number of constants in the data, which goes against our goal of having a polynomial, data-independent translation. We overcome this challenge in the following way: instead of having binary strings of length d^{3l}, we encode the addresses of these d^{3l} bits as a string of length $3l$ over the constants in Cst. We then encode the values of the variables using a $l_v + 3l + 1$-ary predicate Val, with the following meaning: $(\vec{x}, \vec{z}, b)$ in the relation Val denotes that in the value of the variable encoded by $\vec{x}$ the bit at the position $\vec{z}$ has the value b, where b is either 0 or 1. This is depicted in Figure 4.3.

We are now ready to present the guessing part of $\mathcal{P}_{sol}^{\mathcal{T},\Sigma}$, which is rather straightforward. In the remainder of this section, we write $\mathsf{Cst}^i(x_1, \ldots, x_i)$ to abbreviate $\mathsf{Cst}(x_1), \ldots \mathsf{Cst}(x_i)$, for $i \geq 1$. We begin by adding the rules that guess which variables in a potential solution are set to infinity:

$$\mathsf{Fin}(\vec{x}) \leftarrow \mathsf{Var}(\vec{x}), \mathit{not}\ \mathsf{Inf}(\vec{x}), \qquad \mathsf{Inf}(\vec{x}) \leftarrow \mathsf{Var}(\vec{x}), \mathit{not}\ \mathsf{Fin}(\vec{x}).$$

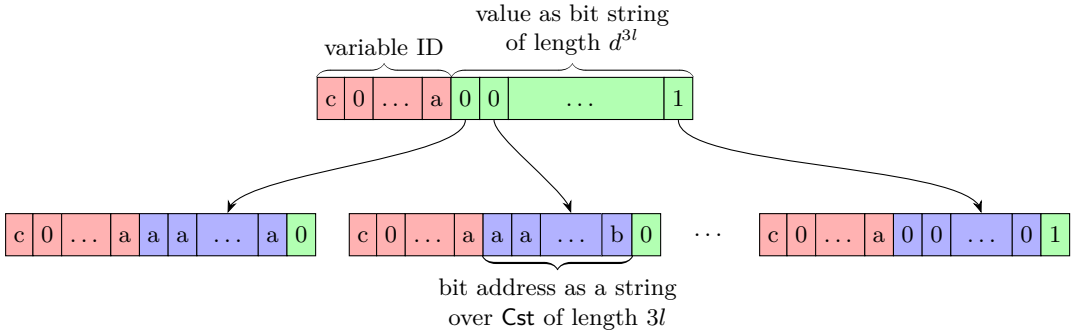


Figure 4.3: Naive (top) vs. our encoding (bottom) of solutions in $\mathcal{P}_{sol}^{\mathcal{T},\Sigma}$

If a variable is not set to infinity, we separately guess each bit of its value by using the following rules:

$$\mathsf{Val}(\vec{x},\vec{z}',\vec{z},0) \leftarrow \mathsf{Fin}(\vec{x}),\mathsf{Cst}^{l}(\vec{z}'),\mathsf{Cst}^{2l}(\vec{z}),\vec{z}' \neq (0,\ldots,0),$$
$$\mathsf{Val}(\vec{x},\vec{z},0) \leftarrow \mathsf{Fin}(\vec{x}),\mathsf{Cst}^{3l}(\vec{z}),\mathit{not}\ \mathsf{Val}(\vec{x},\vec{z},1),$$
$$\mathsf{Val}(\vec{x},\vec{z},1) \leftarrow \mathsf{Fin}(\vec{x}),\mathsf{Cst}^{3l}(\vec{z}),\mathit{not}\ \mathsf{Val}(\vec{x},\vec{z},0).$$

As the variables take values that are bounded by $2^{d^{2l}}$, we only need the first d^{2l} bits to encode them. Note that *first* here refers to the linear order from LEQ in the relational representation of $\mathcal{S}$ and recall that 0 is the least constant according to LEQ. The remaining d^l bits are set to 0 and reserved for accommodating addition, which is reflected in the first rule. The other two rules freely guess the values of the first d^{2l} bits.

We now move to the checking part of $\mathcal{P}_{sol}^{\mathcal{T},\Sigma}$. We use the l_e-ary predicate Sat to store the inequalities that are satisfied by our guess. Due to the shape of the enriched systems (see Definition 3.2.18), checking whether inequalities with occurrences of $\aleph_0$ (i.e., infinity) are satisfied is easy – such inequality is satisfied if and only if infinity occurs on its RHS. We thus have the following rule:

$$\mathsf{Sat}(\vec{y}) \leftarrow \mathsf{Var}(\vec{x}),\mathsf{Iq}(\vec{y}),\mathsf{Iq}_{\mathsf{R}}(\vec{y},\vec{x}),\mathsf{Inf}(\vec{x}).$$

We next focus on the inequalities where all the variables are assigned finite values. These inequalities are stored in the relation FinIq, which is computed as follows:

$$\mathsf{InfIq}(\vec{y}) \leftarrow \mathsf{Iq}(\vec{y}),\mathsf{Var}(\vec{x}),\mathsf{Inf}(\vec{x}),\mathsf{Iq}_{\mathsf{L}}(\vec{y},\vec{x}),$$
$$\mathsf{InfIq}(\vec{y}) \leftarrow \mathsf{Iq}(\vec{y}),\mathsf{Var}(\vec{x}),\mathsf{Inf}(\vec{x}),\mathsf{Iq}_{\mathsf{R}}(\vec{y},\vec{x}),$$
$$\mathsf{FinIq}(\vec{y}) \leftarrow \mathsf{Iq}(\vec{y}),\mathit{not}\ \mathsf{InfIq}(\vec{y}).$$

To check whether an inequality in FinIq is satisfied by our guess, we incrementally compute its LHS (resp. RHS) by iterating through all the variables in the relation Var and all the integers in Int, and storing, at each iteration, the sum of all the variables/integers

considered so far that occur on the LHS (resp. RHS). These intermediate results are stored using $(l_e + \max(l_v, l_a) + 3l + 1)$-ary predicates $\mathsf{Upto}^{\mathsf{v}}_{\mathsf{L}}$, $\mathsf{Upto}^{\mathsf{v}}_{\mathsf{R}}$, $\mathsf{Upto}^{\mathsf{int}}_{\mathsf{L}}$ and $\mathsf{Upto}^{\mathsf{int}}_{\mathsf{R}}$. Intuitively, $\mathsf{Upto}^{\mathsf{v}}_{\mathsf{L}}$ (resp. $\mathsf{Upto}^{\mathsf{v}}_{\mathsf{R}}$) stores a tuple $(\vec{q}, \vec{v}, \vec{p}, b)$ if and only if the bit with the address $\vec{p}$ in the sum of all variables up to and including the variable $\vec{v}$ that occur on the LHS (resp. on the RHS) of the inequality $\vec{q}$ has the value b. We do this until we reach the very last variable. Once we have summed up the values of all the variables, we do the same for the integers. The relation $\mathsf{Upto}^{\mathsf{int}}_{\mathsf{L}}$ stores a tuple $(\vec{q}, \vec{k}, \vec{p}, b)$, if the bit with the address $\vec{p}$ in the sum of all variables and all integers occurring on the LHS of the inequality $\vec{q}$ up to $\vec{k}$ (including $\vec{k}$) has the value b.

For iterating through the variables and integers, we use the linear order stored in the relation LEQ and we lift it to strings over Cst of length at most $\max(2l_v, 2l_a, 3l)$. We further extract the relations $\mathsf{First}^i, \mathsf{Last}^i, \mathsf{Succ}^i$, $1 \leq i \leq \max(2l_v, 2l_a, 3l)$, that store the least string of length i, the greatest string of length i, and the successor relation on the strings of length i, respectively. We also extract from LEQ a $2l_a$-ary successor relation $\mathsf{Succ}_{\mathsf{int}}$ over the integers in Int. In fact, we have already done this during the construction of $\mathcal{P}_{sys}$. Using this relation, we can further extract the relations $\mathsf{First}_{\mathsf{int}}$ and $\mathsf{Last}_{\mathsf{int}}$ that store, respectively, the first and the last integer with respect to this successor relation. Next, we do the same for the variables. We extract a successor relation on the variables in the $2l_v$-ary relation $\mathsf{Succ}_{\mathsf{V}}$ and we compute the relations $\mathsf{First}_{\mathsf{V}}$ and $\mathsf{Last}_{\mathsf{V}}$, storing the first and the last variable with respect to $\mathsf{Succ}_{\mathsf{V}}$, respectively.

Recall that integers are stored in the relation Int as binary strings of length l_a. To facilitate the computation, we next show how to represent integers in the same way as the values of variables. We introduce a new relation $\mathsf{Val}^{\mathsf{int}}$ that stores a tuple $(\vec{k}, \vec{p}, b)$ if and only if for the integer $\vec{k}$, the bit at the position $\vec{p}$ is set to b. To compute this relation, we rely on the auxiliary relations $\mathsf{Val}^{\mathsf{int}}_i$, for each $1 \leq i \leq l_a$, and $\mathsf{Val}^{\mathsf{int}}_{>l_a}$ that are computed as follows:

$$\mathsf{Val}^{\mathsf{int}}_1(\vec{x}, \vec{z}, x_1) \leftarrow \mathsf{Int}(\vec{x}), \mathsf{First}^{3l}(\vec{z}),$$
$$\mathsf{Val}^{\mathsf{int}}_i(\vec{x}, \vec{z}, x_i) \leftarrow \mathsf{Val}^{\mathsf{int}}_{i-1}(\vec{x}, \vec{z}', x'), \mathsf{Succ}^{3l}(\vec{z}', \vec{z}),$$
$$\mathsf{Val}^{\mathsf{int}}_{>l_a}(\vec{x}, \vec{z}, 0) \leftarrow \mathsf{Val}^{\mathsf{int}}_{l_a}(\vec{x}, \vec{z}', \vec{x}'), \mathsf{LEQ}^{3l}(\vec{z}', \vec{z}),$$

for all $1 < i \leq l_a$, where $\vec{x} = x_{l_a}, \ldots, x_1$. Intuitively, the relation $\mathsf{Val}^{\mathsf{int}}_i$, $1 \leq i \leq l_a$, stores the value of the i-th bit, for every integer in Int. As we only use l_a bits to encode integers, for each integer in Int, we set the value of the remaining bits to 0. This is stored in the relation $\mathsf{Val}^{\mathsf{int}}_{>l_a}$. We then simply copy the information stored in the auxiliary relations into the relation $\mathsf{Val}^{\mathsf{int}}$ using the following rules, for all $1 \leq i \leq l_a$:

$$\mathsf{Val}^{\mathsf{int}}(\vec{x}, \vec{z}, x) \leftarrow \mathsf{Val}^{\mathsf{int}}_i(\vec{x}, \vec{z}, x),$$
$$\mathsf{Val}^{\mathsf{int}}(\vec{x}, \vec{z}, x) \leftarrow \mathsf{Val}^{\mathsf{int}}_{>l_a}(\vec{x}, \vec{z}, x).$$

Finally, we need to add the rules that define the addition of binary numbers. To this end, we use a 5-ary relation Add, where a tuple (b, b', b'', c, r) in Add denotes that the

result of adding bits b, b', and b'' is r with the carry c. As these rules are standard, we omit them here. Further, we use the $(l_e + \max(l_v, l_a) + 3l + 1)$-ary predicates $\mathsf{Carry}^{\mathsf{int}}_{\mathsf{L}}$, $\mathsf{Carry}^{\mathsf{int}}_{\mathsf{R}}$, $\mathsf{Carry}^{\mathsf{v}}_{\mathsf{L}}$, and $\mathsf{Carry}^{\mathsf{v}}_{\mathsf{R}}$ to mark relevant carry bits. A tuple $(\vec{q}, \vec{x}, \vec{p}, c)$ is in $\mathsf{Carry}^{\mathsf{v}}_{\mathsf{L}}$ (resp. $\mathsf{Carry}^{\mathsf{int}}_{\mathsf{L}}$) if when adding the bit at the position $\vec{p}$ of the variable (resp. integer) $\vec{x}$ to the result we have obtained so far for the LHS of the inequality $\vec{q}$, we need to take into account a bit with the value c that was carried over from the previous computation. The meaning of the relation $\mathsf{Carry}^{\mathsf{v}}_{\mathsf{R}}$ and $\mathsf{Carry}^{\mathsf{int}}_{\mathsf{R}}$ is defined analogously.

We are now ready to define the rules that compute $\mathsf{Upto}^{\mathsf{v}}_{\mathsf{L}}$, $\mathsf{Upto}^{\mathsf{v}}_{\mathsf{R}}$, $\mathsf{Upto}^{\mathsf{int}}_{\mathsf{L}}$, and $\mathsf{Upto}^{\mathsf{int}}_{\mathsf{R}}$. For ease of presentation, we assume that there is at least one variable and one integer in the enriched system. Notice that this is not a limitation since we can always have a variable in the system that occurs in no inequalities and thus does not influence the solutions. We begin with the rules for $\mathsf{Upto}^{\mathsf{v}}_{\mathsf{L}}$. For an inequality q in FinIq, we add the rules that initialize the sum to the value of the first variable, if it occurs in q on the LHS, and to 0, otherwise:

$$\mathsf{Upto}^{\mathsf{v}}_{\mathsf{L}}(\vec{y}, \vec{0}, \vec{x}, \vec{z}, 0) \leftarrow \mathsf{FinIq}(\vec{y}), \mathsf{First}_{\mathsf{V}}(\vec{x}), \mathit{not}\ \mathsf{Iq}^{\mathsf{v}}_{\mathsf{L}}(\vec{y}, \vec{x}), \mathsf{Cst}^{3l}(\vec{z}),$$
$$\mathsf{Upto}^{\mathsf{v}}_{\mathsf{L}}(\vec{y}, \vec{0}, \vec{x}, \vec{z}, x) \leftarrow \mathsf{FinIq}(\vec{y}), \mathsf{First}_{\mathsf{V}}(\vec{x}), \mathsf{Val}(\vec{x}, \vec{z}, x), \mathsf{Iq}^{\mathsf{v}}_{\mathsf{L}}(\vec{y}, \vec{x}).$$

At the beginning of each new iteration, there is nothing to carry over from the previous computation, so we add the rules that initialize the carry bit at the first position (i.e, the address of the least significant bit in the value of the considered variable) to 0:

$$\mathsf{Carry}^{\mathsf{v}}_{\mathsf{L}}(\vec{y}, \vec{0}, \vec{x}, \vec{z}, 0) \leftarrow \mathsf{FinIq}(\vec{y}), \mathsf{Var}(\vec{x}), \mathsf{First}^{3l}(\vec{z}).$$

We then add the rules that perform the addition of the sum we have so far and the next variable with respect to $\mathsf{Succ}_{\mathsf{V}}$:

$$\begin{aligned}
\mathsf{Upto}^{\mathsf{v}}_{\mathsf{L}}(\vec{y}, \vec{0}, \vec{x}, \vec{z}, x') &\leftarrow \mathsf{Upto}^{\mathsf{v}}_{\mathsf{L}}(\vec{y}, \vec{0}, \vec{x}', \vec{z}, x'), \mathsf{Succ}_{\mathsf{V}}(\vec{x}', \vec{x}), \mathit{not}\ \mathsf{Iq}^{\mathsf{v}}_{\mathsf{L}}(\vec{y}, \vec{x}),\\
\mathsf{Upto}^{\mathsf{v}}_{\mathsf{L}}(\vec{y}, \vec{0}, \vec{x}, \vec{z}, x) &\leftarrow \mathsf{Upto}^{\mathsf{v}}_{\mathsf{L}}(\vec{y}, \vec{0}, \vec{x}', \vec{z}, x'), \mathsf{Succ}_{\mathsf{V}}(\vec{x}', \vec{x}), \mathsf{Iq}^{\mathsf{v}}_{\mathsf{L}}(\vec{y}, \vec{x}), \mathsf{Val}(\vec{x}, \vec{z}, x''),\\
&\quad \mathsf{Carry}^{\mathsf{v}}_{\mathsf{L}}(\vec{y}, \vec{0}, \vec{x}, \vec{z}, z), \mathsf{Add}(x', x'', z, y, x),\\
\mathsf{Carry}^{\mathsf{v}}_{\mathsf{L}}(\vec{y}, \vec{0}, \vec{x}, \vec{z}', y) &\leftarrow \mathsf{Upto}^{\mathsf{v}}_{\mathsf{L}}(\vec{y}, \vec{0}, \vec{x}', \vec{z}, x'), \mathsf{Succ}_{\mathsf{V}}(\vec{x}', \vec{x}), \mathsf{Iq}^{\mathsf{v}}_{\mathsf{L}}(\vec{y}, \vec{x}), \mathsf{Val}(\vec{x}, \vec{z}, x''),\\
&\quad \mathsf{Succ}^{3l}(\vec{z}, \vec{z}'), \mathsf{Carry}^{\mathsf{v}}_{\mathsf{L}}(\vec{y}, \vec{0}, \vec{x}, \vec{z}, z), \mathsf{Add}(x', x'', z, y, x),
\end{aligned}$$

where $\vec{0}$ is the 0-vector of length $\max(l_v, l_a) - l_v$.

Similarly, the relation $\mathsf{Upto}_{\mathsf{L}}^{\mathsf{int}}$ is computed as follows:

$$\begin{aligned}
\mathsf{Upto}_{\mathsf{L}}^{\mathsf{int}}(\vec{y},\vec{0}_1,\vec{x},\vec{z},x) &\leftarrow \mathsf{FinIq}(\vec{y}),\mathsf{First}_{\mathsf{int}}(\vec{x}),\mathsf{Upto}_{\mathsf{L}}^{\mathsf{v}}(\vec{y},\vec{0}_2,\vec{x}',\vec{z},x),\mathsf{Last}_{\mathsf{V}}(\vec{x}'),\\
&\quad\ \mathit{not}\ \mathsf{Iq}_{\mathsf{L}}^{\mathsf{int}}(\vec{y},\vec{x}),\\
\mathsf{Carry}_{\mathsf{L}}^{\mathsf{int}}(\vec{y},\vec{0},\vec{x},\vec{z},0) &\leftarrow \mathsf{FinIq}(\vec{y}),\mathsf{Int}(\vec{x}),\mathsf{First}^{3l}(\vec{z}),\\
\mathsf{Upto}_{\mathsf{L}}^{\mathsf{int}}(\vec{y},\vec{0}_1,\vec{x},\vec{z},x) &\leftarrow \mathsf{FinIq}(\vec{y}),\mathsf{First}_{\mathsf{int}}(\vec{x}),\mathsf{Iq}_{\mathsf{L}}^{\mathsf{int}}(\vec{y},\vec{x}),\mathsf{Upto}_{\mathsf{L}}^{\mathsf{v}}(\vec{y},\vec{0}_2,\vec{x}',\vec{z},x'),\mathsf{Last}_{\mathsf{V}}(\vec{x}'),\\
&\quad\ \mathsf{Val}^{\mathsf{int}}(\vec{x},\vec{z},x''),\mathsf{Carry}_{\mathsf{L}}^{\mathsf{int}}(\vec{y},\vec{0}_1,\vec{x},\vec{z},z),\mathsf{Add}(x',x'',z,y,x),\\
\mathsf{Carry}_{\mathsf{L}}^{\mathsf{int}}(\vec{y},\vec{0}_1,\vec{x},\vec{z}',y) &\leftarrow \mathsf{FinIq}(\vec{y}),\mathsf{First}_{\mathsf{int}}(\vec{x}),\mathsf{Iq}_{\mathsf{L}}^{\mathsf{int}}(\vec{y},\vec{x}),\mathsf{Upto}_{\mathsf{L}}^{\mathsf{v}}(\vec{y},\vec{0}_2,\vec{x}',\vec{z},x'),\mathsf{Last}_{\mathsf{V}}(\vec{x}'),\\
&\quad\ \mathsf{Val}^{\mathsf{int}}(\vec{x},\vec{z},x''),\mathsf{Succ}^{3l}(\vec{z},\vec{z}'),\mathsf{Carry}_{\mathsf{L}}^{\mathsf{int}}(\vec{y},\vec{0}_1,\vec{x},\vec{z},z),\\
&\quad\ \mathsf{Add}(x',x'',z,y,x),\\
\mathsf{Upto}_{\mathsf{L}}^{\mathsf{int}}(\vec{y},\vec{0}_1,\vec{x},\vec{z},x') &\leftarrow \mathsf{Upto}_{\mathsf{L}}^{\mathsf{int}}(\vec{y},\vec{0}_1,\vec{x}',\vec{z},x'),\mathsf{Succ}_{\mathsf{int}}(\vec{x}',\vec{x}),\mathit{not}\ \mathsf{Iq}_{\mathsf{L}}^{\mathsf{int}}(\vec{y},\vec{x}),\\
\mathsf{Upto}_{\mathsf{L}}^{\mathsf{int}}(\vec{y},\vec{0}_1,\vec{x},\vec{z},x) &\leftarrow \mathsf{Upto}_{\mathsf{L}}^{\mathsf{int}}(\vec{y},\vec{0}_1,\vec{x}',\vec{z},x'),\mathsf{Succ}_{\mathsf{int}}(\vec{x}',\vec{x}),\mathsf{Iq}_{\mathsf{L}}^{\mathsf{int}}(\vec{y},\vec{x}),\mathsf{Val}(\vec{x},\vec{z},x''),\\
&\quad\ \mathsf{Carry}_{\mathsf{L}}^{\mathsf{int}}(\vec{y},\vec{0}_1,\vec{x},\vec{z},z),\mathsf{Add}(x',x'',z,y,x),\\
\mathsf{Carry}_{\mathsf{L}}^{\mathsf{int}}(\vec{y},\vec{0}_1,\vec{x},\vec{z}',y) &\leftarrow \mathsf{Upto}_{\mathsf{L}}^{\mathsf{int}}(\vec{y},\vec{0}_1,\vec{x}',\vec{z},x'),\mathsf{Succ}_{\mathsf{int}}(\vec{x}',\vec{x}),\mathsf{Iq}_{\mathsf{L}}^{\mathsf{int}}(\vec{y},\vec{x}),\mathsf{Val}(\vec{x},\vec{z},x''),\\
&\quad\ \mathsf{Succ}^{3l}(\vec{z},\vec{z}'),\mathsf{Carry}_{\mathsf{L}}^{\mathsf{int}}(\vec{y},\vec{0}_1,\vec{x},\vec{z},z),\mathsf{Add}(x',x'',z,y,x),
\end{aligned}$$

where $\vec{0}_1$ is the 0-vector of the length $l_a - \max(l_v, l_a)$ and $\vec{0}_2$ is the 0-vector of the length $l_v - \max(l_v, l_a)$. The relations $\mathsf{Upto}_{\mathsf{R}}^{\mathsf{v}}$ and $\mathsf{Upto}_{\mathsf{R}}^{\mathsf{int}}$ are computed analogously.

We next store the final result of our computation using the relations LHS and RHS as follows:

$$\begin{aligned}
\mathsf{LHS}(\vec{y},\vec{z},b) &\leftarrow \mathsf{Upto}_{\mathsf{L}}^{\mathsf{int}}(\vec{y},\vec{0},\vec{x},\vec{z},b),\mathsf{Last}_{\mathsf{int}}(\vec{x}),\\
\mathsf{RHS}(\vec{y},\vec{z},b) &\leftarrow \mathsf{Upto}_{\mathsf{R}}^{\mathsf{int}}(\vec{y},\vec{0},\vec{x},\vec{z},b),\mathsf{Last}_{\mathsf{int}}(\vec{x}),
\end{aligned}$$

where $\vec{0}$ is the 0-vector of the length $l_a - \max(l_v, l_a)$.

Recall that we distinguish between general inequalities, stored in the relation Iq, and those inequalities that must be satisfied, stored in the relation Iq^*. We next add the rules that check whether all the inequalities in Iq^* are indeed satisfied by comparing the LHS and RHS of inequalities, bit by bit:

$$\begin{aligned}
\mathsf{Sat}(\vec{y}) &\leftarrow \mathsf{LHS}(\vec{y},\vec{z},c),\mathsf{RHS}(\vec{y},\vec{z},c'),\mathsf{Last}^{3l}(\vec{z}),\mathsf{LEQ}(c,c'),c\neq c',\\
\mathsf{Sat}'(\vec{y},\vec{z}) &\leftarrow \mathsf{LHS}(\vec{y},\vec{z},c),\mathsf{RHS}(\vec{y},\vec{z},c'),\mathsf{Last}^{3l}(\vec{z}),c=c',\\
\mathsf{Sat}(\vec{y}) &\leftarrow \mathsf{LHS}(\vec{y},\vec{z}',c),\mathsf{RHS}(\vec{y},\vec{z}',c'),\mathsf{Sat}'(\vec{y},\vec{z}),\mathsf{Succ}^{3l}(\vec{z}',\vec{z}),\mathsf{LEQ}(c,c'),c\neq c',\\
\mathsf{Sat}'(\vec{y},\vec{z}) &\leftarrow \mathsf{Sat}'(\vec{y},\vec{z}'),\mathsf{Succ}^{3l}(\vec{z},\vec{z}'),\mathsf{LHS}(\vec{y},\vec{z},c),\mathsf{RHS}(\vec{y},\vec{z},c'),c=c',\\
\mathsf{Sat}(\vec{y}) &\leftarrow \mathsf{Sat}'(\vec{y},\vec{z}),\mathsf{First}^{3l}(\vec{z}),\\
&\leftarrow \mathsf{Iq}^*(\vec{y}),\mathit{not}\ \mathsf{Sat}(\vec{y}).
\end{aligned}$$

Note that we do not specifically deal with the case where variables that are set to infinity occur on the LHS but not on the RHS of some inequality. Such an inequality $\vec{y}$ cannot be satisfied and indeed, due to the stable model semantics, we are not able to derive $\mathsf{Sat}(\vec{y})$.

Finally, we finish the construction of $\mathcal{P}_{sol}^{\mathcal{T},\Sigma}$ by adding the constraints that ensure that all the implications are satisfied:

$$\leftarrow \mathsf{Im_L}(\vec{x},\vec{y}), \mathsf{Im_R}(\vec{x},\vec{z}), \mathsf{Iq}(\vec{y}), \mathsf{Iq}(\vec{z}), \mathsf{Sat}(\vec{y}), \mathit{not}\ \mathsf{Sat}(\vec{z}).$$

The construction above makes sure that $\mathcal{P}_{sol}^{\mathcal{T},\Sigma}$ has the following properties. Firstly, just like before, all predicates occurring in $\mathcal{P}_{sol}^{\mathcal{T},\Sigma}$ have an arity that is polynomial in the size of $\mathcal{T}$ and Σ and do not depend on the input ABox. Further, the same also holds for the number of rules in $\mathcal{P}_{sol}^{\mathcal{T},\Sigma}$ – it is polynomial in the size of $\mathcal{T}$ and Σ and data-independent. Now, let $\mathit{Rel}(\mathcal{S})$ be the relational representation of a given enriched system $\mathcal{S}$ using the predicates described at the beginning of this section. If we pass $\mathcal{S}$ to $\mathcal{P}_{sol}^{\mathcal{T},\Sigma}$ as additional facts, the answer sets of this program correspond to the solutions of $\mathcal{S}$ over $\mathbb{N}^*$. Moreover, if there is a solution S over $\mathbb{N}^*$ to $\mathcal{S}$ such that all finite values are bounded by $2^{d^{2l}}$, then there is an answer set of $(\mathcal{P}_{sol}^{\mathcal{T},\Sigma}, \mathit{Rel}(\mathcal{S}))$ that corresponds to this solution. To see this last point, observe that for every variable, $\mathcal{P}_{sol}^{\mathcal{T},\Sigma}$ freely guesses any value between 0 and $2^{d^{2l}}$. The observations above in conjunction with Proposition 3.2.22 give rise to the following proposition:

Proposition 4.1.2. *Given an enriched system $\mathcal{S}$ with the relational representation $\mathit{Rel}(\mathcal{S})$, $\mathcal{S}$ has a solution over $\mathbb{N}^*$ if and only if the program $(\mathcal{P}_{sys}^{\mathcal{T},\Sigma}, \mathit{Rel}(\mathcal{S}))$ has an answer set. Moreover, $\mathcal{P}_{sol}^{\mathcal{T},\Sigma}$ is polynomial in the size of $\mathcal{T}$ and Σ.*

Recall that every answer set of the program $(\mathcal{P}_{sys}^{\mathcal{T},\Sigma}, \hat{\mathcal{A}})$ corresponds to the relational representation of $\mathcal{S}_{\mathcal{K}}$, for a given ABox $\mathcal{A}$ and $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$. Thus, in view of Proposition 4.1.2, we have the following result:

Proposition 4.1.3. *Given an input ABox $\mathcal{A}$ over the signature of $\mathcal{T}$, let $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$. The program $\mathcal{P}_{sol}^{\mathcal{T},\Sigma}$ is polynomial in the size of $\mathcal{T}$ and Σ and $\mathcal{S}_{\mathcal{K}}$ has a solution over $\mathbb{N}^*$ iff there is an answer set I of $(\mathcal{P}_{sys}^{\mathcal{T},\Sigma}, \hat{\mathcal{A}})$ such that $(\mathcal{P}_{sol}^{\mathcal{T},\Sigma}, I)$ has an answer set.*

Theorem 4.1.4. *For a TBox $\mathcal{T}$ and $\Sigma \subseteq \mathsf{N_C} \cup \mathsf{N_R}$, we can obtain a program $\mathcal{P}_{sat}^{\mathcal{T},\Sigma}$ in polynomial time such that $(\mathcal{P}_{sat}^{\mathcal{T},\Sigma}, \hat{\mathcal{A}})$ has a stable model if and only if $(\mathcal{T}, \Sigma, \mathcal{A})$ is satisfiable, for all ABoxes $\mathcal{A}$ over the signature of $\mathcal{T}$.*

Proof. The program $\mathcal{P}_{sat}^{\mathcal{T},\Sigma}$ is simply the union of $\mathcal{P}_{sys}^{\mathcal{T},\Sigma}$ and $\mathcal{P}_{sol}^{\mathcal{T},\Sigma}$, both of which can be obtained in polynomial time from $\mathcal{T}$ and Σ (Propositions 4.1.1 and 4.1.3). Moreover, all constants of $\mathcal{P}_{sol}^{\mathcal{T},\Sigma}$ occur also in $\mathcal{P}_{sys}^{\mathcal{T},\Sigma}$, the predicates occurring in $\hat{\mathcal{A}}$ are EDB predicates of both $\mathcal{P}_{sys}^{\mathcal{T},\Sigma}$ and $\mathcal{P}_{sol}^{\mathcal{T},\Sigma}$, and all shared predicates of $\mathcal{P}_{sys}^{\mathcal{T},\Sigma}$ and $\mathcal{P}_{sol}^{\mathcal{T},\Sigma}$ are EDB predicates in $\mathcal{P}_{sol}^{\mathcal{T},\Sigma}$. Due to Proposition 2.4.12, the answer sets of $(\mathcal{P}_{sat}^{\mathcal{T},\Sigma}, \hat{\mathcal{A}}) = \mathcal{P}_{sat}^{\mathcal{T},\Sigma} \cup \hat{\mathcal{A}}$ correspond to

the set $\{I : I$ is an answer set of $\mathcal{P}_{sol}^{\mathcal{T},\Sigma} \cup J$, for some answer set J of $\mathcal{P}_{sys}^{\mathcal{T},\Sigma} \cup \hat{\mathcal{A}}\}$. Finally, combining this with the result from Proposition 4.1.3, we get that $(\mathcal{T}, \Sigma, \mathcal{A})$ is satisfiable, if $\mathcal{P}_{sat}^{\mathcal{T},\Sigma} \cup \hat{\mathcal{A}} = (\mathcal{P}_{sat}^{\mathcal{T},\Sigma}, \hat{\mathcal{A}})$ has an answer set. □

4.2 Query Rewriting and Complexity

In this section, we formally introduce the notion of ontology-mediated queries in the presence of closed predicates and we present a large class of such queries that can be rewritten into Datalog$^{\neg}$.

4.2.1 Ontology-Mediated Queries with Closed Predicates

As expected, an ontology-mediated query with closed predicates is simply an ordinary ontology-mediated query augmented with a set Σ of predicates whose extensions are to be interpreted under the closed-world assumption.

Definition 4.2.1. *An* ontology-mediated query with closed predicates *is a triple* $Q = (\mathcal{T}, \Sigma, q)$*, where* $\mathcal{T}$ *is a TBox,* $\Sigma \subseteq \mathsf{N_C} \cup \mathsf{N_R}$ *is a set of closed predicates and* q *is a first-order query over* $\mathsf{N_C} \cup \mathsf{N_R}$*.*

As before, if it is clear that we are in a setting that involves closed predicates, we refer to these queries simply as *ontology-mediated queries*, or *OMQ* for short. All standard notions relating to query answering in description logics also carry over to OMQs with closed predicates.

Definition 4.2.2. *Given an ABox* $\mathcal{A}$*, a tuple of constants* $\vec{a} = (a_1, \ldots, a_n)$ *from* $\mathsf{N_I}(\mathcal{A}) \cup \mathsf{N_I}(\mathcal{T})$ *is a* certain answer *to* Q *over* $\mathcal{A}$*, if* $\vec{a}$ *is an answer to* q *in every model* $\mathcal{I}$ *of* $(\mathcal{T}, \Sigma, \mathcal{A})$*. We denote the set of certain answers to* Q *over* $\mathcal{A}$ *by* $cert(Q, \mathcal{A})$*. The* query answering problem *is the problem of deciding, given an OMQ* Q*, an ABox* $\mathcal{A}$*, and a tuple of constants* $\vec{a}$*, whether* $\vec{a} \in cert(Q, \mathcal{A})$*.*

One of the most important effects that the closed predicates have on query answering is that setting they make OMQs *non-monotonic*, which is not the case in the standard setting. This means that adding new facts to the ABox may destroy previous answers, i.e., a tuple $\vec{a}$ that was a certain answer to some OMQ Q over an ABox $\mathcal{A}$ may no longer be a certain answer to Q over some ABox $\mathcal{A}'$ that extends $\mathcal{A}$. To illustrate the additional expressive power that closed predicates bring with them, we consider the following simple examples.

Example 4.2.3. *Let* $Q = (\mathcal{T}, \{B\}, q(x))$ *be an OMQ, where*

$$\mathcal{T} = \{A \sqsubseteq \exists r.B, C \sqsubseteq \exists r.B, A \sqcap C \sqsubseteq \bot\}$$
$$q(x) = \exists y \exists z.r(y, x) \wedge r(z, x) \wedge A(y) \wedge C(z)$$

Observe that here $\mathcal{T}$ is an $\mathcal{ALC}$ TBox. Consider an ABox $\mathcal{A} = \{A(a), B(b), C(c)\}$. Since B is closed, $B^{\mathcal{I}} = \{b\}$, for every model $\mathcal{I}$ of $(\mathcal{T}, \{B\}, \mathcal{A})$. According to $\mathcal{T}$, in every model $\mathcal{I}$ of $(\mathcal{T}, \{B\}, \mathcal{A})$, both a and c need an r-successor that is in $B^{\mathcal{I}}$, and so $(a, b) \in r^{\mathcal{I}}$ and $(c, b)^{\mathcal{I}}$. Hence, b is a certain answer to Q over $\mathcal{A}$.

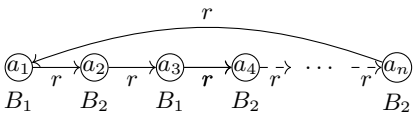
To see that the closed predicates indeed have an effect on reasoning, consider the following OMQ $Q' = (\mathcal{T}, \emptyset, q(x))$ where the concept name B is considered open. Q' has no certain answers over $\mathcal{A}$.

Further, closed predicates make OMQs non-monotonic. For example, b is no longer a certain answer to Q over $\mathcal{A}' = \mathcal{A} \cup \{B(d)\}$.

Example 4.2.4. *Recall the parity query Q_{parity} that asks whether an ABox $\mathcal{A}$ has an odd number of objects in some unary relation A. We can easily express Q_{parity} as a Boolean OMQ with closed predicates $(\mathcal{T}, \Sigma, \bot)$, where the TBox $\mathcal{T}$ consists of the following axioms:*

$$A \equiv B_1 \sqcup B_2 \quad B_1 \sqcap B_2 \sqsubseteq \bot \quad B_1 \sqsubseteq \geq 1r.B_2 \quad B_2 \sqsubseteq \geq 1r.B_1 \quad \top \sqsubseteq \leq 1r.\top \quad \top \sqsubseteq \leq 1r^{-}.\top$$

and $\Sigma = \{A\}$ is the set of predicates that are considered closed. Let $a_1, \ldots, a_n$ be the elements in A, as given by some ABox $\mathcal{A}$. The axioms in $\mathcal{T}$ force models to partition $a_1, \ldots, a_n$ into two sets B_1 and B_2. Further, every element in B_1 has an r-arc to an element in B_2 and vice versa. Due to the functionality of r and r^{-}, which is encoded using the last two axioms of the TBox, each element has exactly one outgoing and at most one ingoing r-arc. As A is a closed predicate, this forces the existence of a cycle in models of $\mathcal{T}$ and $\mathcal{A}$ of the following form:



Such a cycle exists if and only if n is even, which means that for odd n, the $\bot$ is entailed. We call $(\mathcal{T}, \Sigma, \bot)$ is called an incon

Other examples of non-monotonic queries that can be easily expressed using $\mathcal{ALCHOIQ}$ with closed predicates include queries that compare cardinalities of two database relations, e.g., checking whether there are twice as many elements in relation B than in relation A.

For ease of presentation, we note that we only consider OMQs $Q = (\mathcal{T}, \Sigma, q)$ such that all predicates from Σ occur in $\mathcal{T}$, and q is a constant-free query over the predicates occurring in $\mathcal{T}$. Notice that none of these conditions is a true limitation. In particular, if a predicate p occurs in Σ or q and not $\mathcal{T}$, we can simply add an axiom $p \sqsubseteq p$ to $\mathcal{T}$. Further, we can easily simulate a constant c that occurs in q but not in $\mathcal{T}$ using fresh concept names. Let $A_c \in \mathsf{N_C}$ be a fresh concept name and $x_c \in \mathsf{N_V}$ be a fresh variable. Let $\mathcal{T}' = \mathcal{T} \cup \{\{c\} \sqsubseteq A_c\}$ and $q' = q[x_c/c] \wedge A_c(x_c)$. Then the certain answers to $Q' = (\mathcal{T}', \Sigma, q')$ coincide with those of Q, for any ABox $\mathcal{A}$ over the signature of $\mathcal{T}$.

In the remainder of this chapter, we show that we can rewrite a large class of OMQs expressed in $\mathcal{ALCHOIQ}$ with closed predicates into Datalog$^\neg$ in polynomial time. We first clarify what exactly we mean when by *rewriting*.

Definition 4.2.5. *Let $Q = (\mathcal{T}, \Sigma, q(\vec{x}))$ be an OMQ. A Datalog$^\neg$ query $(\mathcal{P}, P)$ is said to be a* Datalog$^\neg$-rewriting *(or a* translation*) of Q if*

$$cert((\mathcal{P}, P), \hat{\mathcal{A}}) = cert(Q, \mathcal{A}),$$

for every ABox $\mathcal{A}$ over the signature of $\mathcal{T}$. Recall that $\hat{\mathcal{A}}$ is obtained from $\mathcal{A}$ by replacing each negative assertion $\neg p(\vec{a})$ with $\bar{p}(\vec{a})$. Moreover, if such a Datalog$^\neg$ query exists, we say that Q is Datalog$^\neg$-rewritable*. A class of OMQs is called* Datalog$^\neg$-rewritable *if every query in this class is Datalog$^\neg$-rewritable.*

4.2.2 Safe-Range OMQs

Recall that, other than decidability, so far there are no upper bounds on the complexity of answering even very simple FO queries, such as conjunctive queries, mediated by $\mathcal{ALCHOIQ}$ ontologies. Therefore, we consider a fragment of FO queries mediated by $\mathcal{ALCHOIQ}$ ontologies with closed predicates for which we can provide some complexity guarantees. In the context of relational databases, one of the most commonly required properties of FO queries is *domain independence* which ensures that queries have a well-defined meaning and their answers do not depend on the objects outside of the database. As this property is generally undecidable for FO logic, the usual way of ensuring domain independence is to consider *safe-range FO queries* whose variables are guaranteed to range only over the constants in the database, achieved by guarding each variable by a positive atom over some database predicate (in traditional databases each predicate is considered closed). Inspired by this approach, we next introduce *ontology-mediated safe-range queries* and we provide Datalog$^\neg$-rewritability and tight complexity results for the case when TBoxes are written in $\mathcal{ALCHOIQ}$. In a nutshell, a safe-range OMQ $Q = (\mathcal{T}, \Sigma, q)$ is an OMQ for which we can guarantee that the variables of q range only over known objects, i.e., those constants that occur in either $\mathcal{T}$ or $\mathcal{A}$, where $\mathcal{A}$ is the ABox over which Q is being answered. We do this by placing a syntactic restriction on the queries that demands each non-answer variable be guarded by a positive atom over some closed predicate. As a result, safe-range OMQs have the following convenient property: in order to answer a safe-range OMQ $Q = (\mathcal{T}, \Sigma, q)$ over some ABox $\mathcal{A}$, it suffices to answer q over all completions of $\mathcal{A}$ with atoms over the predicates in $\mathcal{T}$ and the constants in $\mathcal{A}$ and $\mathcal{T}$ that are consistent with $\mathcal{T}$ and Σ. For the case where $\mathcal{T}$ is an $\mathcal{ALCHOIQ}$ TBox, this property can be exploited to compute a succinct Datalog$^\neg$-rewriting, which in turn gives us the co-NP data complexity upper bound for the query answering problem. As answering even the simplest queries mediated by $\mathcal{ALCHOIQ}$ TBoxes is known to be co-NP hard, the obtained complexity result is tight. We note that our safe-range OMQs are close in spirit to many existing approaches that, in one way or another, restrict certain variables to known individuals. For example, similar restrictions, such as the well-known *DL-safety* criterion [MSS05] and its variations, are commonly used to ensure

decidability when combining DLs and Datalog rules into hybrid knowledge bases. Some further examples of languages that ensure certain variables are bound only to known individuals, although for different purposes, include *description logics with nominal schemata* [KMKH11, KR14] and *existential rules with closed variables* [ALMV18].

We begin by formally defining safe-range OMQs. Much like the traditional approach for safe-range FO queries, we provide the following multi-step syntactic characterization of safe-range OMQs:

1. the procedure SRNF for transforming a FO query q into a logically equivalent query that is in *safe-range normal form (SRNF)* and is therefore more suitable for safety analysis,
2. the procedure rr that takes a FO query q in SRNF and a set Σ of predicates and checks whether all quantified variables of q are Σ-range restricted, i.e., guarded by positive atoms over predicates from Σ. If this is the case, the procedure returns a subset of the free variables in q that are Σ-range restricted, otherwise, it returns 'fail', and
3. a global property that an OMQ Q must satisfy in order to be considered safe-range.

Regarding the first item, we recall the standard procedure in the literature (see [AHV95]) that takes as input an arbitrary FO query q and performs the following equivalence preserving syntactic transformations to place q into SRNF :

Rename variables: Rename variables such that no distinct pair of quantifiers binds the same variable and no variable occurs both free and bound

Eliminate universal quantifiers: Replace $\forall \vec{x} \psi$ by $\neg \exists \vec{x} \neg \psi$.

Eliminate implications and equivalences: Replace $\psi \to \xi$ by $\neg \psi \vee \xi$ and similarly for $\leftrightarrow$.

Push negations: Replace

1. $\neg\neg\psi$ by ψ,
2. $\neg(\psi_1 \wedge \cdots \wedge \psi_n)$ by $(\neg\psi_1 \vee \cdots \vee \neg\psi_n)$, and
3. $\neg(\psi_1 \vee \cdots \vee \psi_n)$ by $(\neg\psi_1 \wedge \cdots \wedge \neg\psi_n)$.

so that for every subformula $\neg\psi$, ψ is either an atom or an existentially quantified formula.

Flatten the formula so that no child of $\wedge$ (resp. $\vee/\exists$) in the syntax tree of the formula is an $\wedge$ (resp. $\vee/\exists$).

We denote the resulting formula by $\mathsf{SRNF}(q)$.

Next, Algorithm 4.1 presents the procedure rr that takes as input a FO query q in SRNF and a set of predicates Σ and recursively computes the set of variables that are Σ-range-restricted. Simply put, if a variable x is Σ-range-restricted in q, then we can be sure that when evaluating q over some interpretation, x can only take as values constants that occur in the extensions of the predicates from Σ. If at any point the procedure discovers that some existentially quantified variable is not Σ-range-restricted, it immediately rejects the query by returning *'fail'*. Note that this procedure is an adaptation of the procedure rr presented in [AHV95] for computing range-restricted variables of FO queries. In particular, for a constant-free FO query q, if Σ contains all predicates occurring in q, i.e., all relevant predicates are considered closed, then the two procedures coincide.

We are now finally ready to give a formal definition of safe-range OMQs.

Definition 4.2.6. *An OMQ $Q = (\mathcal{T}, \Sigma, q)$ is* safe-range *if $\mathsf{rr}(\mathsf{SRNF}(q), \Sigma) \neq \text{'fail'}$.*

Example 4.2.7. *Consider again the OMQ $Q = (\mathcal{T}, \Sigma, q(x))$ from Example 4.2.3, where*

$$\begin{aligned} \mathcal{T} &= \{A \sqsubseteq \exists r.B, C \sqsubseteq \exists r.B, A \sqcap C \sqsubseteq \bot\}, \\ \Sigma &= \{B\}, \\ q(x) &= \exists y \exists z.r(y,x) \land r(z,x) \land A(y) \land C(z). \end{aligned}$$

This query is not safe-range since the existentially quantified variables y and z do not occur in positive atoms over closed predicates (i.e., B) and are therefore not recognized as Σ-range-restricted, so the procedure $\mathsf{rr}(\mathsf{SRNF}(q), \Sigma)$ returns 'fail'.

In contrast, the OMQ $Q' = (\mathcal{T}, \{B, r\}, q(x))$ is safe-range, as the role r is considered closed and can be used as a guard for y and z.

In simple terms, an OMQ is safe-range if all its non-answer variables are guarded by closed predicates. To understand the intuition behind this definition, consider an OMQ $Q = (\mathcal{T}, \Sigma, q)$ and an arbitrary ABox $\mathcal{A}$ over the signature of $\mathcal{T}$. Recall that, by definition, only the constants from $\mathcal{A}$ and $\mathcal{T}$ can occur in certain answers of Q over $\mathcal{A}$. Therefore, the range of the answer variables in q is already implicitly restricted to known constants only. On the other hand, during query answering, a quantified variable x of q is free to take any value, including anonymous domain elements, which means that considering only those cases where x is a constant from $\mathcal{T}$ or $\mathcal{A}$ may yield wrong results. However, since the predicates in Σ are considered closed and their extensions are fully specified by the ABox $\mathcal{A}$, guarding x by a positive atom over a predicate from Σ ensures that x is mapped to a known constant. With this in mind, it is easy to see that Q being safe-range has the following effect: for any model $\mathcal{I}$ of $(\mathcal{T}, \Sigma, \mathcal{A})$, $\vec{a}$ is an answer to q in $\mathcal{I}$ if and only if it is an answer to q in $\mathcal{I}$ restricted to only those constants that appear in $\mathcal{A}$ and $\mathcal{T}$. Thus, safe-range OMQs can be answered over consistent completions of the data.

Definition 4.2.8. *Given a TBox $\mathcal{T}$ and a set of closed predicates Σ, we say that an ABox $\mathcal{A}'$ is a* completion of $\mathcal{A}$ w.r.t $\mathcal{T}$ and Σ *if the following holds:*

- $\mathcal{A} \subseteq \mathcal{A}'$,
- *for each $a \in \mathsf{N_I}(\mathcal{T}) \cup \mathsf{N_I}(\mathcal{A})$ and concept name $C \in \mathsf{N_C}(\mathcal{T})$, either $C(a) \in \mathcal{A}'$ or $\neg C(a) \in \mathcal{A}'$,*
- *for each $\{a, b\} \subseteq \mathsf{N_I}(\mathcal{T}) \cup \mathsf{N_I}(\mathcal{A})$ and role name $r \in \mathsf{N_R}(\mathcal{T})$, either $r(a, b) \in \mathcal{A}'$ or $\neg r(a, b) \in \mathcal{A}'$,*
- *$C(a) \in \mathcal{A}'$ and $C \in \Sigma$ implies $C(a) \in A$, and*
- *$r(a, b) \in \mathcal{A}'$ and $r \in \Sigma$ implies $r(a, b) \in \mathcal{A}$.*

We say that $\mathcal{A}'$ is a consistent completion of $\mathcal{A}$ w.r.t. $\mathcal{T}$ and Σ *if $\mathcal{A}'$ is a completion of $\mathcal{A}$ w.r.t. $\mathcal{T}$ and Σ, and $(\mathcal{T}, \Sigma, \mathcal{A}')$ is satisfiable.*

Proposition 4.2.9. *Let $Q = (\mathcal{T}, \Sigma, q(x_1, \ldots, x_n))$ be a safe-range OMQ. A tuple of constants $\vec{a} = (a_1, \ldots, a_n)$ is a certain answer to Q over an ABox $\mathcal{A}$ iff $\mathcal{A}' \models q[a_1/x_1 \ldots, a_n/x_n]$, for every consistent completion $\mathcal{A}'$ of $\mathcal{A}$ w.r.t. $\mathcal{T}$ and Σ.*

The proposition above already hints at a strategy for obtaining the desired rewriting. Namely, given a safe-range OMQ $Q = (\mathcal{T}, \Sigma, q)$, it is enough to (i) compute a $\mathsf{Datalog}^\neg$ program $\mathcal{P}_{OMQ}^{\mathcal{T},\Sigma}$ whose stable models over the input ABox $\mathcal{A}$ represent consistent completions of $\mathcal{A}$ w.r.t. $\mathcal{T}$ and Σ, for any ABox $\mathcal{A}$ over the signature of $\mathcal{T}$ and (ii) rewrite the FO query q as $\mathsf{Datalog}^\neg$ query $(\mathcal{P}_q, p_q)$.

Regarding (ii), it is well known that a FO query q that is safe-range in a traditional sense can be written as a $\mathsf{Datalog}^\neg$ query $(\mathcal{P}_q, p_q)$, where $\mathcal{P}_q$ is a $\mathsf{Datalog}^\neg$ program that is polynomial in the size of q and that, given a set of ground atoms I, computes the answers to q over I and stores them in the relation p_q. We note that the program $\mathcal{P}_q$ uses only stratified negation, i.e., there are no cyclic dependencies between predicates that involve negation. It is a well-known fact that stratified programs have exactly one stable model [ABW88]. Thus, we have that $\vec{a}$ is an answer to q over I if and only if $p_q(\vec{a})$ occurs in the stable model of $\mathcal{P}_q \cup I$. Now, given an arbitrary safe-range OMQ $Q = (\mathcal{T}, \Sigma, q(x_1, \ldots, x_n))$, the FO query $q(x_1, \ldots, x_n)$ might not satisfy the syntactic criterion that makes it safe-range in the traditional sense, because that additionally requires that each $x_1, \ldots, x_n$ is guarded by some positive atom in q. Note that safe-range OMQs do not place this restriction, as the semantics of OMQs already defines answers to be only over the set of known individuals. To overcome this issue and reuse the rewriting procedure from the literature, we consider the query $q'(x_1, \ldots, x_n) = q(x_1, \ldots, x_n) \wedge \bigwedge_{i=1,\ldots n} \mathsf{Adom}(x_i)$, where Adom is a predicate whose extension consists of constants that occur in $\mathcal{T}$ and $\mathcal{A}$ that is equivalent to q when evaluated over consistent completions of $\mathcal{A}$ w.r.t. $\mathcal{T}$ and Σ, for any ABox $\mathcal{A}$ over the signature of $\mathcal{T}$. As this query is safe-range in the traditional sense, it can be rewritten into $\mathsf{Datalog}^\neg$ and so can $q(x_1, \ldots, x_n)$.

To deal with (i), consider the program $\mathcal{P}_{sat}^{\mathcal{T},\Sigma}$ from the previous section and let $\mathcal{A}$ be an input ABox. Even though each stable model of $(\mathcal{P}_{sat}^{\mathcal{T},\Sigma}, \hat{\mathcal{A}})$ implicitly corresponds to a consistent completion of $\mathcal{A}$ and vice versa, due to the stable model semantics, this program does not actually infer any new atoms over the signature of the TBox and can therefore not be directly used for query answering. We next define a program $\mathcal{P}_{cmpl}^{\mathcal{T},\Sigma}$ that, for an input ABox $\mathcal{A}$ over the signature of $\mathcal{T}$, generates a completion of $\mathcal{A}$ w.r.t. $\mathcal{T}$ and Σ. To this end, for all $A \in \mathsf{N_C}(\mathcal{T}) \setminus \Sigma$, $B \in \mathsf{N_C}(\mathcal{T})$, $r \in \mathsf{N_R}(\mathcal{T}) \setminus \Sigma$ and $s \in \mathsf{N_R}(\mathcal{T})$, we add the following rules to $\mathcal{P}_{cmpl}^{\mathcal{T},\Sigma}$:

$$
\begin{aligned}
A(x) &\leftarrow \mathsf{Adom}(x), \mathit{not}\ \overline{A}(x),\\
\overline{B}(x) &\leftarrow \mathsf{Adom}(x), \mathit{not}\ B(x),\\
r(x,y) &\leftarrow \mathsf{Adom}(x), \mathsf{Adom}(y), \mathit{not}\ \overline{r}(x,y),\\
\overline{s}(x,y) &\leftarrow \mathsf{Adom}(x), \mathsf{Adom}(y), \mathit{not}\ s(x,y).
\end{aligned}
$$

Let $\mathcal{P}_{OMQ}^{\mathcal{T},\Sigma} = \mathcal{P}_{sat}^{\mathcal{T},\Sigma} \cup \mathcal{P}_{cmpl}^{\mathcal{T},\Sigma}$. It is easy to see that $\mathcal{A}'$ is a consistent completion of $\mathcal{A}$ w.r.t. $\mathcal{T}$ and Σ iff there is a stable model I of $(\mathcal{P}_{OMQ}^{\mathcal{T},\Sigma}, \hat{\mathcal{A}})$ with $\hat{\mathcal{A}}' \subseteq I$, for any ABox $\mathcal{A}$ over the signature of $\mathcal{T}$. Note that $\mathcal{P}_{OMQ}^{\mathcal{T},\Sigma}$ is obtained from $\mathcal{T}$ and Σ in polynomial time, which leads to the following result.

Theorem 4.2.10. *Let $Q = (\mathcal{T}, \Sigma, q)$ be a safe-range OMQ. We can obtain in polynomial time a program $\mathcal{P}_{OMQ}^{\mathcal{T},\Sigma}$ from $\mathcal{T}$ and Σ such that the certain answers to Q over $\mathcal{A}$ coincide with the certain answers of $(\mathcal{P}_{OMQ}^{\mathcal{T},\Sigma} \cup \mathcal{P}_q, p_q)$ over $\hat{\mathcal{A}}$, for any ABox $\mathcal{A}$ over the signature of $\mathcal{T}$, where $(\mathcal{P}_q, p_q)$ is a $\mathsf{Datalog}^{\neg}$- rewriting of q. In other words, safe-range queries mediated by $\mathcal{ALCHOIQ}$ ontologies with closed predicates are $\mathsf{Datalog}^{\neg}$-rewritable.*

The previous theorem allows us to infer data complexity upper bound. To obtain tight complexity results, observe that safe-range OMQs subsume the class of OMQs whose associated FO queries are *instance queries (IQ)*, i.e., they consist simply of a first-order atom.

Theorem 4.2.11 ([Sch93])**.** *Answering ontology-mediated IQs is coNP-complete in data complexity for $\mathcal{ALC}$ even without closed predicates.*

Theorem 4.2.12. *The query answering problem for safe-range OMQs is coNP-complete in data complexity for $\mathcal{ALCHOIQ}$.*

Proof sketch. Checking whether a tuple of constants is a certain answer to a $\mathsf{Datalog}^{\neg}$ query is coNP-complete in terms of data complexity (see Table 2.3 and also [DEGV01]). As the obtained query does not depend on $\mathcal{A}$, and $\hat{\mathcal{A}}$ is obtained from $\mathcal{A}$ in polynomial time, we get the desired upper data complexity bound. The matching lower bound comes from Theorem 4.2.11. □

As a final remark, we note that safe-range OMQs also subsume the recently-studied *ontology-mediated unions of quantifier-free conjunctive queries with closed predicates (qfUCQs)*[LSW19]. We can thus transfer our rewritability and complexity results to IQs and qfUCQs mediated by $\mathcal{ALCHOIQ}$ ontologies with closed predicates.

Theorem 4.2.13. *There is a polynomial rewriting of IQs and qfUCQs mediated by $\mathcal{ALCHOIQ}$ ontologies with closed predicates into* Datalog$^{\neg}$.

Theorem 4.2.14. *The query answering problem for IQs and qfUCQs mediated by $\mathcal{ALCHOIQ}$ ontologies with closed predicates is* coNP*-complete in data complexity.*

4.3 Discussion

In this chapter, we presented a translation of $\mathcal{ALCHOIQ}$ with closed predicates into Datalog with negation under the stable model semantics. Our translation uses a very different approach from all other translations in the literature and it is based on a characterization of the satisfiability problem for this logic as a system of linear inequalities with some side conditions. Relying on the integer programming characterization from Chapter 3, given a TBox $\mathcal{T}$ and a set of closed predicates Σ, we first showed how to construct in polynomial time a program for deciding, given an input ABox $\mathcal{A}$, whether the KB $(\mathcal{T}, \Sigma, \mathcal{A})$ is satisfiable. As stated in the discussion of the previous chapter, our approach assumes unary encoding of the numerical values that occur in the given TBox. In particular, if binary encoding is assumed, the size of tiles used in the characterization of the satisfiability problem for some KB $\mathcal{K}$ becomes exponential in the size of $\mathcal{K}$, meaning that our translation is no longer polynomial. At this moment, it remains unclear how our technique could be adapted to also provide a polynomial translation in the case of binary encoding, and is something that needs further investigation. We also remark that our translation hinges on the availability of two distinct constants, 0 and 1, which are always present in the obtained Datalog$^{\neg}$ program. This is not surprising, since, as argued in [AOŠ20], under the standard complexity assumptions, even plain $\mathcal{ALC}$ TBoxes (i.e., those without closed predicates) cannot be rewritten to Datalog$^{\neg}$ if constants are disallowed from appearing in the rules of the obtained program.

We next showed how to further extend the obtained program to answer safe-range OMQs which subsume popular classes of OMQs like ontology-mediated instance queries and quantifier-free unions of conjunctive queries. Even though our results were obtained for $\mathcal{ALCHOIQ}$, we can use the existing results in the literature for eliminating transitivity axioms (see, e.g., [HMS07]) to lift our rewritability and complexity results to $\mathcal{SHOIQ}$ with closed predicates, provided that the queries are formulated only over concepts and simple role names. An important by-product of our polynomial translation, we obtained a complexity result stating that answering these queries is coNP-complete in data complexity.

Our approach already covers a very large class of OMQs, namely the safe-range OMQs. Going above this class would be difficult as it is known that first-order queries quickly

become undecidable, even for very basic extensions of conjunctive queries (CQs) and very lightweight DLs [GBIGKK15]. Also considering conjunctive queries is not a viable option, since it is known that $\mathcal{ALCOIF}$-mediated CQs are co-N2ExpTime-hard even in the absence of closed predicates [GKL11]. Thus, under standard complexity assumptions, there cannot exist a polynomial translation of $\mathcal{ALCHOIQ}$ mediated CQs into Datalog with stable negation. In fact, answering CQs in expressive description logics with nominals, inverses, and number restrictions is a long-standing open problem in the field of DLs for which the only known upper bound is decidability. Although not immediately tied to our goal of providing polynomial time rewritings, in our future work we plan to try to investigate the complexity of this problem.

As a final remark, we note that the considered variant of Datalog underlies Answer Set Programming (ASP), which is a very mature area and many efficient reasoning engines for this rule language exist. While our polynomial time translation is unlikely to yield an efficient tool for reasoning with $\mathcal{ALCHOIQ}$ ontologies, it nevertheless draws an important new connection between DLs and ASP.

Relation	Arity	Meaning
A, $\overline{A}$, for $A \in \mathsf{N_C}(\mathcal{T})$	1	A or $\neg A$, resp., for $A \in \mathsf{N_C}(\mathcal{T})$
r, $\overline{r}$, for $r \in \mathsf{N_R^+}(\mathcal{T})$	2	role r or $\neg r$, resp., for $r \in \mathsf{N_R^+}(\mathcal{T})$
Bin	1	constants 0 and 1
Int	l_{int}	integers using standard binary encoding
Adom	1	constants in $\mathcal{K}$
Adom*	1	constants in $\mathcal{K}$ and a special constant $*$
Cst	1	all constants in $\mathcal{P}_{sys}(\mathcal{K})$, i.e., constants in Bin and Int
RType	$k_\mathcal{T}$	role types for $\mathcal{K}$
Type	$n_\mathcal{T}+1$	types for $\mathcal{K}$
Triple	$n_\mathcal{T}+1+k_\mathcal{T}+l_{int}$	triples (R,T,k), where T is a type, R is a role type and k is an integer
LEQ^i, for $i=1,\dots,l_{tile}$	$2i$	linear order over strings Cst of length i
Tile	l_{tile}	tiles for $\mathcal{K}$
CandT	l_{tile}	candidate tiles for $\mathcal{K}$,
InvCandT	l_{tile}	syntactically same as candidate tiles but triples for $\mathcal{K}$ that are out of order
BadTile	l_{tile}	candidate tiles for $\mathcal{K}$ that violate some condition in Def 3.2.5
In_{B_i}, for $i=1,\dots,n_\mathcal{T}$	$n_\mathcal{T}+1$	types for $\mathcal{K}$ that contain concept name B_i
In_{r_i}, for $i=1,\dots,k_\mathcal{T}$	$k_\mathcal{T}$	role types for $\mathcal{K}$ that contain role r_i
$\mathsf{Succ_{int}}$	$2l_{int}$	successor relation over integers w.r.t. LEQ
$\overline{\mathsf{Succ_{int}}}$	$2l_{int}$	complement of $\mathsf{Succ_{int}}$
OK_i^1, for $i=1,\dots,n$	l_{tile}	candidate tiles whose i-th triple (R,T',k) satisfies T1 in Definition 3.2.5
OK_i^2, for $i=1,\dots,n$	l_{tile}	candidate tiles whose i-th triple (R,T',k) satisfies T2 in Definition 3.2.5
$\mathsf{Upto}_{i,A\sqsubseteq=nr.B\in\mathcal{T}}$, for $i=1,\dots,n$	$l_{tile}+l_{int}$	pairs (τ,z) where $\tau=(T,\rho)$ is a tile for $\mathcal{K}$ s.t. $A\in T$, z is an integer, and among the first i triples of ρ there are z triples (R,T',k) for which $r\in R$ and $B\in T'$
Invrt	$2(n_\mathcal{T}+1)+k_\mathcal{T}$	invertible triples (T,R,T'), where T,T' are types, and R is a role type for $\mathcal{K}$
RType^-	$2k_\mathcal{T}$	pairs of role types (R,R^-)
$\overline{\mathsf{RType}^-}$	$2k_\mathcal{T}$	complement of RType^-
Upto_i, for $i=1,\dots,n$	$l_{tile}+n_\mathcal{T}+1+l_{int}$	triples (τ,T',z), where $\tau=(T,\rho)$ is a tile for $\mathcal{K}$, T' is a type for $\mathcal{K}$, z is an integer, and among the first i triples in ρ there are z triples of the form (R,T',k)
Upto_{i,r_h}, for $i=1,\dots,n$ and $r_h\in\mathsf{N_R^+}(\mathcal{T})$	$l_{tile}+n_\mathcal{T}+1+l_{int}$	triples (τ,T',z), where $\tau=(T,\rho)$ is a tile for $\mathcal{K}$, T' is a type for $\mathcal{K}$, z is an integer, and among the first i triples in ρ there are z triples of the form (R,T',k) with $r_h\in R$
Var	l_{tile}	variables in $\mathcal{S}_\mathcal{K}$
Iq	l_{id}	inequalities in $\mathcal{S}_\mathcal{K}$ (stand-alone or within implications)
Iq*	l_{id}	stand-alone inequalities in $\mathcal{S}_\mathcal{K}$
Im	l_{id}	implications in $\mathcal{S}_\mathcal{K}$
$\mathsf{Iq}_\mathsf{L}^\mathsf{int}/\mathsf{Iq}_\mathsf{R}^\mathsf{int}$	$l_{id}+l_{int}$	pairs of (q,z) s.t. integer z occurs in inequality q on the LHS/RHS
$\mathsf{Iq}_\mathsf{L}^\mathsf{v}/\mathsf{Iq}_\mathsf{R}^\mathsf{v}$	$l_{id}+l_{tile}$	pairs of (q,v) s.t. variable v occurs in inequality q on the LHS/RHS
$\mathsf{Im}_\mathsf{L}/\mathsf{Im}_\mathsf{R}$	$2l_{id}$	pairs of (m,q) such that inequality q occurs in implication m on the LHS/RHS

where $n = c_\mathcal{T}\cdot m_\mathcal{T}$, $l_{int}=\log(\max(1,n))$, $l_{tile}=n_\mathcal{T}+1+n(n_\mathcal{T}+1+k_\mathcal{T}+l_{int})$, and $l_{\mathsf{id}}=2(n_\mathcal{T}+2)+k_\mathcal{T}+\ \max(n,1)(n_\mathcal{T}+1+k_\mathcal{T}+l_{int})$.

Table 4.1: Overview of the signature used for in $\mathcal{P}_{sys}$

Cond.	Inequality/Implication	ID
M1	$\sum_{(T,\rho)\in\mathrm{Tiles}(\mathcal{K}),\{c\}\in T} x_{(T,\rho)} \leq 1$, for $\{c\} \in \mathsf{N}_\mathsf{C}^+(\mathcal{K})$	$(0, c, \vec{0})$
M1	$1 \leq \sum_{(T,\rho)\in\mathrm{Tiles}(\mathcal{K}),\{c\}\in T} x_{(T,\rho)}$, for $\{c\} \in \mathsf{N}_\mathsf{C}^+(\mathcal{K})$	$(1, c, \vec{0})$
M2	$1 \leq \sum_{\tau\in\mathrm{Tiles}(\mathcal{K})} x_\tau$	$(\vec{0})$
M3	$\sum_{\substack{(T,\rho)\in\mathrm{Tiles}(\mathcal{K}),\\(R,T',k)\in\rho}} x_{(T,\rho)} \leq \sum_{\substack{(T',\rho')\in\mathrm{Tiles}(\mathcal{K}),\\(R^-,T,l)\in\rho'}} x_{(T',\rho')}$, for $T, T' \in \mathrm{Types}(\mathcal{K})$, $R \subseteq \mathsf{N}_\mathsf{R}^+(\mathcal{K})$ s.t. (T, R, T') invertible	$(0, \vec{T}, \vec{T'}, \vec{R}, \vec{0})$, where $\vec{T}$ and $\vec{T'}$ encode T and T', respectively, and $\vec{R}$ encodes R
M3	$\sum_{\substack{(T',\rho')\in\mathrm{Tiles}(\mathcal{K}),\\(R^-,T,l)\in\rho'}} x_{(T',\rho')} \leq \sum_{\substack{(T,\rho)\in\mathrm{Tiles}(\mathcal{K}),\\(R,T',k)\in\rho}} x_{(T,\rho)}$, for $T, T' \in \mathrm{Types}(\mathcal{K})$ and $R \subseteq \mathsf{N}_\mathsf{R}^+(\mathcal{K})$ s.t. (T, R, T') invertible	$(1, \vec{T}, \vec{T'}, \vec{R}, \vec{0})$, where $\vec{T}$ and $\vec{T'}$ encode T and T', respectively, and $\vec{R}$ encodes R
M4/M6	$1 \leq x_\tau$, for $\tau \in \mathrm{Tiles}(\mathcal{K})$	$(\vec{t}, \vec{0})$, where $\vec{t}$ encodes τ
M4	$\lvert\{(R, T', k) : (R, T', k) \in \rho\}\rvert \leq \sum_{(T',\rho')\in\mathrm{Tiles}(\mathcal{K})} x_{(T',\rho')}$, for $\tau = (T, \rho) \in \mathrm{Tiles}(\mathcal{K})$ and $T' \in \mathrm{Types}(\mathcal{K})$	$(\vec{t}, \vec{T'}, \vec{0})$, where $\vec{t}$ encodes τ and $\vec{T'}$ encodes T'
M4	$1 \leq x_\tau \implies \lvert\{(R, T', k) : (R, T', k) \in \rho\}\rvert \leq \sum_{(T',\rho')\in\mathrm{Tiles}(\mathcal{K})} x_{(T',\rho')}$, for $\tau = (T, \rho) \in \mathrm{Tiles}(\mathcal{K})$ and $T' \in \mathrm{Types}(\mathcal{K})$,	$(\vec{t}, \vec{T'}, \vec{0})$, where $\vec{t}$ encodes τ and $\vec{T'}$ encodes T'
M5	$1 \leq \sum_{\substack{(T,\rho)\in\mathrm{Tiles}(\mathcal{K}),\\\{c\},B\in T}} x_{(T,\rho)}$, for $\{c\} \in \mathsf{N}_\mathsf{C}^+(\mathcal{K})$, $B \in \mathsf{N}_\mathsf{C}(\mathcal{K})$	$(c, \vec{b}, \vec{0})$, where $\vec{b}$ encodes the type containing only B
M5	$1 \leq \sum_{\substack{(T,\rho)\in\mathrm{Tiles}(\mathcal{K}),\\\{c_1\},B\in T}} x_{(T,\rho)} \implies 1 \leq \sum_{\substack{(T',\rho')\in\mathrm{Tiles}(\mathcal{K}),\\\{c_2\},B'\in T'}} x_{(T',\rho')}$, for $\{c_1\}, \{c_2\} \in \mathsf{N}_\mathsf{C}^+(\mathcal{K})$ and $B, B' \in \mathsf{N}_\mathsf{C}(\mathcal{K})$	$(c_1, c_2, \vec{b}, \vec{b'}, \vec{0})$, where $\vec{b}$ (resp. $\vec{b'}$) encodes the type containing only B (resp. B')
M6	$\sum_{\substack{(T,\rho)\in\mathrm{Tiles}(\mathcal{K}),\{\{c\},B\}\subseteq T,\\\lvert\{(R,T',k)\in\rho : r\in R\}\rvert\leq 0}} x_{(T,\rho)} \leq 0$, for $T' \in \mathrm{Types}(\mathcal{K})$, $c \in \mathsf{N}_\mathsf{I}(\mathcal{K})$, $B \in \mathsf{N}_\mathsf{C}(\mathcal{K})$, and $r \in \mathsf{N}_\mathsf{R}^+(\mathcal{K})$	$(c, \vec{T'}, \vec{b}, \vec{r}, \vec{0})$, where $\vec{T'}$ encodes T', $\vec{b}$ encodes the type containing only B, and $\vec{r}$ encodes the role type containing only r
M6	$1 \leq x_{\tau'} \implies \sum_{\substack{(T,\rho)\in\mathrm{Tiles}(\mathcal{K}),\{\{c\},B\}\subseteq T,\\\lvert\{(R,T',k)\in\rho : r\in R\}\rvert\leq 0}} x_{(T,\rho)} \leq 0$, for $c \in \mathsf{N}_\mathsf{I}(\mathcal{K})$, $B \in \mathsf{N}_\mathsf{C}(\mathcal{K})$, $\tau' = (T', \rho') \in \mathrm{Tiles}(\mathcal{K})$, and $r \in \mathsf{N}_\mathsf{R}^+(\mathcal{K})$	$(c, \vec{t'}, \vec{b}, \vec{r}, \vec{0})$, where $\vec{t'}$ encodes τ', $\vec{b}$ encodes the type containing only B, and $\vec{r}$ encodes the role type containing only r

where $l_{\mathsf{id}} = 2(n_\mathcal{T} + 2) + k_\mathcal{T} + \max(c_\mathcal{T} \cdot m_\mathcal{T}, 1)(n_\mathcal{T} + 1 + k_\mathcal{T} + \log(\max(c_\mathcal{T} \cdot m_\mathcal{T}, 1)))$, and $\vec{0}$ is always a all-zero vector of the appropriate length.

Table 4.2: Identifiers of inequalities and implications of $\mathcal{S}_\mathcal{K}$,

Algorithm 4.1: Computation of Σ-range-restricted free variables in a FO query $q(\vec{x})$

Input: a constant-free FO query $q(\vec{x})$ in SRNF and a set Σ of predicates
Result: a subset of free variables of q or *'fail'*

case q **of**
- $R(\vec{x})$ **:**
  - **if** $R \in \Sigma$ **then**
    - $\mathsf{rr}(q, \Sigma) = \vec{x}$
  - **else**
    - $\mathsf{rr}(q, \Sigma) = \emptyset$
- $(\varphi_1 \wedge \varphi_2)$ **:**
  - $\mathsf{rr}(q, \Sigma) = \mathsf{rr}(\varphi_1, \Sigma) \cup \mathsf{rr}(\varphi_2, \Sigma)$
- $(\varphi_1 \vee \varphi_2)$ **:**
  - $\mathsf{rr}(q, \Sigma) = \mathsf{rr}(\varphi_1, \Sigma) \cap \mathsf{rr}(\varphi_2, \Sigma)$
- $(\neg\varphi)$ **:**
  - $\mathsf{rr}(q, \Sigma) = \emptyset$
- $\varphi \wedge x = y$ **:**
  - **if** $\mathsf{rr}(\varphi, \Sigma) \cap \{x, y\} = \emptyset$ **then**
    - $\mathsf{rr}(q, \Sigma) = \mathsf{rr}(\varphi, \Sigma)$
  - **else**
    - $\mathsf{rr}(q, \Sigma) = \mathsf{rr}(\varphi, \Sigma) \cup \{x, y\}$
- $\exists x\varphi$ **:**
  - **if** $x \in \mathsf{rr}(\varphi, \Sigma)$ **then**
    - $\mathsf{rr}(q, \Sigma) = \mathsf{rr}(\varphi, \Sigma) \setminus \{x\}$
  - **else**
    - **return** *'fail'*

CHAPTER 5

Descriptive Complexity of OMQs with Closed Predicates

So far, we have explored the relative expressiveness and data complexity of OMQs with closed predicates and we have shown that there is a large class of such OMQs that can be rewritten into Datalog$^{\neg}$ and answered in coNP. In this chapter, we investigate the *expressive power* of OMQs from a *descriptive complexity* [Imm99] perspective. In this context, we ask whether a given OMQ language is powerful enough to express all queries computable within some time or space resources, i.e., belonging to a certain complexity class. In fact, many OMQ languages based on expressive DLs (including the ones investigated in the previous chapter) are coNP complete in data complexity, but, to the best of our knowledge, no capturing results exist. It was shown in [BtCLW14] that there are OMQ languages that capture certain subclasses of coNP related to *constraint satisfaction problems* (CSPs). More recently, a close connection between OMQ languages with the so-called *closed predicates* and *surjective CSPs* was shown in [LSW19]. We continue this line of work and we focus on finding an OMQ language that precisely captures the complexity class coNP. It is easy to see that OMQs based on standard expressive DLs and conjunctive queries—while being coNP-complete in data complexity—are not powerful enough to express all queries computable in coNP. This follows directly from the *monotonicity* of standard OMQ languages (e.g., in all cases where the OMQ language is based on first-order logic). Specifically, if Q is a Boolean OMQ in such a language, then for all ABox pairs $\mathcal{A}_1 \subseteq \mathcal{A}_2$ we have that $Q(\mathcal{A}_1) = 1$ implies $Q(\mathcal{A}_2) = 1$. These OMQ languages cannot capture coNP since there exist (rather trivial) *non-monotonic* queries computable in coNP (or even polynomial time). Here is a simple example of such a (non-monotonic) query: "Does the input ABox have an odd number of individuals?" We have already seen that closed predicates introduce non-monotonicity, making OMQ languages with closed predicates a good candidate for capturing coNP. In this chapter, we investigate what features, other than closed predicates, make an OMQ

language sufficiently expressive to capture the class of all queries computable in coNP, while still maintaining coNP-completeness in data complexity.

Contributions and Relevant Publications. We can summarize the contributions presented in this chapter as follows:

- We first present an inexpressibility result for *non-monotonic* OMQs based on $\mathcal{ALCHOI}$ with closed predicates. For such expressive OMQs the aforementioned monotonicity-based argument does not apply, and we need a more sophisticated approach. Specifically, by analyzing an existing algorithm in [AOŠ20] and invoking the *Non-deterministic Time Hierarchy Theorem*, we show that instance queries mediated by $\mathcal{ALCHOI}$ TBoxes with closed predicates are not expressive enough to capture coNP.
- We then present an OMQ language that is powerful enough to express all coNP computable queries. As our base DL we choose $\mathcal{ALCHOIF}$ (with closed predicates) and we add to it restricted forms of transitivity, complex role expressions and *nominal schemata* [KMKH11]. We argue that these additions do not cause an increase in combined complexity and data complexity, i.e. answering ontology-mediated instance queries remains complete for NExpTime and coNP, respectively. This is done by suitably modifying the mosaic-based algorithm from Chapter 3.
- We prove that our enriched OMQ language is powerful enough to express all (so-called *generic*) Boolean queries computable in coNP. Each such generic query q is associated to a signature Σ as well as to a set of ABoxes over Σ (also called Σ-ABoxes) in which the answer to the query is "true". By saying that "q is computable in coNP" we mean that there is a non-deterministic Turing Machine M_q that recognizes the language of strings representing Σ-ABoxes in which the answer to the query is "false" and runs in polynomial time in the size (of the string representation) of the input ABox. We show that the enriched OMQ language can properly express the computations of M_q. As a consequence of this and the previous point, we obtain a language that captures precisely coNP.

The results from this chapter have been published in:

> [LOŠ23] **Sanja Lukumbuzya**, Magdalena Ortiz, and Mantas Šimkus. "On the expressive power of ontology-mediated queries: Capturing coNP". In Proceedings of the 36th International Workshop on Description Logics, DL 2023, CEUR-WS, 2023.

Organization In Section 5.1 we introduce the notion of generic Boolean queries and what it means for such a query to be coNP-computable. In Section 5.2 we present our observation about expressiveness limitations of standard (i.e., monotonic) DLs when it

comes to capturing coNP. Furthermore, we show also show that inconsistency/instance queries mediated by $\mathcal{ALCHOI}$ ontologies with closed predicates are still not expressive enough. In the same section, we present our candidate OMQ language: the extension of $\mathcal{ALCHOIF}$ with restricted nominal schemata in combination with inconsistency queries. Section 5.3 is dedicated that the query answering problem for the chosen OMQ language is still coNP-complete in data complexity. The main result of this chapter is presented in Section 5.4.2, where we show that our language indeed captures coNP. Finally, we end this chapter with a brief discussion in Section 5.5.

5.1 Generic Boolean Queries

The main goal of this chapter is to show that we can define an OMQ language based on the DL $\mathcal{ALCHOIF}$ with closed predicates that *captures* the complexity class coNP. As a first step, we need to precisely define what we mean by "capturing a complexity class". To this end, we introduce the notion of *generic boolean queries* over DL ABoxes, which are simply functions that map ABoxes to either 0 (false) or 1 (true), and we clarify what it means for such a query to be a member of a certain complexity class.

Definition 5.1.1. *We say ABoxes $\mathcal{A}_1, \mathcal{A}_2$ are* isomorphic*, if they are equal up to the renaming of individuals, i.e. there is a bijection $f : \mathsf{N_I}(\mathcal{A}_1) \to \mathsf{N_I}(\mathcal{A}_2)$ such that $\mathcal{A}_2 = \{A(f(c)) : A(c) \in \mathcal{A}_1)\} \cup \{r(f(c), f(d)) : r(c,d) \in \mathcal{A}_1)\}$.*

Definition 5.1.2 (Generic Boolean Queries)**.** *A* Generic Boolean Query *Q over a signature $\Sigma \subseteq \mathsf{N_C} \cup \mathsf{N_R}$ (Σ-GBQ) is a function that maps each Σ-ABox $\mathcal{A}$ to a value $Q(\mathcal{A}) \in \{0, 1\}$, and is such that $Q(\mathcal{A}_1) = Q(\mathcal{A}_2)$ holds for any pair $\mathcal{A}_1, A_2$ of isomorphic Σ-ABoxes.*

The assumption that answers GBQs are invariant under isomorphic ABoxes is natural: we are interested in queries about the structure of ABoxes, and they should not depend on the concrete names of individuals. Dropping this assumption would render the expressiveness analysis virtually meaningless: because an OMQ (or any standard database query) can only use a finite number of constants in the query expression, many computationally trivial queries could not be expressed even in very powerful query languages.

Recall that standard complexity classes, including coNP, are defined using Turing machines (cf. Chapter 2). Intuitively, a Σ-GBQ Q belongs to a complexity class $\mathcal{C}$ if there is a suitable Turing machine that can decide, for every Σ-ABox $\mathcal{A}$, whether Q maps $\mathcal{A}$ to 1 within the time and space bounds that are imposed by $\mathcal{C}$. However, Turing Machines do not operate on ABoxes but rather on strings, which means that in order to compute an answer to Q over an ABox $\mathcal{A}$, we need to suitably encode $\mathcal{A}$ as a string. To this end, we choose a simple encoding, where we first enumerate all pairs c_i, c_j of individuals in $\mathcal{A}$. Then, for each such pair, we store in a *single symbol* all the concept names asserted for those individuals along with the roles that link them. Note that different types of encodings are possible (cf. [Imm99], Chapter 2.2).

Definition 5.1.3 (Encoding ABoxes as words)**.** *Consider a fixed signature Σ. A* 2-type over Σ *is a tuple (T, R, T'), where $T, T' \subseteq \Sigma \cap \mathsf{N_C}$ and $R \subseteq \Sigma \cap \mathsf{N_R^+}$. We let Γ^Σ denote the set of all 2-types over Σ. We next define an* encoding function *enc^Σ that maps Σ-ABoxes to words over Γ^Σ. Assume a Σ-ABox $\mathcal{A}$ with ℓ individuals and take an arbitrary enumeration $c_1, \dots, c_\ell$ of the individuals in $\mathcal{A}$. Then we define*

$$enc^\Sigma(\mathcal{A}) = \sigma_{1,1} \dots \sigma_{1,\ell}\, \sigma_{2,1} \dots \sigma_{2,\ell} \dots \sigma_{\ell,1} \dots \sigma_{\ell,\ell},$$

where each $\sigma_{i,j} := (\{A : A(c_i) \in \mathcal{A}\}, \{r : r(c_i, c_j) \in \mathcal{A}\}, \{A : A(c_j) \in \mathcal{A}\})$.

Based on the encoding above, we can now formally define membership of a GBQ in a complexity class.

Definition 5.1.4. *Let Σ be a signature and Q a Σ-GBQ. We say Q* belongs to *a complexity class $\mathcal{C}$, if there is a Turing machine M that decides the language*

$$\{enc^\Sigma(\mathcal{A}) : \mathcal{A} \text{ is a } \Sigma\text{-ABox with } Q(\mathcal{A}) = 1\},$$

over Γ^Σ within the time or space bounds imposed by $\mathcal{C}$.

Proposition 5.1.5. *Assume a GBQ Q over Σ. The following are equivalent:*

1. *Q belongs to* coNP.
2. *There is an integer k and a NTM M with alphabet Γ^Σ such that, for any Σ-ABox $\mathcal{A}$ we have:*
   - *$Q(\mathcal{A}) = 0$ if and only if M accepts $enc^\Sigma(\mathcal{A})$;*
   - *M terminates within $|enc^\Sigma(\mathcal{A})|^k$ computation steps.*

5.2 Inexpressibility Results and Language Extension

As stated in Chapter 2, OMQ languages based on standard DLs whose TBoxes can be expressed in first-order logic are monotonic. In particular, this holds even for very expressive DLs, like $\mathcal{ALCHOIQ}$ and its sublogics. There are many very simple GBQs that can be computed in coNP (or even PTime) but are non-monotonic and therefore not expressible in monotonic OMQ languages.

Theorem 5.2.1. *There exists a GBQ Q_1 over Σ such that (a) Q_1 belongs to* coNP*, and (b) there is no OMQ $Q_2 = (\mathcal{T}, q)$, with an $\mathcal{ALCHOIQ}$ TBox $\mathcal{T}$ and a FO query q, such that $Q_1(\mathcal{A}) = Q_2(\mathcal{A})$ for all Σ-ABoxes $\mathcal{A}$.*

Proof. Take any signature Σ and let $Q_1 = Q_{parity}$ be the previously-introduced parity query that asks whether in a given Σ-ABox $\mathcal{A}$ there is an odd number of individuals in some unary relation $A \in \Sigma$. In other words, for all Σ-ABoxes $\mathcal{A}$, $Q_1(\mathcal{A}) = 1$ if and only

if ‘ This query is clearly in coNP, in fact, it can be answered using logarithmic space only. However, as already explained, Q_1 is not monotonic. For example, $Q_1(\{A(c_1)\}) = 1$ but $Q_1(\{A(c_1), A(c_2)\}) = 0$. Thus Q_1 cannot be captured as Q_2 above, since any such Q_2 is monotonic, i.e. for any pair $\mathcal{A}_1 \subseteq A_2$ of Σ-ABoxes, if $Q_2(\mathcal{A}_1) = 1$, then also $Q_2(\mathcal{A}_2) = 1$. □

We have already seen in Example 4.2.4 in Chapter 4, the parity query from the previous example can easily be expressed in $\mathcal{ALCHOIQ}$ with closed predicates since closed predicates add non-monotonicity. In fact, Example 4.2.3 from Chapter 4 shows that even much weaker DL $\mathcal{ALC}$ with closed predicates already exhibits non-monotonic behavior. Thus, we need a different argument to show that a certain OMQ language with closed predicates does not capture coNP.

We next argue that OMQ languages whose query answering problem can be reduced to deciding the inconsistency of the DL $\mathcal{ALCHOI}$ with closed predicates do not capture coNP. To this end, we introduce the notion of ontology-mediated *inconsistency queries*.

Definition 5.2.2 (Inconsistency query)**.** *Let $Q = (\mathcal{T}, \Sigma, q)$ be an OMQ. If $q = \bot$, we say that Q is an* ontology-mediated inconsistency query with closed predicates*, or simply an* inconsistency query*. We often omit $\bot$, and simply write $(\mathcal{T}, \Sigma)$ to denote the inconsistency query $(\mathcal{T}, \Sigma, \bot)$.*

Theorem 5.2.3. *There exists a GBQ Q_1 over Σ such that:*

(a) Q_1 belongs to coNP, and

(b) there is no inconsistency query $Q_2 = (\mathcal{T}, \Sigma')$, with $\mathcal{T}$ in $\mathcal{ALCHOI}$, such that $Q_1(\mathcal{A}) = Q_2(\mathcal{A})$ holds for all Σ-ABoxes $\mathcal{A}$.

Proof. To see this, we can analyze the running time of the algorithm in [AOŠ20] for checking satisfiability of an $\mathcal{ALCHOI}$ knowledge base $\mathcal{K} = (\mathcal{T}, \Sigma', \mathcal{A})$, where Σ' is a set of closed predicates. We assume that $\mathcal{T}, \Sigma'$ are fixed, and we want an algorithm that takes an ABox $\mathcal{A}$ as input and checks if $\mathcal{K} = (\mathcal{T}, \Sigma', \mathcal{A})$ is consistent. Specifically, the algorithm presented in [AOŠ20] runs in time bounded by $|\mathcal{A}|^k \times \ell + v$, where ℓ and v are constants that depend on $\mathcal{T}$ and Σ', while k is a constant that does not depend on $\mathcal{T}$ or Σ'. In other words, there is a constant k, such that for any $\mathcal{T}$ and Σ', we can build a non-deterministic algorithm that checks the consistency of an input ABox $\mathcal{A}$ in the KB $\mathcal{K} = (\mathcal{T}, \Sigma', \mathcal{A})$, and that runs in time $\mathcal{O}(|\mathcal{A}|^k)$.

The key here is that the constant k does not depend on $\mathcal{T}$ or Σ'. Using an $\mathcal{ALCHOI}$ TBox with closed predicates one can capture decision problems that can be solved via a non-deterministic TM in time that is polynomial with degree k. However, using the *Non-deterministic Time Hierarchy Theorem* [Coo72] (see also [Žák83]), we know that there are problems that can be solved in non-deterministic polynomial time, but not in non-deterministic polynomial time with the polynomial degree k. Specifically, there a problems solvable in time $\mathcal{O}(n^{k+1})$ but not $\mathcal{O}(n^k)$. □

We note that the above result can be formulated also for OMQs with atomic queries and a certain class of conjunctive queries, but it is unclear if it generalizes to OMQs with full conjunctive queries. This is because the proof of Theorem 5.2.3 relies on an upper bound on the running time of a known algorithm for $\mathcal{ALCHOI}$ with closed predicates. To the best of our knowledge, no suitable upper bounds on data complexity are known in the case of CQs over $\mathcal{ALCHOI}$ KBs. Furthermore, at this point, it is unfortunately unclear whether one can prove the same inexpressibility result for $\mathcal{ALCHOIF}$ or $\mathcal{ALCHOIQ}$ with closed predicates. This is left as future work.

5.2.1 $\mathcal{ALCHOIF}^+$

As stated above, it is unclear whether OMQs based on $\mathcal{ALCHOIF}$ with closed predicates are capable of capturing coNP. However, we present an extension of $\mathcal{ALCHOIF}$ that we call $\mathcal{ALCHOIF}^+$, and we prove that the OMQ language based on $\mathcal{ALCHOIF}^+$ with closed predicates and inconsistency queries is indeed powerful enough to capture the desired class of GBQs. In addition to standard $\mathcal{ALCHOIF}$ axioms and closed predicates, this logic supports (i) *transitivity* axioms over roles with whose domain/range is restricted to the active domain of the KB and (ii) axioms involving *role composition* and *nominal schemata*. Regarding (ii), nominal schemata were introduced in [KMKH11] and they are known to cause an exponential increase in the complexity of reasoning [KR14]. In order to keep the complexity unaffected, our language extension only allows *syntactically restricted shapes* of nominal schemata that we show that we can handle within the provided complexity bounds, i.e., coNP in data complexity. Moreover, our extension also allows us to combine nominal schemata with role composition, however, once again, only in axioms of certain syntactic shape. We next formally introduce the syntax and semantics of $\mathcal{ALCHOIF}^+$ and explain the intuitions behind the non-standard axioms.

Syntax. We begin with the syntax of $\mathcal{ALCHOIF}^+$.

Definition 5.2.4. *In $\mathcal{ALCHOIF}^+$, a* role *is an expression of the form p or p^-, where $p \in \mathsf{N}_R$ is a role name. An expression of the form $A?$, where $A \in \mathsf{N}_C$, is called a* test role*, and* complex roles *are expressions of the form*

$$A? \circ r_1 \circ A_1? \circ \dots r_n \circ A_n,$$

where $n \geq 1$, r_i is a role, and A,A_i are test roles, for all $1 \leq i \leq n$.

(Complex) concepts *in $\mathcal{ALCHOIF}^+$ are simply the complex concepts in $\mathcal{ALCHOIF}$. As a reminder, they are defined inductively as follows:*

- *every concept name $A \in \mathsf{N}_C$ is a concept,*
- *every nominal $\{o\}$, $o \in \mathsf{N}_I$ is a concept,*
- $\top$ (top) *and* $\bot$ (bottom) *are concepts,*

- *if* C_1 *and* C_2 *are concepts, then so are* $C_1 \sqcap C_2$ *(*concept conjunction*),* $C_1 \sqcup C_2$ *(*concept disjunction*), and* $\neg C_1$ *(*concept negation*), and*
- *if* C *is a concept and* r *is a role, then* $\exists r.C$ *(*existential restriction*) and* $\forall r.C$ *(*universal restriction*) are concepts.*

Terminological axioms *in* $\mathcal{ALCHOIF}^+$ *have one of the following forms:*

A1. $C \sqsubseteq D$,

A2. $\mathsf{func}(r)$,

A3. $\mathsf{trans}(p)$,

A4. $\exists P.(\{x\} \sqcap \exists s.\{y\}) \sqcap \exists R.\{y\} \sqsubseteq B$,

A5. $\exists P.(\{x\} \sqcap \neg\exists s.\{y\}) \sqcap \exists R.\{y\} \sqsubseteq B$,

A6. $\{x\} \sqsubseteq \forall S.\{x\}$, *or*

A7. $\{x\} \sqcap A \sqsubseteq \forall s.\neg\{x\}$,

where A, B *are concept names,* C, D *are concepts,* r *is a role,* p, s *are* restricted *roles and* $p \in \mathsf{N}_R$*,* P, R *are complex roles consisting only of test roles and functional roles,* S *is a complex role and* $\{x, y\} \in \mathsf{N}_V$*.*

Note that in the previous definition, we refer to *functional* and *restricted* roles. We that a role p is functional if $\mathsf{func}(p) \in \mathcal{T}$. Intuitively, a role p is restricted if we can guarantee that, in any model $\mathcal{I}$ of the KB, $(e, e') \in p^{\mathcal{I}}$ implies that e and e' are constants occurring in the KB. We next give a syntactic criterion that characterizes such roles. We note that this characterization is incomplete, in the sense that there may be other roles whose extensions are restricted over the active domain, however, it is nonetheless sufficient for our purposes.

Definition 5.2.5. *A basic concept* $A \in \mathsf{N}_C^+$ *is* restricted w.r.t. an $\mathcal{ALCHOIF}^+$ TBox $\mathcal{T}$ and a set Σ of closed predicates *if one of the following conditions hold:*

- $A \in \Sigma \cup \{\{a\} : a \in \mathsf{N}_I(\mathcal{T})\} \cup \{\bot\}$,
- $A \sqsubseteq B_1 \sqcup \cdots \sqcup B_n \in \mathcal{T}$*, where* B_i *is a restricted concept or an expression of the form* $\exists r.C$ *or* $\exists r^-.C$*, for a* restricted role r *and a concept* C*, for all* $1 \leq i \leq n$*.*

A role $r \in \mathsf{N}_R^+$ *is called* restricted w.r.t. a TBox $\mathcal{T}$ and a set Σ of closed predicates *if one of the following holds:*

- $r \in \Sigma$,

- $\{\exists r \sqsubseteq A, \exists r^- \sqsubseteq B\} \subseteq \mathcal{T}$, *where* A *and* B *are restricted concepts,*
- r^- *is a restricted role, or*
- $r \sqsubseteq s$, *where* s *is a restricted role.*

Proposition 5.2.6. *It can be recognized in polynomial time whether a given role* $r \in \mathsf{N}_\mathsf{R}^+(\mathcal{T})$ *or a basic concept* $A \in \mathsf{N}_\mathsf{R}^+(\mathcal{T})$ *is restricted w.r.t. a given* $\mathcal{ALCHOIF}^+$ *TBox* $\mathcal{T}$ *and a set of closed predicates* Σ.

Proof. Observe that Definition 5.2.5 does not use axioms (A4)-(A7) to infer that some concept or role is restricted. Thus, we can assume that $\mathcal{T}$ is an $\mathcal{ALCHOIF}$ ABox. We can then devise an algorithm that, starting from the initial sets $S_C = (\Sigma \cap \mathsf{N_C}) \cup \{\{a\} : a \in \mathsf{N_I}(\mathcal{T})\} \cup \{\bot\}$ of restricted basic concepts and $S_R = \{r, r^- : r \in \Sigma \cap \mathsf{N_R}\}$ of restricted role names, saturates these sets with all restricted basic concepts and roles, respectively. The algorithm computes in one step all concept names and roles that we can infer are restricted from S_C, S_R and the axioms of $\mathcal{T}$. This clearly takes polynomial time. This is repeated as long as new information is inferred. In the worst case, we have one iteration per each $A \in \mathsf{N_C}$ and $r \in \mathsf{N_R^+}$, which means there are at most polynomially many iterations of the loop. Finally, we simply check whether r (resp. A) is in the saturated set S_R (resp. S_C). □

We next briefly present the intuitions behind axioms (A4)-(A7) rather informally. Consider some interpretation $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$. Axioms of the form (A4) allow an anonymous domain element $d \in \Delta^\mathcal{I}$ to infer its participation in the concept B as a result of some connection between individuals in the data part (i.e., ABox) that this element can see using chains of test and functional roles. More precisely, if d has a way to reach some individual a using a chain P of test and functional roles, d can reach some individual b in the same way via a chain R, and the pair (a, b) participates in the extension of some restricted role s, then it can be inferred that d is in B (axiom (A4)). Similarly, axioms of the form (A5) allow us to infer the same from the absence of such connection in the data part, i.e., if (a, b) is *not* in the extension of some restricted role s. Moving on to axioms of the form (A6), they allow us to assert that all role chains of a certain type starting from an individual must in fact be cycles. Finally, (A7) is used to forbid the existence of certain self-loops on individuals.

Semantics We next present the semantics of $\mathcal{ALCHOIF}^+$. Let $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ be an interpretation. The extension of the interpretation function $\cdot^\mathcal{I}$ to complex roles P of the form $A? \circ r_1 \circ A_1? \circ \cdots \circ r_n \circ A_n?$ is defined as:

$$P^\mathcal{I} := \{e_0 \in \Delta^\mathcal{I} \cap A^\mathcal{I} : \exists e_1, \ldots, e_n \in \Delta^\mathcal{I} \text{ s.t. } (e_{i-1}, e_i) \in r_i^\mathcal{I}, e_i \in A_i^\mathcal{I}, \text{ for all } 1 \leq i \leq n\}.$$

We still need to define what it means for $\mathcal{I}$ to satisfy axioms that contain nominal variables, i.e., axioms (A4)-(A7). In [KR14, KMKH11] the semantics of axioms containing nominal

variables is given by grounding the axiom with respect to the set of all individuals $\mathsf{N_I}$, where $\mathsf{N_I}$ is assumed to be finite. For our purposes, we ground such axioms with respect to $\mathsf{N_I}(\mathcal{K})$ by uniformly replacing all nominal variables with nominals $\{a\}$, s.t. $a \in \mathsf{N_I}(\mathcal{K})$ in all possible ways. We first define what it means to ground an axiom with nominal variables with respect to a given set of individuals, similarly to the way that the grounding is defined in Datalog (cf. Section 2.4).

Definition 5.2.7. *Given a set of individuals $C \subseteq \mathsf{N_I}$, the* grounding of an axiom α w.r.t. *C, in symbols $ground(\alpha, C)$, is a set of axioms obtained from α by replacing every nominal variable $\{x\}$ in α with the nominal $\{\sigma(x)\}$, for every total function $\sigma : vars(\alpha) \to C$, where $vars(\alpha)$ denotes the set of all variables occurring in α. The* grounding of a KB $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$ w.r.t. *C is then computed as $ground(\mathcal{K}, C) = \bigcup_{\alpha \in \mathcal{T}} ground(\alpha, C)$.*

Definition 5.2.8. *Let $\mathcal{I}$ be an interpretation and let $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$ be an $\mathcal{ALCHOIF}^+$ KB with closed predicates. We say that $\mathcal{I}$* satisfies *$\mathcal{K}$, if $\mathcal{I}$ satisfies $ground(\mathcal{K}, \mathsf{N_I}(\mathcal{K}))$.*

The way that $\mathcal{ALCHOIF}^+$ was crafted ensures that the computational complexity of answering inconsistency queries is the same as that of $\mathcal{ALCHOIF}$.

Theorem 5.2.9. *The KB satisfiability problem in $\mathcal{ALCHOIF}^+$ with closed predicates is NP-complete in data complexity, and NExpTime-complete in combined complexity. Answering Boolean inconsistency queries mediated by $\mathcal{ALCHOIF}^+$ ontologies with closed predicates is coNP-complete in data complexity and co-NExpTime-complete in combined complexity.*

We dedicate the next section to the proof of Theorem 5.2.9.

5.3 Data Complexity of $\mathcal{ALCHOIF}^+$

The goal of this section is to provide a proof of Theorem 5.2.9 by devising a decision procedure for KB satisfiability of $\mathcal{ALCHOIF}^+$ KBs with closed predicates that runs in nondeterministic exponential time in the size of the given knowledge base, and in nondeterministic polynomial time, if the TBox and the set of closed predicates are considered fixed. We begin by giving a brief overview of the steps involved in this procedure. The first step is to guess the extensions of restricted concepts and roles over the individuals occurring in the given knowledge base – these concepts and role names are now considered closed. Since all transitivity axioms only involve restricted roles, whose extensions are fully known after the first step, we can right away check whether they are satisfied and eliminate them. Thus, what is left to do is devise a procedure that can decide the satisfiability of KBs with closed predicates whose TBox contains no transitivity axioms, and where all restricted concepts and role names are closed. We do this by modifying the mosaic approach introduced in Chapter 3 for $\mathcal{ALCHOIF}$ (cf. Section 3.3).

We next formalize this. To this end, consider an $\mathcal{ALCHOIF}^+$ KB $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$.

Guess restricted concepts and roles. As already mentioned, the first step is to guess the extensions of restricted concepts and roles. To this end, we non-deterministically construct a new KB $\mathcal{K}_{ext} = (\mathcal{T}, \Sigma \cup \mathsf{RestrPred}, \mathcal{A}_{ext})$, where $\mathcal{A}_{ext}$ is obtained by potentially adding $A(a)$, for each individual $a \in \mathsf{N_I}(\mathcal{K})$ and each restricted concept name A, and $r(a, b)$, for each pair of individuals $a, b \in \mathsf{N_I}\mathcal{K}$ and each restricted role name r. Notice that $\mathcal{K}_{ext}$ is obtained in time polynomial in the size of $\mathcal{K}$.

Eliminate transitivity. For all transitivity axioms $\mathsf{trans}(r) \in \mathcal{T}$, check whether whenever $\{r(a, b), r(b, c)\} \in \mathcal{A}_{ext}$, then also $r(a, c) \in \mathcal{A}_{ext}$, for all individuals a, b, c occurring in $\mathcal{A}_{ext}$. This check is also done in time polynomial in the size of $\mathcal{K}_{ext}$ (and therefore in the size of $\mathcal{K}$). If this is not the case, $\mathcal{K}_{ext}$ is unsatisfiable. Otherwise $\mathcal{K}_{ext}$ is satisfiable if and only if $\mathcal{K}^{ntr}_{ext}$ is satisfiable, where $\mathcal{K}^{ntr}_{ext}$ is obtained from $\mathcal{K}_{ext}$ by removing all transitivity axioms.

Decide satisfiability of $\mathcal{K}^{ntr}_{ext}$. We now show how to decide satisfiability of an $\mathcal{ALCHOIF}^+$ KB $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$ that contains no transitivity axioms, and where the role s in the axioms of the form (A4), (A5) and (A7) is closed instead of restricted. To this end we modify the approach from Chapter 3 to characterize models of $\mathcal{K}$ in terms of *tiles*, i.e., compact description of domain elements and their relevant neighborhood, and *mosaics*, i.e., functions that tell us how many instances of tiles of each kind we need in order to build a model. We can then use regular integer programming techniques to decide whether such a mosaic exists.

As before, our characterization assumes (w.l.o.g.) that KBs are given in a certain normal form and their TBoxes are closed under role inclusions. Here, we reuse the $\mathcal{ALCHOIF}$ normal form (cf. Definition 3.1.14 in Chapter 3) for the part of the TBox that can be expressed in $\mathcal{ALCHOIF}$ (i.e., axioms of the form (A1) and (A2)) and we leave the remaining axioms as they are, with the caveat that all restricted roles must be in the set of closed predicates Σ. More precisely, given a KB $K = (\mathcal{T}, \Sigma, \mathcal{A})$, we assume that all axioms in $\mathcal{T}$ have one of the following forms:

$$\textbf{(N1)}\ B_1 \sqcap \cdots \sqcap B_{k-1} \sqsubseteq B_k \sqcup \cdots \sqcup B_m,$$

$$\textbf{(N2')}\ B_1 \sqsubseteq \exists p.B_2 \qquad \textbf{(N3)}\ B_1 \sqsubseteq \forall p.B_2, \qquad \textbf{(N4)}\ r \sqsubseteq s, \qquad \textbf{(N5)}\ \mathsf{func}(r),$$

$$\textbf{(A4)}\ \exists P.(\{x\} \sqcap \exists s.\{y\}) \sqcap \exists R.\{y\} \sqsubseteq B, \quad \textbf{(A5)}\ \exists P.(\{x\} \sqcap \neg\exists s.\{y\}) \sqcap \exists R.\{y\} \sqsubseteq B,$$

$$\textbf{(A6)}\ \{x\} \sqsubseteq \forall S.\{x\}, \qquad \textbf{(A7)}\ \{x\} \sqcap A \sqsubseteq \forall s.\neg\{x\},$$

where $\{B, B_1, \ldots, B_m\} \subseteq \mathsf{N^+_C}$, $p \in \mathsf{N_R}$, and $r \in \mathsf{N^+_R}$, S is a complex role, P, R are complex roles consisting only of functional roles and tests, $s \in \Sigma \cup \mathsf{N_R}$, and $\{x, y\} \subseteq \mathsf{N_V}$.

Let us next define the relevant notions. In Section 3.2.2, we introduced the notion of a *(unary) type* for a knowledge base $\mathcal{K}$ and we denoted the set of all types for $\mathcal{K}$ by Types($\mathcal{K}$). Recall, a type T for $\mathcal{K}$ is a subset of $\mathsf{N^+_C}(\mathcal{K})$ that specifies which basic concepts a domain element of this type participates in. Recall also that, due to SNA, T contains at

most one nominal. We import this notion of the unary type, and we additionally define two sets $\mathsf{Paths}_\exists(\mathcal{K})$ and $\mathsf{Paths}_\forall(\mathcal{K})$ that store all complex roles (without the initial test) and their subroles that occur in existential and universal restrictions in $\mathcal{K}$, respectively. We note that ε serves as a marker for the end of the role, and is there simply for the ease of presentation. The significance of these sets will become obvious in the rest of this section.

Definition 5.3.1. *Given a knowledge base $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$, let $\mathsf{Paths}_\exists(\mathcal{K})$ be defined as*

$$\begin{aligned}\mathsf{Paths}_\exists(\mathcal{K}) = \{&f_1 \circ A_1? \circ f_2 \circ A_2? \circ \cdots \circ f_n, \circ A_n? \circ \varepsilon,\\ &f_2 \circ A_2? \circ \cdots \circ f_n \circ A_n? \circ \varepsilon,\\ &\ldots,\\ &fn \circ A_n? \circ \varepsilon,\\ &\varepsilon : P = A_0? \circ f_1 \circ A_1? \circ f_2 \circ A_2? \circ \cdots \circ f_n, \circ A_n?,\\ &\text{ and } P \text{ occurs in } \mathcal{K} \text{ in some axiom of the form (A4) or (A5)}\}\end{aligned}$$

Similarly we let $\mathsf{Paths}_\forall(\mathcal{K})$ be defined as follows:

$$\begin{aligned}\mathsf{Paths}_\forall(\mathcal{K}) = \{&f_1 \circ A_1? \circ f_2 \circ A_2? \circ \cdots \circ f_n, \circ A_n? \circ \varepsilon,\\ &f_2 \circ A_2? \circ \cdots \circ f_n \circ A_n? \circ \varepsilon,\\ &\ldots,\\ &fn \circ A_n? \circ \varepsilon\\ &\varepsilon : P = A_0? \circ f_1 \circ A_1? \circ f_2 \circ A_2? \circ \cdots \circ f_n, \circ A_n?,\\ &\text{ and } \forall P.\{x\} \text{ occurs in some axiom of } \mathcal{K}\}\end{aligned}$$

Furthermore, we let $\Pi_\exists(\mathcal{K})$ and $\Pi_\forall(\mathcal{K})$ be defined as:

$$\begin{aligned}\Pi_\exists(\mathcal{K}) &= \{(P, \{a\}) : P \in \mathsf{Paths}_\exists(\mathcal{K}), \{a\} \in \mathsf{N}_\mathsf{C}^+(\mathcal{K})\}\\ \Pi_\forall &= \{(P, \{a\}), (P, \bot) : P \in \mathsf{Paths}_\forall(\mathcal{K}), \{a\} \in \mathsf{N}_\mathsf{C}^+(\mathcal{K})\}\end{aligned}$$

Observation: For each pair $(P, C) \in \Pi_\exists(\mathcal{K})$, the complex role P consists only of tests and functional roles.

Next, we adapt the previously-introduced notion of tiles for $\mathcal{ALCHOIF}$ (see Definition 3.3.1) to cater to $\mathcal{ALCHOIF}^+$ KBs. The main difference between the two notions is that the tiles for $\mathcal{ALCHOIF}^+$, in addition to the unary type T of the central element d and the description ρ of its relevant neighborhood, also need to keep track of whether d is a part of some relevant role chain whose existence forces us to infer some information. More precisely, given an $\mathcal{ALCHOIF}^+$ KB with closed predicates and an interpretation $\mathcal{I}$, a tile $(T, \rho, \pi_\exists, \pi_\forall)$ for $\mathcal{K}$ describes a domain element $d \in \Delta^\mathcal{I}$ for which the following hold:

1. d participates in exactly in those basic concepts that are given in T,
2. for each $(P, C) \in \pi_\exists$, there is an element e s.t. (d, e) participates in the complex role P and e participates in C or, in case $P = \varepsilon$, d participates in C
3. for each $(P, C) \in \pi_\forall$ and all elements e, if (d, e) participates in the complex role P, then e participates in C, or in case $P = \varepsilon$, d participates in C, and
4. for each 4-tuple $(R, T', \pi'_\exists, \pi'_\forall) \in \rho$, d has an arc labeled by R to a domain element that participates in the basic concepts given by T' and satisfies $\pi'_\exists$, $\pi'_\forall$ as explained in items 1 and 2.

In this case, we say that d is an *instance* of τ. We illustrate this rather informal explanation on a short example.

Example 5.3.2. *Consider the following $\mathcal{ALCHOIF}^+$ KB $\mathcal{K} = (\mathcal{T}, \{s\}, \{s(a, b)\})$, where the TBox $\mathcal{T}$ consists of the following axioms:*

$$\begin{aligned}
&A_1 \sqsubseteq \exists r_1.A_2, \qquad A_2 \sqsubseteq \exists r_3.\{a\},\\
&A_1 \sqsubseteq \exists r_2.A_3, \qquad A_3 \sqsubseteq \exists r_4.\{b\},\\
&\exists P.(\{x\} \sqcap \exists s.\{y\}) \sqcap \exists R.\{y\} \sqsubseteq B,\\
&\{x\} \sqsubseteq \forall S.\{x\},\\
&\mathsf{func}(r_1), \quad \mathsf{func}(r_2), \quad \mathsf{func}(r_3), \quad \mathsf{func}(r_4),
\end{aligned}$$

where $P = A_1? \circ r_1 \circ A_2? \circ r_3 \circ \{a\}?$, $R = A_1? \circ r_3 \circ A_3? \circ r_4 \circ \{b\}?$, $S = \top? \circ r_3^- \circ A_2? \circ r_1^- \circ A_1? \circ r_2 \circ A_3? \circ r_4 \circ \top? \circ s^- \circ \top?$.

Let $\tau = (\{A_1, B, \top\}, \rho, \pi_\exists, \pi_\forall)$ be a tile for $\mathcal{K}$ where ρ, $\pi_\exists$, and $\pi_\forall$ are given as follows:

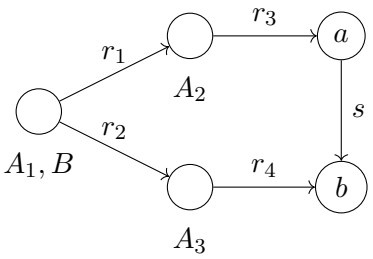
$$\begin{aligned}
\rho = \{\ &(\{r_1\}, \{A_2, \top\}, \{(r_2 \circ \{a\}? \circ \varepsilon, \{a\})\},\\
&\{(r_1^- \circ A_1? \circ r_2 \circ A_3? \circ r_4 \circ \top? \circ s^- \circ \top? \circ \varepsilon, \{a\})\}),\\
&(\{r_2\}, \{A_3, \top\}, \{(r_4 \circ \{b\}? \circ \varepsilon, \{b\})\}, \{(r_4 \circ \top? \circ s^- \circ \top? \circ \varepsilon, \{a\})\})\},\\
\pi_\exists = \{\ &(r_1 \circ A_2? \circ r_2 \circ \{a\}? \circ \varepsilon, \{a\}),\\
&(r_3 \circ A_3? \circ r_4 \circ \{b\}? \circ \varepsilon, \{b\})\},\\
\pi_\forall = \{\ &(r_2 \circ A_3? \circ r_4 \circ \top? \circ s^- \circ \top? \circ \varepsilon, \{a\})\}
\end{aligned}$$

In a model $\mathcal{I}$ of $\mathcal{K}$, a domain element $d \in \Delta^\mathcal{I}$ that is an instance of $\mathcal{T}$ has the following properties:

- *d has the unary type $\{A_1, B, \top\}$, i.e., $d \in A_1^\mathcal{I}$, $d \in B^\mathcal{I}$ and $d \notin D^\mathcal{I}$, for all $D \in \mathsf{N}_\mathsf{C}^+(\mathcal{K}) \setminus \{A_1, B, \top\}$.*
- *d can reach the individual a via some the complex role $A_1? \circ r_1 \circ A_2? \circ r_3 \circ \{a\}?$.*

- *d can reach the individual b via some the complex role $A_1? \circ r_3 \circ A_3? \circ r_4 \circ \{b\}?$.*
- *The only element that d can reach via the complex role $r_2 \circ A_3? \circ r_4 \circ \top? \circ s^- \circ \top?$ is the individual a.*
- *d has an r_1-successor e with the unary type $\{A_2, \top\}$ that can reach the individual a via some complex role $r_3 \circ \{a\}?$. Moreover, a is the only element that e can reach via $r_1^- \circ A_1? \circ r_2 \circ A_3? \circ r_4 \circ \top? \circ s^- \circ \top?$.*
- *d has an r_2-successor e' with the unary type $\{A_3, \top\}$ that can reach the individual b via some complex role $r_4 \circ \{b\}?$. Moreover, a is the only element that e' can reach via $r_4 \circ \top? \circ s^- \circ \top$.*

For example, in the model $\mathcal{I}$ of $\mathcal{K}$ given below, we can say that the left-most element is an instance of τ.



We are now ready to give the formal definition of tiles for $\mathcal{ALCHOIF}^+$ KBs.

Definition 5.3.3. *Given a KB $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$, a* tile *for $\mathcal{K}$ is a tuple $\tau = (T, \rho, \pi_\exists, \pi_\forall)$, where $T \in \mathit{Types}(\mathcal{K})$, $\pi_\exists \subseteq \Pi_\exists(\mathcal{K})$, $\pi_\forall \subseteq \Pi_\forall(\mathcal{K})$ and ρ is a set of tuples $(R, T', \pi'_\exists, \pi'_\forall)$, where $R \subseteq \mathrm{N}_R^+(\mathcal{K})$, $T' \in \mathit{Types}(\mathcal{K})$, $\pi'_\exists \subseteq \Pi_\forall(\mathcal{K})$, $\pi'_\forall \subseteq \Pi_\forall(\mathcal{K})$ and the following conditions are satisfied:*

TF^+1. $|\rho| \leq |\mathcal{T}|$

TF^+2. If $B_1 \sqcap \cdots \sqcap B_{k-1} \sqsubseteq B_k \sqcup \cdots \sqcup B_m \in \mathcal{T}$ and $\{B_1, \ldots, B_{k-1}\} \subseteq T$, then $\{B_k, \ldots, B_m\} \cap T \neq \emptyset$

TF^+3. If $A \sqsubseteq \exists r.B \in \mathcal{T}$ and $A \in T$, then there is $(R, T', \pi'_\exists, \pi'_\forall) \in \rho$ such that $r \in R$ and $B \in T'$

TF^+4. For all $(R, T', \pi'_\exists, \pi'_\forall) \in \rho$, the following hold:

- *(a) If $A \sqsubseteq \forall r.B \in \mathcal{T}$, $A \in T$ and $r \in R$, then $B \in T'$*
- *(b) If $A \sqsubseteq \forall r.B \in \mathcal{T}$, $A \in T'$ and $r^- \in R$, then $B \in T$*
- *(c) If $r \sqsubseteq s \in \mathcal{T}$ and $r \in R$, then $s \in R$*

TF^+5. *If* $\mathsf{func}(r) \in \mathcal{T}$, *then* $|\{(R, T', \pi'_\exists, \pi'_\forall) \in \rho : r \in R\}| \leq 1$

TF^+6. *If* $A(b) \in \mathcal{A}$ *and* $\{b\} \in T$, *then* $A \in T$

TF^+7. *If* $\neg A(b) \in \mathcal{A}$ *and* $\{b\} \in T$, *then* $A \notin T$

TF^+8. *For all* $(R, T', \pi'_\exists, \pi'_\forall) \in \rho$, *the following hold:*

- *(a) If* $p(a, b) \in \mathcal{A}$, $\{p \sqsubseteq r, \mathsf{func}(r)\} \subseteq \mathcal{T}$, $\{a\} \in T$ *and* $r \in R$, *then* $\{b\} \in T'$
- *(b) If* $p(a, b) \in \mathcal{A}$, $\{p \sqsubseteq r, \mathsf{func}(r^-)\} \subseteq \mathcal{T}$, $\{b\} \in T$, *and* $r^- \in R$, *then* $\{a\} \in T'$
- *(c) If* $\neg p(a, b) \in \mathcal{A}$, $r \sqsubseteq p \in \mathcal{T}$, $\{a\} \in T$, *and* $r \in R$, *then* $\{b\} \notin T'$
- *(d) If* $\neg p(a, b) \in \mathcal{A}$, $r \sqsubseteq p^- \in \mathcal{T}$, $\{b\} \in T$, *and* $r \in R$, *then* $\{a\} \notin T'$

TF^+9. *If* $A \in \Sigma \cap \mathrm{N}_C$ *and* $A \in T$, *then there exists* $c \in \mathrm{N}_I$ *such that* $\{c\} \in T$ *and* $A(c) \in \mathcal{A}$

TF^+10. *If* $r \in \Sigma \cap \mathrm{N}_R$, *then for all* $(R, T', \pi'_\exists, \pi'_\forall) \in \rho$ *with* $r \in R$, *there exist* $c, d \in \mathrm{N}_I$ *such that* $\{c\} \in T$, $\{d\} \in T'$ *and* $r(c, d) \in \mathcal{A}$.

TF^+11. $|\pi_\exists| \leq |\mathsf{Paths}_\exists(\mathcal{K})|$

TF^+12. $|\pi_\forall| \leq |\mathsf{Paths}_\forall(\mathcal{K})|$

TF^+13. *If* $\{a\} \in \mathsf{N}_C^+(\mathcal{K})$, $\{a\} \in T$, *then* $(\varepsilon, \{a\}) \in \pi_\exists$

TF^+14. *If* $\exists A_1? \circ P.(\{x\} \sqcap \exists s.\{y\}) \sqcap \exists A_2? \circ R.\{y\} \sqsubseteq A \in \mathcal{T}$, $\{\{a\}, \{b\}\} \subseteq \mathsf{N}_C^+(\mathcal{K})$, $\{A_1, A_2\} \subseteq T$, $\{(P, \{a\}), (R, \{b\})\} \subseteq \pi_\exists$ *and* $s(a, b) \in \mathcal{A}$, *then* $A \in \mathcal{T}$.

TF^+15. *If* $\exists A_1? \circ P.(\{x\} \sqcap \neg\exists s.\{y\}) \sqcap \exists A_2? \circ R.\{y\} \sqsubseteq A \in \mathcal{T}$, $\{\{a\}, \{b\}\} \subseteq \mathsf{N}_C^+(\mathcal{K})$, $\{A_1, A_2\} \subseteq T$, $\{(P, \{a\}), (R, \{b\})\} \subseteq \pi_\exists$ *and* $s(a, b) \notin \mathcal{A}$, *then* $A \in \mathcal{T}$.

TF^+16. *If* $\{x\} \sqsubseteq \forall A? \circ P.\{x\} \in \mathcal{T}$, $\{a\} \in \mathsf{N}_C^+(\mathcal{K})$, *and* $\{\{a\}, A\} \in T$, *then either* $(P, \{a\}) \in \pi_\forall$ *or* $(P, \bot) \in \pi_\forall$.

TF^+17. *If* $(\varepsilon, C) \in \pi_\forall$, *then* $C \in T$.

TF^+18. *For all* $(R, T', \pi'_\exists, \pi'_\forall) \in \rho$

- *(a) If* $(r \circ A? \circ P, \{a\}) \in \Pi_\exists(\mathcal{K})$, *and* $r \in R$, $A \in T'$, *and* $(P, \{a\}) \in \pi'_\exists$, *then* $(r \circ A? \circ P, \{a\}) \in \pi_\exists$
- *(b) If* $(r \circ A? \circ P, \{a\}) \in \Pi_\exists(\mathcal{K})$, *and* $r^- \in R$, $A \in T$, *and* $(P, \{a\}) \in \pi_\exists$, *then* $(r \circ A? \circ P, \{a\}) \in \pi'_\exists$.
- *(c) if* $(r \circ A? \circ P, C) \in \pi_\forall$, $r \in R$ *and* $A \in T'$, *either* $(P, C) \in \pi'_\forall$ *or* $(P, \bot) \in \pi'_\forall$.
- *(d) if* $(r \circ A? \circ P, C) \in \pi'_\forall$, $r^- \in R$ *and* $A \in T$, *either* $(P, C) \in \pi_\forall$ *or* $(P, \bot) \in \pi_\forall$.
- *(e) If* $\{x\} \sqcap A \sqsubseteq \forall s.\{x\} \in \mathcal{T}$, $\{a\} \in \mathsf{N}_C^+(\mathcal{K})$, $\{\{a\}, A\} \subseteq \mathcal{T}$ *and* $\{s, s^-\} \cap R \neq \emptyset$, *then* $\{a\} \notin \mathcal{T}$

TF$^+$19. If $p(a,b) \in \mathcal{A}$, $p \sqsubseteq r \in \mathcal{T}$, $\mathsf{func}(r) \in \mathcal{T}$, $\{a\} \in T$, *there is some* $(R, T', \pi'_\exists, \pi'_\forall) \in \rho$ *s.t.* $r \in R$ *and* $\{b\} \in T'$.

TF$^+$20. If $p(a,b) \in \mathcal{A}$, $p^- \sqsubseteq r \in \mathcal{T}$, $\mathsf{func}(r) \in \mathcal{T}$, $\{b\} \in T$, *there is some* $(R, T', \pi'_\exists, \pi'_\forall) \in \rho$ *s.t.* $r \in R$ *and* $\{a\} \in T'$.

TF$^+$21. If $\{a\} \in \mathsf{N}_\mathsf{C}^+(\mathcal{K})$, $s(a,a) \in \mathcal{A}$ *and* $\{x\} \sqcap A \sqsubseteq \forall s.\neg\{x\} \in \mathcal{T}$, *then* $\{\{a\}, A\} \not\subseteq T$.

As before, we explain the intuitions behind the conditions placed on tiles. Conditions TF$^+$1-TF$^+$10 are inherited from the characterization of $\mathcal{ALCHOIF}$ (see Definition 3.3.1) and ensure that the description of d is locally consistent with $\mathcal{ALCHOIF}$ axioms occurring in $\mathcal{T}$ and the assertions in $\mathcal{A}$ under the closed predicates Σ. We further add conditions TF$^+$11-TF$^+$21 to support axioms of the forms (A4)-(A7). Namely, TF$^+$11 and TF$^+$12 ensure that our extended tiles are still 'small', i.e. polynomial in the size of $(\mathcal{T}, \Sigma)$ but constant in the size of $\mathcal{A}$. To support axioms of type (A4) and (A5) we do the following. Assume $\exists P.(\{x\} \sqcap \exists s.\{y\}) \sqcap \exists R.\{y\} \in \mathcal{T}$, where $P = A_0? \circ r_1 \circ A_1? \circ \cdots \circ r_n \circ A_n?$. We start from the nominals and add a pair $(\varepsilon, \{a\}) \in \pi_\exists$ whenever $\{a\}$ is in the unary type of $\tau = (t, \rho, \pi_\exists, \pi_\forall)$ (TF$^+$13). Now, if we have a tile $\tau' = (T', \rho', \pi'_\exists, \pi'_\forall)$ such that the r_n-connection to τ is enforced in some way, $(r_n, \{a\}) \in \pi'_\exists$ must exist. This backwards labelling from the nominals is repeated and we have that whenever a tile has a forced S-connection to a τ, there is a pair $(S, \{a\})$ in its $\pi_\exists$ component, for each subrole of S of P (TF$^+$18 (a)-(b)). Finally, TF$^+$14 requires that if a tile is aware there is a P path to some nominal $\{a\}$ and an R path to another nominal $\{b\}$ s.t. $s(a,b)$ is in the ABox (recall s is a closed role), then A must be in its unary type. Axiom of the form (A5) is handled similarly using TF$^+$15. Further, we also handle axiom of the form (A6) similarly, but for each such axiom $\{x\} \sqsubseteq \forall P.\{x\}$, where $P = A_0? \circ r_1 \circ A_1? \circ \cdots \circ r_n \circ A_n?$, we start by adding either $(P, \{a\}) \in \pi_\forall$ or $(P, \bot)$, for each tile $\tau = (t, \rho, \pi_\exists, \pi_\forall)$ s.t. $\{a\}$ and A_0 occur in T (TF$^+$16). Intuitively, this ensures that starting from the nominal $\{a\}$, either all paths of shape P lead back to $\{a\}$ or no such path exists. Then, using TF$^+$18 (c)-(d), we label all known r_n-successors of τ with either $(R, \{a\})$ in their $\pi_\forall$ components denoting that all R-paths from domain elements with this description must end in a, or with $(R, \bot)$ stating that domain elements with this description have no R-paths to any other elements, where $R = r_2 \circ A_2? \circ \cdots \circ r_n \circ A_n?$. TF$^+$17 then simply checks that all P-paths from some nominal $\{a\}$ also end in $\{a\}$, i.e., that the axiom is satisfied. Condition TF$^+$18 (e) is rather straightforward and it handles axiom of the form (A7). Conditions TF$^+$19 and TF$^+$20 ensure that all connections involving functional roles are stored within the tiles. Finally, condition TF$^+$21 deals with axioms of type (A7) by disallowing for all nominals $\{a\} \in \mathsf{N}_\mathsf{C}^+(\mathcal{K})$ that both $\{a\}$ and A are in the unary type of some tile, if $\{x\} \sqcap A \sqsubseteq \forall s.\neg\{x\}$ and we know that $s(a,a) \in \mathcal{A}$. Recall that, since s is a closed role, and so the only way to have (a,a) participate in s is if this is asserted by the ABox.

We next also adapt the definition of mosaics.

Definition 5.3.4. *Let $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$ be a KB. A* mosaic *for $\mathcal{K}$ is a function $N : Tiles(\mathcal{K}) \to \mathbb{N}^*$ such that:*

MF$^+$1. For every $\{c\} \in \mathsf{N}_C^+(\mathcal{K})$: $\sum_{\tau=(T,\rho,\pi_\exists,\pi_\forall)\in Tiles(\mathcal{K}),\ \{c\}\in T} N(\tau) = 1$

MF$^+$2. The following inequality is satisfied: $\sum_{\tau\in Tiles(\mathcal{K})} N(\tau) \geq 1$

MF$^+$3. For all $T, T' \in Types(\mathcal{K})$, $R \subseteq \mathrm{N}_R^+(\mathcal{K})$ with $r \in R$ and $\mathsf{func}(r^-) \in \mathcal{T}$, every $\pi_\exists, \pi'_\exists \subseteq \Pi_\exists(\mathcal{K})$, $\pi_\forall, \pi'_\forall \subseteq \Pi_\forall(\mathcal{K})$, the following holds:

$$\sum_{\substack{\tau=(T,\rho,\pi_\exists,\pi_\forall)\in Tiles(\mathcal{K}),\\ (R,T',\pi'_\exists,\pi'_\forall)\in\rho}} N(\tau) \leq \sum_{\substack{\tau'=(T',\rho',\pi'_\exists,\pi'_\forall)\in Tiles(\mathcal{K}),\\ (R^-,T,\pi_\exists,\pi_\forall)\in\rho'}} N(\tau')$$

MF$^+$4. For all $\tau = (T, \rho, \pi_\exists, \pi_\forall) \in Tiles(\mathcal{K})$ and $(R, T', \pi'_\exists, \pi'_\forall) \in \rho$ the following holds: if $N(\tau) > 0$, then there exists ρ' such that $\tau' = (T', \rho', \pi'_\exists, \pi'_\forall) \in Tiles(K)$ and $N(\tau') > 0$.

MF$^+$5. For all $\{a\}, \{b\} \in \mathsf{N}_C^+(\mathcal{K})$ and all $A, B \in \mathrm{N}_C(\mathcal{K})$, if there exist $p, r \in \mathrm{N}_R^+(\mathcal{K})$ for which any of the following conditions hold:

- *(a) $p(a, b) \in \mathcal{A}$, $p \sqsubseteq r \in \mathcal{T}$ and $A \sqsubseteq \forall r.B \in \mathcal{T}$,*
- *(b) $p(b, a) \in \mathcal{A}$, $p \sqsubseteq r^- \in \mathcal{T}$ and $A \sqsubseteq \forall r.B \in \mathcal{T}$,*
- *(c) $p(a, b) \in \mathcal{A}$, $p \sqsubseteq r \in \mathcal{T}$ and $A \sqsubseteq \exists r.B \in \mathcal{T}$ and $\mathsf{func}(r) \in \mathcal{T}$, or*
- *(d) $p(b, a) \in \mathcal{A}$, $p \sqsubseteq r^- \in \mathcal{T}$ and $A \sqsubseteq \exists r.B \in \mathcal{T}$ and $\mathsf{func}(r) \in \mathcal{T}$,*

we have that the following implication holds:

$$\sum_{\substack{\tau=(T,\rho,\pi_\exists,\pi_\forall)\in Tiles(\mathcal{K}),\\ \{a\}\in T, A\in T}} N(\tau) > 0 \implies \sum_{\substack{\tau'=(T',\rho',\pi'_\exists,\pi'_\forall)\in Tiles(\mathcal{K}),\\ \{b\}\in T', B\in T'}} N(\tau') > 0$$

MF$^+$6. For all $p(a, b) \in \mathcal{A}$ and $\tau = (T, \rho, \pi_\exists, \pi_\forall) \in Tiles(\mathcal{K})$ with $\{a\} \in T$, $N(\tau) > 0$ implies that there exists a tile $\tau' = (T', \rho', \pi'_\exists, \pi'_\forall)$ with $N(\tau') > 0$ s.t.

- *(a) $\{b\} \in T'$,*
- *(b) for all $(r \circ A? \circ P, C) \in \Pi_\exists(\mathcal{K})$, $p \sqsubseteq r \in \mathcal{T}$, $(P, C) \in \pi'_\exists$, and $A \in T'$, we have $(r \circ A? \circ P, C) \in \pi_\exists$,*
- *(c) for all $(r \circ A? \circ P, C) \in \Pi_\exists(\mathcal{K})$, $p^- \sqsubseteq r \in \mathcal{T}$, $(P, C) \in \pi_\exists$, and $A \in T$, we have $(r \circ A? \circ P, C) \in \pi'_\exists$,*

(d) for all $(r \circ A? \circ P, C) \in \pi_\forall$, $p \sqsubseteq r \in \mathcal{T}$, *and* $A \in T'$, *we have*

$$\{(P, C), (P, \bot)\} \cap \pi'_\forall \neq \emptyset,$$

(e) for all $(r \circ A? \circ P, C) \in \pi'_\forall$, $p^- \sqsubseteq r \in \mathcal{T}$, *and* $A \in T$, *we have*

$$\{(P, C), (P, \bot)\} \cap \pi_\forall \neq \emptyset.$$

In the definition above, MF$^+$1-MF$^+$5 are direct adaptations from the characterization of $\mathcal{ALCHOIF}$ to include the extended notion of tiles, so we do not discuss them here. Finally, the new condition MF$^+$6 ensures that if a connection is enforced between two elements due to some ABox assertion $p(a, b)$ and this is *not* reflected in the ρ component of either tile, it must still be the case that $\pi_\exists$ and $\pi_\forall$ components of these elements are compatible, i.e., the connection $p(a, b)$ is taken into account.

Example 5.3.5. *Consider the* $\mathcal{ALCHOIF}^+$ *KB* $\mathcal{K} = (\mathcal{T}, \{s\}, \{s(a, b)\})$ *from Example 5.3.2 and consider the following tiles for* $\mathcal{K}$*:*

$\tau_a = (\{\{a\}, \top\}, \emptyset, \pi_{\exists_a}, \pi_{\forall_a})$, *where* $\pi_{\exists_a}$ *and* $\pi_{\forall_a}$ *are given as follows:*

$$\pi_{\exists_a} = \{(\varepsilon, \{a\})\}$$
$$\pi_{\forall_a} = \{(r_3^- \circ A_2? \circ r_1^- \circ A_1? \circ r_2 \circ A_3? \circ r_4 \circ \top? \circ s^- \circ \top? \circ \varepsilon, \{a\})\}$$

$\tau_b = (\{\{b\}, \top\}, \emptyset, \pi_{\exists_b}, \pi_{\forall_b})$, *where* $\pi_{\exists_b}$ *and* $\pi_{\forall_b}$ *are given as follows:*

$$\pi_{\exists_b} = \{(\varepsilon, \{b\})\}$$
$$\pi_{\forall_b} = \{(r_3^- \circ A_2? \circ r_1^- \circ A_1? \circ r_2 \circ A_3? \circ r_4 \circ \top? \circ s^- \circ \top? \circ \varepsilon, \bot), (s^- \circ \top? \circ \varepsilon, \{a\})\}$$

$\tau_1 = (\{A_2, \top\}, \rho_1, \pi_{\exists_1}, \pi_{\forall_1})$, *where* ρ_1, $\pi_{\exists_1}$, *and* $\pi_{\forall_1}$ *are given as follows:*

$$\rho_1 = \{(\{r_3\}, \{\{a\}, \top\}, \pi_{\exists_a}, \pi_{\forall_a})\}, \qquad \pi_{\exists_1} = \{(r_2 \circ \{a\}? \circ \varepsilon, \{a\})\},$$
$$\pi_{\forall_1} = \{(r_1^- \circ A_1? \circ r_2 \circ A_3? \circ r_4 \circ \top? \circ s^- \circ \top? \circ \varepsilon, \{a\})\}$$

$\tau_2 = (\{A_3, \top\}, \rho_2, \pi_{\exists_2}, \pi_{\forall_2})$, *where* ρ_2, $\pi_{\exists_2}$, *and* $\pi_{\forall_2}$ *are given as follows:*

$$\rho_2 = \{(\{r_4\}, \{\{b\}, \top\}, \pi_{\exists_b}, \pi_{\forall_b}\}, \qquad \pi_{\exists_2} = \{(r_4 \circ \{b\}? \circ \varepsilon, \{b\})\},$$
$$\pi_{\forall_2} = \{(r_4 \circ \top? \circ s^- \circ \top? \circ \varepsilon, \{a\})\}$$

$\tau_3 = (\{A_1, B, \top\}, \rho_3, \pi_{\exists_3}, \pi_{\forall_3})$, *where* ρ_3, $\pi_{\exists_3}$, *and* $\pi_{\forall_3}$ *are given as follows:*

$$\rho_3 = \{(\{r_1\}, \{A_2, \top\}, \pi_{\exists_1}, \pi_{\forall_1}), (\{r_2\}, \{A_3, \top\}, \pi_{\exists_2}, \pi_{\forall_2})\},$$
$$\pi_{\exists_3} = \{(r_1 \circ A_2? \circ r_2 \circ \{a\}? \circ \varepsilon, \{a\}), (r_3 \circ A_3? \circ r_4 \circ \{b\}? \circ \varepsilon, \{b\})\},$$
$$\pi_{\forall_3} = \{(r_2 \circ A_3? \circ r_4 \circ \top? \circ s^- \circ \top? \circ \varepsilon, \{a\})\}$$

Then the following function is a mosaic for $\mathcal{K}$ *and it corresponds to the model* $\mathcal{I}$ *from Example 5.3.2:*

$$N = \begin{cases} 1, & \text{if } \tau \in \{\tau_a, \tau_b, \tau_1, \tau_2, \tau_3\}, \\ 0, & \text{otherwise.} \end{cases}$$

We say that an $\mathcal{ALCHOIF}^+$ KB $\mathcal{K}$ *respects role inclusions* if it satisfies the two condition in Definition 3.3.3. The following result establishes the connection between the existence of mosaics and the satisfiability of $\mathcal{ALCHOIF}^+$ KBs with closed predicates.

Theorem 5.3.6. *Let $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$ be a KB. $\mathcal{K}$ is satisfiable if and only if $\mathcal{K}$ respects role inclusions and there exists a mosaic for $\mathcal{K}$.*

Although the extended notion of tiles makes the proof of the theorem above more involved, at its essence, this proof is very similar to the proofs of Theorems 3.2.12 and 3.3.4 and is therefore delegated to the appendix.

Consider an $\mathcal{ALCHOIF}^+$ KB $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$. Having established the connection between mosaic existence and satisfiability of $\mathcal{K}$, we can follow the same steps as before to build an enriched system of integer linear inequalities whose solutions over $\mathbb{N}^*$ correspond to the mosaics of $\mathcal{K}$ and then use integer programming techniques to decide whether $\mathcal{K}$ is satisfiable. Regarding the size of the obtained system, we note that, due to conditions TF^+1, TF^+11, and TF^+12, as well as the fact that the sets $\Pi_\exists(\mathcal{K})$ and $\Pi_\forall(\mathcal{K})$ are polynomial in the size of $\mathcal{K}$, the number of tiles for $\mathcal{K}$ is exponential in the size of $\mathcal{K}$, but only polynomial if $\mathcal{T}$ and Σ are considered fixed. Moreover, it is easy to see that each condition in the mosaic introduces at most an exponential number of inequalities/implications in the size of $\mathcal{K}$ (polynomial, if $\mathcal{T}$ and Σ are fixed), and the same bounds also apply to the size of each of these inequalities, whether they are stand-alone or they occur in some implication, as well as the whole enriched system for $\mathcal{K}$. As we can decide the existence of integer solutions to enriched systems in NP (see Proposition 3.2.22), the result in Theorem 5.2.9 follows.

5.4 Encoding the Turing Machine

In the previous section, we showed that inconsistency queries mediated by $\mathcal{ALCHOIF}^+$ ontologies with closed predicates are coNP complete in data complexity. We next present the main result of this chapter which states that the same query language is also capable of expressing all coNP-computable GBQs.

Theorem 5.4.1 (Main result)**.** *Assume a signature Σ and a GBQ Q over Σ that belongs to coNP. Then there is a TBox $\mathcal{T}$ in $\mathcal{ALCHOIF}^+$ such that the Boolean inconsistency query OMQ $q = (\mathcal{T}, \Sigma)$ has the following property: for all Σ-ABoxes $\mathcal{A}$, $Q(\mathcal{A}) = q(\mathcal{A})$.*

The rest of this section serves as a proof sketch for the theorem above. Let Q be a GBQ over some signature Σ that is in coNP. According to Proposition 5.1.5, there is a nondeterministic Turing machine M that decides the language $\{enc^\Sigma(\mathcal{A}) : Q(\mathcal{A}) = 0, \mathcal{A} \text{ is a } \Sigma\text{-ABox}\}$ and its running time is bounded by n^k, where k is a constant and n is the size of the input word. We show that we can come up with a TBox $\mathcal{T}_M$ such that for the ontology-mediated inconsistency query $Q_M = (\mathcal{T}_M, \Sigma)$, $Q_M(\mathcal{A}) = Q(\mathcal{A})$, for all Σ-ABoxes $\mathcal{A}$. The basic idea is to craft $\mathcal{T}_M$ in a way that ensures that all models of $(\mathcal{T}, \Sigma, \mathcal{A})$

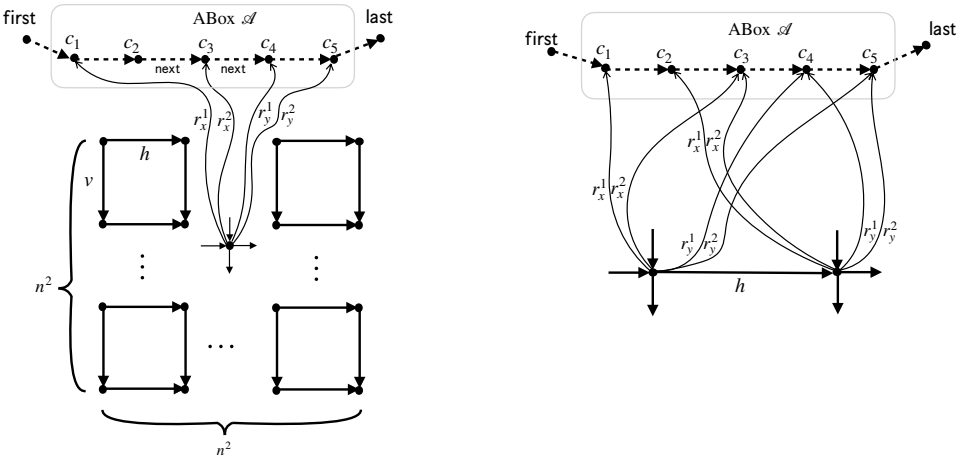


Figure 5.1: Construction of the $n^k \times n^k$ grid, for $k = 2$. Left: Assigning coordinates to grid nodes. Right: Propagation of coordinates along horizontal successors.

contain a grid structure of size $n^k \times n^k$. We then use this grid to simulate the given Turing machine M as follows. The first row of the grid stores the initial configuration of M while each subsequent row stores the next configuration in some computation of M. Finally, we eliminate those computations that do not end in acceptance of the word. As a result, we have that each model of $(\mathcal{T}_M, \Sigma, \mathcal{A})$ corresponds to a computation of M that accepts $enc^\Sigma(\mathcal{A})$, and vice versa: every computation of M that ends in acceptance of $enc^\Sigma(\mathcal{A})$ corresponds to some model of $(\mathcal{T}_M, \Sigma, \mathcal{A})$, for all Σ-ABoxes $\mathcal{A}$. Thus, checking whether M accepts $enc^\Sigma(\mathcal{A})$ boils down to checking whether $(\mathcal{T}_M, \Sigma, \mathcal{A})$ is unsatisfiable, which is equivalent answering the inconsistency query Q_M.

We now begin with our construction. In the rest of this section we assume we are given a GBQ in the form of a nondeterministic Turing machine $M = (\Gamma^\Sigma, Q, \delta, q_0, q_{acc}, q_{rej})$ and an integer constant k.

5.4.1 Constructing the $n^k \times n^k$ Grid

Consider an arbitrary ABox $\mathcal{A}$ over some signature Σ. We next show how to build a KB $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$ s.t. that each model $\mathcal{I}$ of $\mathcal{K}$ contains a $n^k \times n^k$ grid formed by the domain elements, where n is the number of known individuals (i.e., the number of individuals occurring in $\mathcal{A}$ plus two special constants first and last).

We begin by generating different domain elements that serve as grid nodes. Each such grid node is associated with two words of length k over the known individuals that serve as its x- and y-coordinate in the grid. This is accomplished using two sets of functional roles: $r_x^1, \dots r_x^k$ and $r_y^1, \dots r_y^k$. We say that a domain element e has an x-coordinate $c_1 c_2 \cdots c_k$, if (e, c_i) participates in r_x^i, for each $i, 1 \leq i \leq k$. The y-coordinate of e is defined analogously. This is illustrated in Figure 5.1, left. In the first step of the construction, we let the special individual first be the origin of the grid and set its x- and y- coordinates to

$\mathsf{first} \cdots \mathsf{first}$. To generate the remainder of the grid nodes, we add axioms that create a binary tree rooted in first using two roles h and v, denoting horizontal and vertical successors, respectively. We next assign the x- and y-coordinates to each grid node in the tree making sure that they respect a certain pattern. To do this, we use a linear order over the known individuals that we can can easily generate by using first and last as the designated first and last elements and guessing the remaining part of the successor relation, encoded using the role . We then lift this linear order to words of length k over the available individuals and add axioms that require that for each grid node e with the horizontal successor e', the x-coordinate of e' is the successor of the x-coordinate of e with respect to the generalized linear order, while the y-coordinate remains unchanged. We then do a similar thing with the vertical successor e'' of e. Namely, the y-coordinate of e'' is the successor of the y-coordinate of e with respect to the generalized linear order, while the x-coordinate stays the same. Figure 5.1, on the right illustrates this. It is not hard to see that all possible pairs of x- and y-coordinates occur within this tree. Now, the only thing that is left to do is to merge nodes with same coordinates. This is easy: we simply let the special individual last be the only grid node with $\mathsf{last} \cdots \mathsf{last}$ as its x- and y-coordinate. Propagating backwards from last while relying on the fact that each grid node has at most one h- and at most one v-predecessor, we can easily see that each different combination of the coordinates occurs exactly one time – thus we have n^{2k} different grid nodes. Moreover, the way we assigned their coordinates ensures that they form a proper grid. We next detail the construction by providing all the relevant axioms.

Collect constants from $\mathcal{A}$. We first collect in Adom all the individuals occurring in $\mathcal{A}$:

$$\mathsf{Adom} \equiv \bigsqcup_{A \in \mathsf{N}_C^+ \cap \Sigma} A \sqcup \bigsqcup_{r \in \mathsf{N}_R^+ \cap \Sigma} (\exists r \sqcup \exists r^-)$$

Guess a linear order. We next add the axioms that guess a linear order over the known individuals, stored using the concept name Node. We use two individuals First and Last as designated first and last elements in this linear order. The role stores the successor relation, and $\mathsf{lessThan}$ is a role that stores the induced "less than" relation.

$\mathsf{Node} \equiv \mathsf{Adom} \sqcup \{\mathsf{First}\} \sqcup \{\mathsf{Last}\}$	$\{x\} \sqcap \mathsf{Node} \sqsubseteq \forall \mathsf{lessThan}.\neg\{x\}$
$\mathsf{Node} \sqsubseteq \exists \mathsf{next}.\mathsf{Node} \sqcup \{\mathsf{Last}\}$	$\mathsf{next} \sqsubseteq \mathsf{lessThan}$
$\mathsf{Node} \sqsubseteq \exists \mathsf{next}^-.\mathsf{Node} \sqcup \{\mathsf{First}\}$	$(\mathsf{trans}\, \mathsf{lessThan})$
$\mathsf{func}(\mathsf{next})$	$\exists \mathsf{lessThan} \sqsubseteq \mathsf{Node}$
$\mathsf{func}(\mathsf{next}^-)$	$\exists \mathsf{lessThan}^- \sqsubseteq \mathsf{Node}$

Axioms on the left-hand side are responsible for guessing the successor relation of the linear order that is being generated. They ensure that all individuals except for the last one have a successor, and all individuals except for the first one have a predecessor. Moreover, successors and predecessors must be unique. Axioms on the right-hand side say that the transitive closure of next contains no cycles, meaning that we have a proper

linear order. The last two axioms serve as guards to ensure that a transitivity assertion is made over a restricted role.

Creating the grid structure. To create a $n^k \times n^k$ grid, we take the approach above and add, for all $1 \leq i \leq k$, the following axioms that create a binary tree routed in first using h and v:

$$\begin{aligned}
&\mathsf{GridNode} \sqsubseteq \textstyle\bigsqcap_{1\leq i\leq k}(\exists r_x^i.\mathsf{Node} \sqcap \exists r_y^i.\mathsf{Node}) && \mathsf{func}(h)\\
&\{\mathsf{First}\} \equiv \mathsf{GridNode} \sqcap \textstyle\bigsqcap_{1\leq i\leq k}(\exists r_x^i.\{\mathsf{First}\} \sqcap \exists r_y^i.\{\mathsf{First}\}) && \mathsf{func}(h^-)\\
&\mathsf{GridNode} \sqsubseteq \exists h.\mathsf{GridNode} \sqcup (\textstyle\bigsqcap_{1\leq i\leq k} \exists r_x^i.\{\mathsf{Last}\}) && \mathsf{func}(v)\\
&\mathsf{GridNode} \sqsubseteq \exists v.\mathsf{GridNode} \sqcup (\textstyle\bigsqcap_{1\leq i\leq k} \exists r_y^i.\{\mathsf{Last}\}) && \mathsf{func}(v^-)\\
&\textstyle\bigsqcap_{1\leq i\leq k} \exists r_x^i.\{\mathsf{Last}\} \sqsubseteq \neg\exists h.\top && \mathsf{func}(r_x^i)\\
&\textstyle\bigsqcap_{1\leq i\leq k} \exists r_y^i.\{\mathsf{Last}\} \sqsubseteq \neg\exists v.\top && \mathsf{func}(r_y^i)
\end{aligned}$$

The first axiom on the left-hand side states that every grid node has 2k pointers to the known individuals using functional roles r_x^i and r_y^i, $1 \leq i \leq k$ that encode its x- and y-coordinates. The second axiom on the left-hand side sets first as a designated origin point with x- and y-coordinates $\mathsf{first} \cdot \mathsf{first}$. The rest of the axioms simply create the tree.

We next make sure that the coordinates align, i.e., if e' is an h-successor of e, then the y-coordinates of e and e' coincide, while the x-coordinate of e' is a successor of the x-coordinate of e w.r.t. to the linear order in extended to words of length k. For example, if the x-coordinate of e is $c_k \cdots c_i \cdot \mathsf{last} \cdots \mathsf{last}$, where $c_i \neq 1$, then the x-coordinate of d is $c_k \cdots c_i' \cdot \mathsf{first} \cdots \mathsf{first}$, where c_i' is the successor of c_i according to the given linear order. We now define the axioms that do this and add for all i, $1 \leq i \leq k$:

$$\begin{aligned}
&\mathsf{GridNode} \sqcap \neg\exists r_x^i.\{\mathsf{Last}\} \sqcap \textstyle\bigsqcap_{1\leq j\leq i} \exists r_x^j.\{\mathsf{Last}\} \sqsubseteq \mathsf{IncrX}_i\\
&\mathsf{IncrX}_i \sqsubseteq \forall h.(\textstyle\bigsqcap_{1\leq j\leq i} \exists r_x^j.\{\mathsf{First}\})\\
&\{\mathsf{First}\} \sqsubseteq \forall (r_x^i)^-.\forall h^-.\forall r_x^i.\{\mathsf{Last}\})\\
&\{x\} \sqsubseteq \forall \mathsf{Node?} \circ (r_x^i)^- \circ \mathsf{IncrX}_i? \circ h \circ r_x^i \circ \mathsf{next}^-.\{x\})\\
&\{x\} \sqsubseteq \forall \mathsf{Node?} \circ (r_x^j)^- \circ \mathsf{IncrX}_i? \circ h \circ r_x^j.\{x\}\\
&\{x\} \sqsubseteq \forall \mathsf{Node?} \circ (r_y^i)^- \circ h \circ r_y^i.\{x\}
\end{aligned}$$

We only show how to handle the x-coordinate, the y-coordinate is treated analogously.

Finally, we add the axiom that triggers the merging of the nodes with same coordinates:

$$\{\mathsf{Last}\} \equiv \mathsf{GridNode} \sqcap \bigsqcap_{1\leq i\leq k} (\exists r_x^i.\{\mathsf{Last}\} \sqcap \exists r_y^i.\{\mathsf{Last}\})$$

5.4.2 Encoding the Runs

We next simulate the computation of M using the grid we just created. We assume we have the following concept names available: (i) $A_1, \bar{A}_1, A_2, \bar{A}_2, A_s, \bar{A}_s$, for all $A \in \Sigma \cap \mathrm{N}_C$ and all $r \in \Sigma \cap \mathrm{N}_R$, (ii) L_γ, for all symbols $\gamma \in \Gamma^{\Sigma'} \cup \{B\}$, (iii) S_q, for all $q \in Q$, and (iv) $H_<$ and $H_>$.

Copying $\mathcal{A}$ onto the input tape. The first row of the grid, referred to as the *input tape*, represents the initial configuration of M. Recall that we encode Σ-ABoxes over the signature as words of length n^2 where each position in the word represents a pair of individuals in $\mathcal{A}$ and each pair of individuals occurring in $\mathcal{A}$ is represented by one position in the word. We now add axioms that make sure that each one of the first n^2 cells on the input tape corresponds to a single pair of individuals occurring in the KB, while the remainder of the cells on the input tape are filled out with the blank symbol B. This is done by assuring that every input cell, i.e., a node in the first row, has two pointers to known individuals: hasFst and hasSnd. If for some input cell e there are two known individuals a, b s.t. (e, a) participates in hasFst and (e, b) participates in hasSnd, then e represents the pair (a, b). To ensure that all pairs are represented on the input tape, we follow the same approach as for the grid construction. Namely, the available linear order is lifted to pairs of known individuals, and we require that a horizontal successor of some input cell also represents the next pair w.r.t. to this linear order. Once all pairs are represented, the remaining input cells are set to blank, i.e., they participate in the concept L_B. The axioms that are used to achieve this are given as follows:

$$\begin{aligned}
\mathsf{InputCell} &\equiv \mathsf{GridNode} \sqcap \prod_{1 \le i \le k} \exists r^i_y.\{\mathsf{First}\} \\
L_B &\sqsubseteq \neg\exists\mathsf{hasFst} \sqcap \neg\exists\mathsf{hasSnd} \\
\mathsf{InputCell} &\sqsubseteq L_B \sqcup (\exists.\mathsf{hasFst}.\mathsf{Node} \sqcap \exists.\mathsf{hasSnd}.\mathsf{Node}) \\
\mathsf{InputCell} \sqcap \prod_{1 \le i \le k} \exists r^i_x.\{\mathsf{First}\} &\sqsubseteq \exists.\mathsf{hasFst}.\{\mathsf{First}\} \sqcap \exists.\mathsf{hasSnd}.\{\mathsf{First}\} \\
\neg L_B \sqcap \neg\exists\mathsf{hasFst}.\{\mathsf{Last}\} &\sqsubseteq \mathsf{IncrFst} \\
\exists\mathsf{hasFst}.\{\mathsf{Last}\} \sqcap \neg\exists\mathsf{hasSnd}.\{\mathsf{Last}\} &\sqsubseteq \mathsf{IncrSnd} \\
\exists\mathsf{hasFst}.\{\mathsf{Last}\} \sqcap \exists\mathsf{hasSnd}.\{\mathsf{Last}\} &\sqsubseteq \mathsf{SwitchToBlank} \\
\mathsf{SwitchToBlank} &\sqsubseteq \forall h.\mathsf{SwitchToBlank} \\
\{x\} &\sqsubseteq \forall \mathsf{hasFst}^- \circ \mathsf{IncrFst}? \circ h \circ \mathsf{hasFst} \circ \mathsf{next}^-.\{x\} \\
\{x\} &\sqsubseteq \forall \mathsf{hasSnd}^- \circ \mathsf{IncrFst}? \circ h \circ \mathsf{hasSnd}.\{x\} \\
\{x\} &\sqsubseteq \forall \mathsf{hasSnd}^- \circ \mathsf{IncrSnd}? \circ h \circ \mathsf{hasSnd} \circ \mathsf{next}^-.\{x\} \\
\mathsf{IncrSnd} &\sqsubseteq \forall h.\exists\mathsf{hasFst}.\{\mathsf{First}\} \\
\mathsf{SwitchToBlank} &\sqsubseteq \forall h.L_B
\end{aligned}$$

We next need put the correct symbols in each cell on the input tape. Recall that, if position i in the encoding of the ABox represents the pair (a, b), we have the following

symbol at position i: $\gamma = (\{A : A(a) \in \mathcal{A}\}, \{r : r(a,b) \in \mathcal{A}\}, \{A : A(b) \in \mathcal{A}\}) \in \Gamma^{\Sigma'}$. We next add axioms that ensure exactly that. Namely, if a cell on the input tape represents the pair (a,b), then it participates in the concept L_γ. We first copy the information about which concept and role names a and b participate in. To this end, for $A \in \Sigma \cap \mathrm{N}_C$ and every $r \in \Sigma \cap \mathrm{N}_R$ we add:

$$\begin{array}{lll} A_1 \sqcap \bar{A}_1 \sqsubseteq \bot & \exists \mathsf{hasFst}.A \sqsubseteq A_1 & \exists \neg \mathsf{hasFst}.A \sqsubseteq \bar{A}_1 \\ A_2 \sqcap \bar{A}_2 \sqsubseteq \bot & \exists \mathsf{hasSnd}.A \sqsubseteq A_2 & \exists \neg \mathsf{hasSnd}.A \sqsubseteq \bar{A}_2 \\ A_s \sqcap \bar{A}_s \sqsubseteq \bot & \exists \mathsf{hasFst}.(\{x\} \sqcap \exists s.\{y\}) \sqcap \exists \mathsf{hasSnd}.\{y\} \sqsubseteq A_s & \\ & \exists \mathsf{hasFst}.(\{x\} \sqcap \neg \exists s.\{y\}) \sqcap \exists \mathsf{hasSnd}.\{y\} \sqsubseteq \bar{A}_s & \end{array}$$

Finally, for each $\gamma = (T, R, T') \in \Gamma^\Sigma$, we add:

$$\bigsqcap_{\substack{A \in T, \\ B \in (\Sigma \cap \mathrm{N}_C) \setminus T}} (A_1 \sqcap \bar{B}_1) \sqcap \bigsqcap_{\substack{A \in T', \\ B \in (\Sigma \cap \mathrm{N}_C) \setminus T'}} (A_2 \sqcap \bar{B}_2) \sqcap \bigsqcap_{\substack{s \in R, \\ (r \in \Sigma \cap \mathrm{N}_R) \setminus R}} (A_s \sqcap \bar{A}_r) \sqsubseteq L_\gamma,$$

We now use the rest of the grid to simulate the computation of the TM M. Recall that a row in the grid stores a configuration that M is currently in, while v corresponds to time. To this end, we need to ensure that for each row ϱ in the grid satisfies two conditions. Firstly, there is exactly one element e in ϱ where S_q holds for some and at most one $q \in Q$. For other elements $e' \neq e$ in ϱ, S_q does not hold for any q. Secondly, for all elements of e in ϱ, L_γ holds for exactly one $\gamma \in \Gamma \cup \{B\}$. It is then clear that each row is indeed a valid encoding of some configuration of M. If S_q for a node e in ϱ, then M is in state q and the read-write head is in the position e.

We next add the axioms that ensure that at the beginning, M is in the state q_0 and the read-write head is above the first symbol:

$$\mathsf{InputCell} \sqcap \bigsqcap_{1 \leq i \leq k} r_x^i.\{\mathsf{First}\} \sqsubseteq S_{q_0} \qquad S_q \sqsubseteq \bigsqcap_{q' \in (Q \setminus \{q\})} S_{q'}, \text{ for all } q \in Q$$

Further, for all $(q, \gamma) \in Q \times \Gamma \cup \{B\}$, we add the following axiom that selects one configuration among possible next configurations, and overwrites the current symbol, changes the state and moves the read-write head accordingly:

$$S_q \sqcap L_\gamma \sqsubseteq \Big(\bigsqcup_{(q', \gamma', +1) \in \delta(q, \gamma)} \forall v.(L_{\gamma'} \sqcap \forall h.S_{q'}) \Big) \sqcup \Big(\bigsqcup_{(q', \gamma', -1) \in \delta(q, \gamma)} \forall v.(L_{\gamma'} \sqcap \forall h^-.S_{q'}) \Big)$$

For all states $q \in Q$, we mark the positions that are not under the read-write head:

$$S_q \sqsubseteq (\forall h.H_<) \sqcap (\forall h^-.H_>) \qquad H_< \sqsubseteq \bigsqcap_{q \in Q} S_{q'} \sqcap \forall h.H_< \qquad H_> \sqsubseteq \bigsqcap_{q \in Q} S_{q'} \sqcap \forall h^-.H_<$$

The intuition of the above is as follows. If in some position e we have S_q, then all the position to the right from e are marked with $H_<$. Intuitively, $H_<$ (resp. $H_>$) says that the read-write head is behind (resp. ahead) and thus these positions do not participate in S_q, for any q.

As one of the last steps, we need to add an axiom that copies the content of the tape that is not overwritten. For all $\gamma \in \Gamma \cup \{B\}$ we add: $L_\gamma \sqcap (H_> \sqcup H_<) \sqsubseteq \forall v.L_\gamma$

We are now almost done with our construction of $\mathcal{T}_M$: for any Σ-ABox $\mathcal{A}$, each model of $(\mathcal{T}, \Sigma, \mathcal{A})$, where $\mathcal{T}$ is the TBox we have constructed so far, corresponds to one possible computation of M on the encoding of $\mathcal{A}$. By assumption, M always terminates, which means that in each model of the theory we will either have some object for which q_{acc} holds or some object for which q_{rej} holds. Finally, to obtain $\mathcal{T}_M$, we add the axiom $q_{rej} \sqsubseteq \bot$ to $\mathcal{T}$. Now, every model of $(\mathcal{T}_M, \Sigma, \mathcal{A})$ corresponds to computation of M accepting the encoding of $\mathcal{A}$. Thus, for $Q_M = (\mathcal{T}_M, \Sigma)$, $Q_M(\mathcal{A}) = 1$ if and only if there are no accepting computations of M ran on enc^Σ, that is, $Q(A) = 1$.

5.5 Discussion

In this chapter, we have discussed some of the expressiveness limitation of very expressive OMQ languages, and then proposed an extension of $\mathcal{ALCHOIF}$ equipped with closed predicates as OMQ language that captures precisely the class of generic Boolean queries over ABoxes that are computable in coNP.

We saw that $\mathcal{ALCHOI}$ with closed predicates and IQs/inconsistency queries cannot express all coNP computable queries. However, the relationship between richer OMQ languages that are based on this logic, e.g., CQs mediated by $\mathcal{ALCHOI}$ ontologies with closed predicates and coNP remains unclear. We note that there is no known upper bound on the data complexity of this language, so investigating this problem is an important step towards characterization of its expressive power. Another question that remains open is whether inconsistency queries mediated by plain $\mathcal{ALCHOIF}$ (i.e., without nominal schemata) ontologies with closed predicates can express all coNP computable GBQs. At this point the argument of Theorem 5.2.3 cannot be adapted. To see this, observe that the key observation leading to the result of Theorem 5.2.3 is that there is an algorithm that checks whether a given $\mathcal{ALCHOI}$ KB $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$ is satisfiable in $|\mathcal{A}|^k \times l + v$, where k, l, v are constants and k *does not depend on* $\mathcal{T}$ *or* Σ. However, in our satisfiability procedure for $\mathcal{ALCHOIF}$ with closed predicates, this constant depends on the TBox.

We conclude this chapter by observing that the arguments presented in the chapter can also be applied to standard Horn-DLs (with no closed predicates). For instance, the OMQ language that couples inconsistency and instance queries with $\mathcal{ELHIF}$ ontologies is PTime-hard, but it cannot express all queries computable in PTime. We believe that extending $\mathcal{ELHIF}$ with a built-in linear order is not sufficient to capture PTime, but

that the further addition of the features described in Section 5.2.1 leads to a DL that precisely captures PTime. This remains to be investigated in our future work.

CHAPTER 6

Reasoning about Predicate Boundedness

A distinguishing feature of DLs, aimed at dealing with information incompleteness, is the ability to describe and reason about *anonymous objects*, that is, elements in the domain of interest that are not represented by known individuals but whose existence is logically implied by the background knowledge.

Consider a simple TBox with the following two axioms:

$$\mathsf{Employee} \equiv \{\mathsf{Robin}\} \sqcup \{\mathsf{Skyler}\}$$
$$\mathsf{Employee} \sqsubseteq \neg\mathsf{Task} \sqcap \geq 2\mathsf{hasTask}.\mathsf{Task} \sqcap \leq 5\mathsf{hasTask}.\mathsf{Task}$$

which state that Robin and Skyler are precisely the employees of some company, that the sets of employees and tasks are disjoint, and that each employee must be assigned between 2 and 5 tasks to work on. If we inspect the models of this TBox, then the *named* objects corresponding to Robin and Skyler will be associated with 2 to 5 *anonymous* objects corresponding to tasks.

When the number of anonymous objects cannot be bounded (e.g., when it is forced to be infinite by a recursive TBox), one often faces high computational complexity of standard reasoning tasks, and undecidability of more sophisticated problems, like answering Datalog-based recursive queries (see, e.g., [LR98, Ros07a]) or reasoning about data-manipulating actions (see, e.g., [BHCDG$^+$13]). However, in many scenarios, the number of anonymous objects in relevant predicates can be inferred to be bounded due to numeric constraints present in the TBox. For example, if we add to the above TBox the statement that every task must be associated to an employee ($\mathsf{Task} \sqsubseteq \exists \mathsf{hasTask}^-.\mathsf{Employee}$), then the extension of Task in any model is bounded by 10. We suggest [BBR20] for a longer discussion of numeric constraints in DLs.

The question we investigate in this chapter is the existence of upper bounds on predicate sizes for ontologies written in $\mathcal{ALCHOIQ}$ with *closed predicates*. Our goal is to understand when and how we can infer bounds on the sizes of open predicates from the extensions of closed predicates, by taking into account the (numeric and other) constraints specified in a TBox.

Contributions and Relevant Publications. Our contributions can be summarized as follows:

- We introduce the notion of *bounded predicates* for DLs with closed predicates. Intuitively, a predicate is *bounded* w.r.t a concrete knowledge base, if there exists an integer constant that provides an upper bound on the size of the predicate's extension in all models of the knowledge base. Since we are interested in aiding the design of ontologies and ontology-mediated queries, we also consider a stronger variant of predicate boundedness that requires the predicate to be bounded independently of the actual extensional data stored in a knowledge base.

- We formalize two decision problems – checking predicate boundedness in the weak and in the strong sense, and we characterize their computational complexity. For $\mathcal{ALCHOIQ}$, our method is based on integer programming and it involves an intermediate step that reformulates the problem in terms of another problem called *finite-infinite satisfiability*. The task in this problem is to check whether a TBox has a model in which some specific predicates have a *finite extension*, while some other given predicates have an *infinite extension*. This is closely related to *mixed satisfiability* studied in [GGI+20] and is interesting in its own right.

- We show that our results yield worst-case optimal complexity bounds in all cases. Specifically, checking boundedness in the weak and the strong sense is co-NExpTime-complete for $\mathcal{ALCHOIQ}$ TBoxes. Moreover, in case predicate boundedness is inferred, the concrete bound can be readily computed; it is double exponential in the size of the input. This is worst-case optimal: one can craft a TBox that forces a double exponential number of elements in a bounded predicate. For the case of strong boundedness, we show that there exists a function $f_{\mathcal{T},\Sigma}$ depending on $\mathcal{T}$ and Σ, such that $f_{\mathcal{T},\Sigma}$ computes a finite upper bound on the number of different domain elements that can occur in extensions of predicates that are bounded w.r.t $\mathcal{T}$ and Σ, for all models of $(\mathcal{T}, \Sigma, \mathcal{A})$, where $\mathcal{A}$ is an arbitrary ABox $\mathcal{A}$ over the signature of $\mathcal{T}$ with at most n distinct individuals. This function is generally doubly-exponential in the size of $(\mathcal{T}, \Sigma, \mathcal{A})$, but it is polynomial function if we fix the TBox, i.e., the number of elements in bounded predicates will grow polynomially in the size of the input ABox.

- Finally, we show how our results can be used to extend safe-range queries introduced in Chapter 4 as well as to define a new decidability-ensuring safety condition for OMQs based on $\mathcal{ALCHOIQ}$ and Datalog, providing worst-case optimal results

on the combined and data complexity. This is interesting because there are very few positive results on query answering in this expressive DL. We also infer that the data complexity of satisfiability of $\mathcal{ALCHOIQ}$ KBs with closed predicate is coNP-complete.

The results from this chapter have been published in:

[LŠ21] **Sanja Lukumbuzya**, and Mantas Šimkus. "Bounded Predicates in Description Logics with Counting". In Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, pp. 1966-1972. 2021.

Organization. The rest of this chapter is organized as follows. In Section 6.1, we introduce two notions of predicate boundedness with respect to first-order logic theories and a set of distinguished predicates: (i) weak boundedness, i.e., predicates whose extension is bounded in size in all models of a given KB and (ii) strong boundedness, which is the data independent version of weak boundedness, i.e., predicates that are weakly bounded in all KBs $(\mathcal{T}, \Sigma, \mathcal{A})$, for a given FO-theory $\mathcal{T}$ and Σ. We next define the problem of FI-satisfiability of FO theories which asks whether some theory $\mathcal{T}$ has a model in which some predicates must have finite extensions while some other predicates must have infinite extensions. Finally, we show that we can reduce the problem of deciding whether all predicates in some set are strongly bounded w.r.t. some theory $\mathcal{T}$ and set of predicates Σ by reduction to FI-unsatisfiability. In Section 6.3 we show how we can use integer programming to decide boundedness when FO theories are written in $\mathcal{ALCHOIQ}$ and we provide tight complexity results as well as a general bound on the size of extensions of bounded predicates. Finally, in Section 6.4, we use bounded predicates to show decidability and complexity results for answering Datalog and safe-range queries with a relaxed safety condition, followed by a discussion of our results in Section 6.5.

6.1 Bounded Predicates

Let us begin with an example that illustrates boundedness of predicates.

Example 6.1.1. *Consider an $\mathcal{ALCHOIQ}$ TBox $\mathcal{T}$ consisting of the following axioms:*

$$\begin{aligned} \mathsf{Empl} &\sqsubseteq \leq 5\ \mathsf{assgndTo.Proj}, \\ \mathsf{Proj} &\sqsubseteq \geq 1 \mathsf{assgndTo}^{-}.\mathsf{Empl}, \\ \mathsf{Empl} \sqcap \mathsf{Proj} &\sqsubseteq \bot, \end{aligned}$$

stating that each employee in some company can be assigned to at most five projects, that all projects must be assigned to least one employee, and that the sets of projects and employees are disjoint. Let $\mathcal{A}$ be an arbitrary ABox over Empl *that lists all n employees of the company, i.e., $\mathcal{A} = \{\mathsf{Empl}(a_1), \ldots, \mathsf{Empl}(a_n)\}$. Further, let $\mathcal{I}$ be an arbitrary model*

of $(\mathcal{T}, \{\mathsf{Empl}\}, \mathcal{A})$. *As* Empl *is viewed as a closed predicate,* $|\mathsf{Empl}^{\mathcal{I}}| = |\mathcal{A}| = n$. *Taking into account the axioms from* $\mathcal{T}$, *it is easy to see that the number of projects in* $\mathcal{I}$ *can be at most* $5n$, *i.e.,* $|\mathsf{Proj}^{\mathcal{I}}| \leq 5n$. *Therefore, the size of the extensions of* Proj *is in a way bounded w.r.t.* $\mathcal{T}$ *and* Empl.

Consider a TBox $\mathcal{T}$ and a set of predicates Σ occurring in $\mathcal{T}$. Intuitively, the predicates in Σ are either considered closed or their extensions are known to be of bounded size. We next formally define what it means for some predicate to be bounded with respect to a specific knowledge base $(\mathcal{T}, \Sigma, \mathcal{A})$, in case we are given an ABox $\mathcal{A}$, as well as with respect to $\mathcal{T}$ and Σ, if the ABox is not given.

Definition 6.1.2. *Let* $\mathcal{T}$ *be a DL TBox,* Σ *be a set of predicates occurring in* $\mathcal{T}$, *and* $\mathcal{A}$ *be an ABox over the signature of* $\mathcal{T}$. *A predicate* p *is* bounded w.r.t the knowledge base $(\mathcal{T}, \Sigma, \mathcal{A})$ *if there exists an integer bound* $b \in \mathbb{N}$ *s.t. in every model* $\mathcal{I}$ *of* $(\mathcal{T}, \Sigma, \mathcal{A})$, $|p^{\mathcal{I}}| \leq b$ *holds. We say that* p *is* bounded w.r.t $\mathcal{T}$ and Σ *if* p *is bounded w.r.t.* $(\mathcal{T}, \Sigma, \mathcal{A})$, *for every ABox* $\mathcal{A}$ *over the signature of* $\mathcal{T}$.

Based on the definition above, we can define two decision problems. The first problem is the problem of deciding whether all predicates in a given set are bounded w.r.t. a given TBox formulated in the DL $\mathcal{L}$ and a set of closed predicates is called *strong boundedness*, or Boundedness($\mathcal{L}$), and it is defined as follows:

Boundedness($\mathcal{L}$)	
Input:	A triple $(\mathcal{T}, \Sigma, \Sigma_B)$, where $\mathcal{T}$ is a TBox in DL $\mathcal{L}$ and Σ, Σ_B are sets of predicates occurring in $\mathcal{T}$.
Question:	Is each $p \in \Sigma_B$ bounded w.r.t. $\mathcal{T}$ and Σ?

Notation: For a given TBox $\mathcal{T}$ and a set of predicates $\Sigma \subseteq \mathsf{N_C} \cup \mathsf{N_R}$, let $\mathsf{B_P}(\mathcal{T}, \Sigma)$ be the set of all predicates occurring in $\mathcal{T}$ and Σ that are bounded w.r.t. $\mathcal{T}$ and Σ:

$$\mathsf{B_P}(\mathcal{T}, \Sigma) = \{p \in \mathsf{N_C}(\mathcal{T}) \cup \mathsf{N_R}(\mathcal{T}) \cup \Sigma : p \text{ is bounded w.r.t. } \mathcal{T} \text{ and } \Sigma\}.$$

Moreover, let $\mathsf{B_C}(\mathcal{T}, \Sigma) = \mathsf{B_P}(\mathcal{T}, \Sigma) \cap \mathsf{N_C}$ and $\mathsf{B_R}(\mathcal{T}, \Sigma) = \mathsf{B_P}(\mathcal{T}, \Sigma) \cap \mathsf{N_R}$.

The second decision problem, called *weak boundedness*, or W-Boundedness($\mathcal{L}$), refers to the problem of deciding, given an $\mathcal{L}$-KB $\mathcal{K}$ and a set Σ_B of predicates over the signature of $\mathcal{T}$, whether all predicates in Σ_B are bounded w.r.t. $\mathcal{K}$.

W-Boundedness($\mathcal{L}$)	
Input:	A pair $(\mathcal{K}, \Sigma_B)$, where $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$, $\mathcal{T}$ is a TBox in DL $\mathcal{L}$, Σ, Σ_B are sets of predicates occurring in $\mathcal{T}$, and $\mathcal{A}$ is an ABox over the signature of $\mathcal{T}$.
Question:	Is each $C \in \Sigma_B$ bounded w.r.t. $\mathcal{K}$?

Boundedness via FI-Satisfiability. We now focus specifically on strong boundedness for description logics that are fragments of the first-order logic. Note that, if not specified otherwise, under *boundedness* we understand strong boundedness. As a technical tool we introduce another decision problem called *FI-satisfiability*, or FI-SAT($\mathcal{L}$), which is the problem of deciding for a given TBox $\mathcal{T}$ expressed in the DL $\mathcal{L}$, and two sets of predicates Σ_F and Σ_I, whether $\mathcal{T}$ has a model in which all the predicates in Σ_F have finite extensions and all the predicates in Σ_I have infinite extensions.

FI-SAT($\mathcal{L}$)	
Input:	A triple $(\mathcal{T}, \Sigma_F, \Sigma_I)$, where $\mathcal{T}$ is a TBox in DL $\mathcal{L}$, $\Sigma_F \cup \Sigma_I$ is a set of predicates occurring in $\mathcal{T}$.
Question:	Is there an *FI-model* of $(\mathcal{T}, \Sigma_F, \Sigma_B)$, i.e., a model $\mathcal{I}$ of $\mathcal{T}$ s.t. $p^{\mathcal{I}}$ is finite for all $p \in \Sigma_F$ and is infinite for all $p \in \Sigma_I$?

If an instance $(\mathcal{T}, \Sigma_F, \Sigma_I)$ of FI-SAT($\mathcal{L}$) has an FI-model, we say that this instance is *FI-satisfiable*, otherwise it is *FI-unsatisfiable*.

Example 6.1.3. *Recall the TBox $\mathcal{T}$ from Example 6.1.1 and let $\Sigma_F = \{\mathsf{Empl}\}$, $\Sigma_I = \{Proj\}$. Let $\mathcal{I}$ be an arbitrary model of $\mathcal{I}$ such that $\mathsf{Empl}^{\mathcal{I}}$ is finite. We have already argued that the axioms of $\mathcal{T}$ force us to have $|\mathsf{Proj}^{\mathcal{I}}| \leq 5 \cdot |\mathsf{Empl}^{\mathcal{I}}|$. Thus, whenever $\mathsf{Empl}^{\mathcal{I}}$ is finite, so is $\mathsf{Proj}^{\mathcal{I}}$ and the triple $(\mathcal{T}, \Sigma_F, \Sigma_I)$ is not FI-satisfiable.*

By contrast, the triple $(\mathcal{T}, \Sigma'_F, \Sigma'_I)$ where $\Sigma'_F = \{\mathsf{Proj}\}$ and $\Sigma'_I = \{\mathsf{Empl}\}$, is FI-satisfiable. Indeed, it is easy to see that $\mathcal{T}$ has a model in which there are infinitely many employees but only finitely many projects.

We next show that we can reduce the boundedness problem to deciding FI-satisfiability. From the Definition 6.1.2, it is easy to observe that a concept or a role name p is not bounded w.r.t. $\mathcal{T}$ and Σ if and only if there exists an ABox $\mathcal{A}$ over the signature of $\mathcal{T}$ such that for every natural number n there exists a model $\mathcal{I}$ of $(\mathcal{T}, \Sigma, \mathcal{A})$ with $|p^{\mathcal{I}}| > n$. The following proposition, formulated for FO, shows that in this case $(\mathcal{T}, \Sigma, \mathcal{A})$ also has a model in which the extension of p is infinite.

Proposition 6.1.4. *Let $\mathcal{T}$ be an arbitrary FO theory and let p be a unary or a binary predicate. If for every natural number n there exists a model $\mathcal{I}$ of $\mathcal{T}$ in which $|p^{\mathcal{I}}| > n$, then there also exists a model $\mathcal{J}$ of $\mathcal{T}$ in which $p^{\mathcal{J}}$ is infinite.*

Proof. Let p be a binary predicate. For every $k \geq 1$, let φ_k be a formula which is satisfied by interpretations in which there are k or more tuples in the extension of p:

$$\varphi_k := \exists x_1, x'_1, \ldots, x_k, x'_k \bigwedge_{1 \leq i < j \leq k} (x_i \neq x_j \vee x'_i \neq x'_j) \wedge p(x_i, x'_i).$$

Consider the FO theory $\mathcal{T}' = \mathcal{T} \cup \{\varphi_n : n \geq 1\}$. As for every n, there is a model $\mathcal{I}$ of $\mathcal{T}$ in which $|p^{\mathcal{I}}| > n$, every finite subset of $\mathcal{T}'$ is satisfiable. By compactness of FO, $\mathcal{T}'$ is satisfiable as well. However, by construction, $\mathcal{T}'$ only admits models in which the extension of p is infinite. Hence, there exists a model $\mathcal{J}$ of $\mathcal{T}'$ such that $p^{\mathcal{J}}$ is infinite. As $\mathcal{T} \subseteq \mathcal{T}'$, $\mathcal{J}$ is also a model of $\mathcal{T}$ and so there is a model of $\mathcal{T}$ in which the extension of p is infinite. The case where p is a unary predicate is shown analogously. □

In view of the previous proposition, it is easy to show that the problem of boundedness reduces to deciding FI-unsatisfiability.

Proposition 6.1.5. *Let $\mathcal{T}$ be a TBox in DL $\mathcal{L}$, and Σ, Σ_B be sets of predicates occurring in $\mathcal{T}$. $(\mathcal{T}, \Sigma, \Sigma_B)$ is a yes-instance of Boundedness($\mathcal{L}$) if and only if $(\mathcal{T}, \Sigma, \{p\})$ is a no-instance of FI-SAT($\mathcal{L}$), for all $p \in \Sigma_B$.*

Proof. We first show that a predicate $p \in \Sigma_B$ is not bounded w.r.t. $\mathcal{T}$ and Σ if and only if there exists a model $\mathcal{I}$ of $\mathcal{T}$ in which all predicates from Σ have finite extensions and $p^{\mathcal{I}}$ is infinite.

($\Rightarrow$:) Assume that p is not bounded w.r.t. $\mathcal{T}$ and Σ. By definition, this means that there is some ABox $\mathcal{A}$ over the signature of $\mathcal{T}$ s.t. for every $b \in \mathbb{N}$, there exists a model $\mathcal{J}$ of $(\mathcal{T}, \Sigma, \mathcal{A})$ in which $|p^{\mathcal{J}}| > b$. Note that, since we are considering DLs that are fragments of FO, we can obtain from $(\mathcal{T}, \Sigma, \mathcal{A})$ an FO theory $\mathcal{T}'$ whose models coincide with those of $(\mathcal{T}, \Sigma, \mathcal{A})$ as follows:

$$\mathcal{T}' = \mathrm{fo}(\mathcal{T}) \cup \mathcal{A} \cup \{\varphi_p : p \in \Sigma\},$$

where $\mathrm{fo}(\mathcal{T})$ is a theory obtained by translating $\mathcal{T}$ into FO, and for every $p \in \Sigma$, φ_p encodes the semantics of the closed predicates. Namely, for each concept name $A \in \Sigma$, we let $\Delta_A = \{c : A(c) \in \mathcal{A}\}$ and $\varphi_A := \forall x A(x) \rightarrow \bigvee_{c \in \Delta_A} x = c$. For a role name $r \in \Sigma$, φ_r is defined in a similar manner. It is easy to see that $\mathcal{T}'$ has exactly the same models as $(\mathcal{T}, \Sigma, \mathcal{A})$. Due to Proposition 6.1.4, there exists a model $\mathcal{I}$ of $\mathcal{T}'$ (and thus also of $(\mathcal{T}, \Sigma, \mathcal{A})$) in which $p^{\mathcal{I}}$ is infinite. As ABoxes are finite by definition and the extensions of the predicates in Σ are fully specified by $\mathcal{A}$, we have that the predicates in Σ have finite extensions in $\mathcal{I}$.

($\Leftarrow$:) Let $\mathcal{J}$ be a model of $\mathcal{T}$ s.t. the predicates in Σ have finite extensions and p has an infinite extension in $\mathcal{J}$, and let $\mathcal{A} = \{q(\vec{c}) : \vec{c} \in q^{\mathcal{I}}, q \in \Sigma\}$. As $\mathcal{J}$ is a model of $(\mathcal{T}, \Sigma, \mathcal{A})$ and $|p^{\mathcal{J}}| > k$, for all $k \in \mathbb{N}$, p is not bounded w.r.t. $\mathcal{T}$ and Σ.

We have therefore shown that a predicate $p \in \Sigma_B$ is bounded w.r.t. $\mathcal{T}$ and Σ if and only if $(\mathcal{T}, \Sigma, \{p\})$ is a no-instance of FI-SAT($\mathcal{L}$). The result of the proposition is now immediate. □

Corollary 6.1.6. *Let $\mathcal{T}$ be a TBox in DL $\mathcal{L}$, and Σ, Σ_B be sets of predicates occurring in $\mathcal{T}$. $(\mathcal{T}, \Sigma, \Sigma_B)$ is a yes-instance of* Boundedness($\mathcal{L}$) *if and only if $(\mathcal{T}, \Sigma, \{p\})$ is a no-instance of FI-SAT($\mathcal{L}$), for all $p \in \Sigma_B$.*

6.2 FI-enriched Systems and Programs

In this Section, we introduce *FI-enriched systems* as an auxiliary tool that will be useful in for analyzing predicate boundedness when the theory is written in the DL $\mathcal{ALCHOIQ}$. Intuitively, FI-enriched systems further extend the notion of enriched systems introduced in Chapter 3 (see Definition 3.2.18) in a way that allows us to specify that certain variables must take on finite values, as well as that in a certain set of variables, at least one variable needs to take on the infinite value $\aleph_0$. We next formally define such systems.

Definition 6.2.1. *An* FI-enriched system (of integer linear inequalities) *is a tuple $(V, \mathcal{E}, V_F, V_I, I)$, where:*

- *V is a set of variables,*
- *$\mathcal{E}$ is a set of inequalities of the form*

$$a_1 \cdot x_1 + \cdots + a_n \cdot x_n + c \leq b_1 \cdot y_1 + \cdots + b_m \cdot y_m + d,$$

  where $a_1, \ldots, a_n, b_1, \ldots, b_m \geq 0$ are variable coefficients*, $c, d \geq 0$ are* constant coefficients*, and $x_1, \ldots, x_n, y_1, \ldots, y_m \in V$,*
- *$V_F \subseteq V$,*
- *$V_I \subseteq 2^V$, and*
- *I is the set of implications of the form $\alpha \Rightarrow \beta$, where α and β are inequalities.*

We also extend the definition of *solutions* as follows. We say that a function $S : V \to \mathbb{N}^*$ is a solution of $(V, \mathcal{E}, V_F, V_I, I)$ over $\mathbb{N}^*$ if all inequalities and implications are satisfied and additionally:

- $S(x) \neq \aleph_0$, for every $x \in V_F$,
- for every set $X \in V_I$, there exists some variable $x \in X$ s.t. $S(x) = \aleph_0$.

Another notion that will be useful throughout this chapter is the notion of *FI-enriched integer linear programs* that couple FI-enriched systems with an objective function that admissible solutions should maximize.

Definition 6.2.2. *An* FI-enriched integer linear program (FI-enriched ILP)*, is a pair* $\mathfrak{I} = (a_1 \cdot x_1 + \cdots + a_n \cdot x_n, \mathcal{F})$*, where* $\mathcal{F} = (V, \mathcal{E}, V_F, V_I, I)$ *is an FI-enriched system of integer linear inequalities,* $x_1, \ldots, x_n \in V$ *and* $a_1, \ldots, a_n$ *are integers. A solution* S *to* $\mathcal{F}$ *over* $\mathbb{N}^*$ *is an* optimal solution *to* $\mathfrak{I}$ *if for all solutions* S' *of* $\mathcal{F}$ *over* $\mathbb{N}^*$ *we have that* $a_1 \cdot S(x_1) + \cdots + a_n \cdot S(x_n) \geq a_1 \cdot S'(x_1) + \cdots + a_n \cdot S'(x_n)$*. The value* $a_1 \cdot S(x_1) + \cdots + a_n \cdot S(x_n)$ *is the* optimal value *of* $\mathfrak{I}$*.*

In the rest of this section, we show that we can decide existence of solutions to FI-enriched systems as well as compute optimal solutions to FI-enriched ILPs using ordinary integer programming techniques. In Chapter 3 we defined ordinary systems of integer linear inequalities. *Ordinary integer linear programs* can be defined analogously to FI-enriched ILPs.

Definition 6.2.3. *We call a pair* $(a_1 \cdot x_1 + \cdots + a_n \cdot x_n, \mathcal{S})$*, where* $a_1, \ldots, a_n$ *are integers and* $x_1, \ldots, x_n \in V$*, an* enriched integer linear program (enriched ILP) *if* $\mathcal{S} = (V, \mathcal{E}, I)$ *is an enriched system and* ordinary integer linear program (ordinary ILP) *if* $I = \emptyset$*. Optimal solutions and optimal values of enriched and ordinary ILPs are defined analogously to those of FI-enriched ILPs, but only solutions to* $\mathcal{S}$ *over* $\mathbb{N}$ *are considered.*

An ordinary ILP may have none, one or multiple optimal solutions. However, it follows from well-known integer programming results (cf. Theorem 10.3 in [Sch99]) that if an ordinary ILP has an optimal solution, then it has one in which all values are bounded by a certain function. We recall this result below.

Proposition 6.2.4. *Let* $\mathfrak{I} = (a_1 \cdot x_1 + \cdots + a_n \cdot x_n, \mathcal{S})$ *be an ordinary ILP, where* $\mathcal{S} = (V, \mathcal{E})$*. Assume that* $\mathfrak{I}$ *has a finite optimum value* b*, let* c *be a coefficient in* $\mathcal{E}$ *s.t. for all other coefficients* c' *occurring in* $\mathcal{E}$*,* $|c| \geq |c'|$*, and let* $a = \max\{|c|, |a_1|, \ldots, |a_n|, 1\}$*. Then*

$$b \leq (a+1)^{16(|V|+|\mathcal{E}|+|V|\cdot|\mathcal{E}|)}.$$

Proof. From Theorem 10.3 in [Sch99], we have that the following holds:

$$\begin{aligned} 2 + \log_2(b+1) &\leq 4(|V| \cdot |\mathcal{E}| + |V| \cdot |\mathcal{E}| \cdot (2 + \log_2(a+1)) + |\mathcal{E}| + |\mathcal{E}| \cdot (2 + \log_2(a+1))) \\ &\quad + |V| + |V| \cdot (2 + \log_2(a+1)) \\ &\leq 4(|V| \cdot |\mathcal{E}| + |V| + |\mathcal{E}|) \cdot (3 + \log_2(a+1)) \\ &\leq 4(|V| \cdot |\mathcal{E}| + |V| + |\mathcal{E}|) \cdot (4 \log_2(a+1)) \\ &\leq 16(|V| \cdot |\mathcal{E}| + |V| + |\mathcal{E}|) \cdot \log_2(a+1). \end{aligned}$$

From this, it is easy to see that $b \leq (a+1)^{16(|V|\cdot|\mathcal{E}|+|V|+|\mathcal{E}|)}$.

□

We next show that we can transfer the above-presented complexity results as well as properties of ordinary systems and ILPs to the FI-enriched setting. Relying on the result from [GGBIG+19] (see proof of Theorem 13 in [GGBIG+19]), given an FI-enriched system $\mathcal{F}$, we can reduce $\mathcal{F}$ to an enriched system $\mathcal{S} = (V', \mathcal{E}', I')$ whose solutions over $\mathbb{N}$ correspond to the solutions of $\mathcal{F}$ over $\mathbb{N}^*$.

Proposition 6.2.5. *Let $\mathcal{F} = (V, \mathcal{E}, V_F, V_I, I)$ be an FI-enriched system such that all the coefficients of $\mathcal{F}$ are in $\{0, \pm 1, \ldots, \pm a\}$. We can obtain in polynomial time an enriched system $\mathcal{S} = (V', \mathcal{E}', I')$ from $\mathcal{F}$ with the following properties:*

- *For every solution S of $\mathcal{F}$ over $\mathbb{N}^*$, there is a solution S' of $\mathcal{S}$ over $\mathbb{N}$ such that $S(x) = S'(x)$, for every $x \in V$ with $S(x) \neq \aleph_0$,*
- *For every solution S' of $\mathcal{S}$ over $\mathbb{N}^*$, there is a solution S of $\mathcal{F}$ over $\mathbb{N}^*$ such that either $S(x) = S'(x)$ or $S(x) = \aleph_0$, for every $x \in V$,*
- $|V'| = 2|V|$,
- $|\mathcal{E}'| \leq |\mathcal{E}| + |V_F| + |V_I|$,
- $|I'| \leq |I| + |\mathcal{E}|$, *and*
- *all coefficients in $\mathcal{S}$ are in $\{0, \pm 1, \ldots, \pm a\}$.*

Proof. (Proof adapted from [GGI+20]) The enriched system $\mathcal{S}$ is obtained as follows. For every variable $x \in V$ we introduce a fresh variable x^∞ and set $V' = V \cup \{x^\infty : x \in V\}$.

The set $\mathcal{E}'$ of inequalities is obtained from $\mathcal{E}$ as the smallest set that contains:

- $x^\infty = 0$, for every $x \in V_F$,
- $x_1^\infty + \cdots + x_n^\infty > 0$, for every set $\{x_1, \ldots, x_n\} \in V_I$, and
- $a_1 \cdot x_1 + \cdots + a_n \cdot x_n + c \leq b_1 \cdot y_1 + y_1^\infty + \cdots + b_m \cdot y_m + y_m^\infty + d$, for each inequality $a_1 \cdot x_1 + \cdots + a_n \cdot x_n + c \leq b_1 \cdot y_1 + \cdots + b_m \cdot y_m + d \in \mathcal{E}$.

Furthermore, the set I' of implications is obtained from I as the smallest set for which the following holds:

- For every implication $\alpha \implies \beta \in I$, where

$$\alpha = a_1 \cdot x_1 + \cdots + a_n \cdot x_n + c \leq b_1 \cdot y_1 + \cdots + b_m \cdot y_m + d,$$
$$\beta = a'_1 \cdot x'_1 + \cdots + a'_l \cdot x'_l + c' \leq b'_1 \cdot y'_1 + \cdots + b'_k \cdot y'_k + d',$$

  $\alpha' \implies \beta' \in I'$, where α' and β' are defined as follows:

$$\alpha' = a_1 \cdot x_1 + x_1^\infty + \cdots + a_n \cdot x_n + x_n^\infty + c \leq b_1 \cdot y_1 + \cdots + b_m \cdot y_m + d,$$
$$\beta' = a'_1 \cdot x'_1 + \cdots + a'_l \cdot x'_l + c' \leq b'_1 \cdot y'_1 + y'^\infty_1 + \cdots + b'_1 \cdot y'_1 + y'^\infty_1 + d'.$$

- For every inequality $a_1 \cdot x_1 + \cdots + a_n \cdot x_n + c \leq b_1 \cdot y_1 + \cdots + b_m \cdot y_m + d \in \mathcal{E}$, we have that $1 \leq x_1^\infty + \cdots + x_n^\infty \implies 1 \leq y_1^\infty + \cdots + y_m^\infty$ is in I'.

Let $\mathcal{S} = (V', \mathcal{E}', I')$. We next show that $\mathcal{S}$ has a solution over $\mathbb{N}$ if and only if $\mathcal{F}$ has a solution over $\mathbb{N}^*$. Indeed, let S be a solution of $\mathcal{F}$ over $\mathbb{N}^*$ and let $B = 1 + c + \sum_{x \in V, S(x) \neq \aleph_0} c \cdot S(x)$, where c be the maximal coefficient in $\mathcal{F}$. Let $S' : V' \to \mathbb{N}$ be a function defined as follows:

$$S'(x) = \begin{cases} S(x), & \text{if } x \in V \text{ and } S(x) \neq \aleph_0, \\ 0, & \text{if } x \in V \text{ and } S(x) = \aleph_0, \\ 0, & \text{if } x = y^\infty, y \in V, \text{ and } S(y) \neq \aleph_0, \\ B, & \text{if } x = y^\infty, y \in V, \text{ and } S(y) = \aleph_0. \end{cases}$$

It is easy to verify that S' is indeed a solution to $\mathcal{S}$. Conversely, let S' be a solution to $\mathcal{S}$ and let $S : V \to \mathbb{N}^*$ be a function defined as

$$S(x) = \begin{cases} S'(x), & \text{if } x^\infty \in V' \text{ and } S'(x^\infty) = 0, \\ \aleph_0, & \text{if } x^\infty \in V' \text{ and } S'(x^\infty) > 0. \end{cases}$$

Once again, we can readily verify that S is a solution to $\mathcal{F}$.

By the construction of $\mathcal{S}$, we can easily see that:

- $|V'| = 2|V|$,
- $|\mathcal{E}'| \leq |\mathcal{E}| + |V_F| + |V_I|$,
- $|I'| \leq |I| + |\mathcal{E}|$, and
- all coefficients in $\mathcal{S}$ are in $\{0, \pm 1, \ldots, \pm a\}$.

□

The proposition above allows us to transfer the following results from enriched systems to FI-enriched systems.

Proposition 6.2.6. *An FI-enriched system $\mathcal{F} = (V, \mathcal{E}, V_F, V_I, I)$ with the coefficients from $\{0, \pm 1, \ldots, \pm a\}$ has a solution over $\mathbb{N}^*$ if it has a solution over $\mathbb{N}^*$ where all finite values are bounded by:*

$$(2|V| + |I| + 2|\mathcal{E}| + |V_F| + |V_I|) \cdot ((2|\mathcal{E}| + |V_F| + |V_I| + |I|) \cdot a)^{2(2|\mathcal{E}| + |V_F| + |V_I| + |I|) + 1}.$$

Moreover, deciding whether $\mathcal{F}$ has a solution over $\mathbb{N}^*$ is in NP.

Proof. Let $\mathcal{S} = (V', \mathcal{E}', I')$ be an enriched system obtained from $\mathcal{F}$ with the properties from Proposition 6.2.5. Due to Proposition 3.2.22, if $\mathcal{S}$ has a solution over $\mathbb{N}$, then it has one in which all values are bounded by $(|V'| + |I'| + |\mathcal{E}'|) \cdot ((|\mathcal{E}'| + |I'|) \cdot a)^{2(|\mathcal{E}'|+|I'|)+1}$, where a is the maximal coefficient in $\mathcal{S}$. Thus, in view of Proposition 6.2.5, we can conclude that if $\mathcal{F}$ has a solution over $\mathbb{N}^*$, then it has one in which all finite values are bounded by

$$(2|V| + |I| + 2|\mathcal{E}| + |V_F| + |V_I|) \cdot ((2|\mathcal{E}| + |V_F| + |V_I| + |I|) \cdot a)^{2(2|\mathcal{E}|+|V_F|+|V_I|+|I|)+1}.$$

Furthermore, as we can decide in NP whether $\mathcal{S}$ has a solution over $\mathbb{N}$ and $\mathcal{S}$ was obtained in polynomial time from $\mathcal{F}$, the NP upper bound for FI-enriched systems follows. □

We have already seen that we can decide if an enriched system $\mathcal{S}$ has a solution by non-deterministically constructing an ordinary system that completes the set of inequalities of $\mathcal{S}$ with the inequalities occurring in the implications of $\mathcal{S}$ s.t. all solutions to this ordinary system are also solutions to $\mathcal{S}$. Together with the previous two propositions, we can now show that there is an upper bound on the optimal value of FI-enriched systems.

Proposition 6.2.7. *Let $\mathfrak{I} = (a_1 \cdot x_1 + \cdots + a_n \cdot x_n, \mathcal{F})$ be an enriched ILP, where $\mathcal{F} = (V, \mathcal{E}, V_F, V_I, I)$, and assume that $\mathfrak{I}$ has a finite optimal value b. Then $b \leq (a + 1)^{16(2|\mathcal{E}|+|I|+|V_F|+|V_I|+1)\cdot(2|V|+1)}$, where c is a coefficient in $\mathcal{F}$ with the maxium absolute value and $a = \max\{c, |a_1|, \ldots, |a_n|, 1\}$.*

Proof. Let $\mathfrak{I} = (a_1 \cdot x_1 + \cdots + a_n \cdot x_n, \mathcal{F})$ be an FI-enriched linear program, where $\mathcal{F} = (V, \mathcal{E}, V_F, V_I, I)$. Moreover, let $\mathfrak{I}' = (a_1 \cdot x_1 + \cdots + a_n \cdot x_n, \mathcal{S})$, where $\mathcal{S} = (V', \mathcal{E}', I')$ is an enriched system as described in Proposition 6.2.5.

Observation 6.2.8. *It is immediate from the properties of $\mathcal{S}$ in Proposition 6.2.5 that $\mathfrak{I}$ has the finite optimal value b if and only if b is the optimal value of $\mathfrak{I}'$.*

Furthermore, it is implicit in the proof of Proposition 3.2.22 that we can obtain a set of ordinary systems $\mathfrak{S}$ such that the set of solutions to $\mathcal{S}$ over $\mathbb{N}$ coincides with the set consisting of all functions that are solutions over $\mathbb{N}$ to some $\mathcal{S}' \in \mathfrak{S}$. More precisely, the set $\mathfrak{S}$ is obtained as follows:

$$\mathfrak{S} = \{(V', \mathcal{E}' \cup \mathcal{X}) : |\mathcal{X}| = I \text{ and for each } \alpha \implies \beta \in I, \text{ either } \overline{\alpha} \in \mathcal{X} \text{ or } \beta \in I\},$$

where α denotes the inequality corresponding to the negation of α.

Let $\mathfrak{P} = \{(a_1 \cdot x_1 + \cdots + a_n \cdot x_n, \mathcal{S}') : \mathcal{S}' \in \mathfrak{S}\}$. It is now easy to show that $\mathfrak{I}$ has a finite optimal value b if and only if all ILPs in $\mathfrak{P}$ have optimal solutions and the following holds:

$$\max\{a_1 \cdot S(x_1) + \cdots + a_n \cdot S(x_n) \ : \ S \text{ is an optimal solution to some ILP in } \mathfrak{P}\} = b.$$

($\Rightarrow$:) Assume that $\mathfrak{I}$ has an optimal solution S with a finite optimal value b. Due to Observation 6.2.8, b is the optimal value of $\mathfrak{I}'$. This means that there is a solution S' to $\mathcal{S}$ over $\mathbb{N}$ such that $a_1 \cdot S'(x_1) + \cdots + a_n \cdot S'(x_n) = b$. Furthermore, S' must be a solution to some ILP in $\mathfrak{P}$ and so, we have:

$$\max\{a_1 \cdot S(x_1) + \cdots + a_n \cdot s(x_n) \ : \ S \text{ is an optimal solution to some ILP in } \mathfrak{P}\} \geq b.$$

Let $\mathcal{S}'$ be an ILP in $\mathfrak{S}$ and assume towards a contradiction that either (i) $\mathcal{S}'$ in $\mathfrak{S}$ that does not have a finite optimal value or (ii) $\mathcal{S}'$ has an optimal solution S' s.t. $a_1 \cdot S'(x_1) + \cdots + a_n \cdot S'(x_n) > b$.

(i) Assume that $\mathcal{S}'$ does not have an optimal solution. In particular, this means that for every solution S_1 of $\mathcal{S}'$, there is another solution S_2 of $\mathcal{S}'$ s.t. $a_1 \cdot S_1(x_1) \cdots a_n \cdot S_1(x_n) < a_1 \cdot S_2(x_1) \cdots a_n \cdot S_2(x_n)$. As all of these solutions are also solutions of $\mathcal{S}$, it follows that there exists a solution S'' of $\mathcal{S}$ over $\mathbb{N}$ for which $a_1 \cdot S''(x_1) + \cdots + a_n \cdot S''(x_n) > b$ which is a contradiction to the optimality of b.

(ii) Assume that $\mathcal{S}'$ has an optimal solution S'' s.t. $a_1 \cdot S''(x_1) + \cdots + a_n \cdot S''(x_n) > b$. The argument is now the same as in the previous case. This S'' is also a solution to $\mathcal{S}$ which is a contradiction to the optimality of b.

($\Leftarrow$:) Assume that all ILPs in $\mathfrak{P}$ have optimal solutions and the following holds:

$$\max\{a_1 \cdot S(x_1) + \cdots + a_n \cdot S(x_n) \ : \ S \text{ is an optimal solution to some ILP in } \mathfrak{P}\} = b.$$

Assume towards a contradiction that (I) one of the following holds: (i) has no optimal solution, (ii) has an optimal solution S s.t. $a_1 \cdot S(x_1) + \cdots + a_n \cdot S(x_n) = \aleph_0$, or (iii) has a finite optimal value that is greater than b.

(i) Assume that $\mathcal{I}$ has no optimal solution. In particular, this means that for every solution S_1 of $\mathcal{F}$ over $\mathbb{N}^*$, there is another solution S_2 of $\mathcal{F}$ over $\mathbb{N}^*$ s.t. $a_1 \cdot S_1(x_1) \cdots a_n \cdot S_1(x_n) < a_1 \cdot S_2(x_1) \cdots a_n \cdot S_2(x_n)$. We also point out that in this case, none of the $S(x_1), \ldots, S(x_n)$ can be $\aleph_0$, since according to our definition of optimal solutions, that would make S optimal. Due to the properties of $\mathcal{S}$, we can conclude that $\mathcal{S}$ also does not have an optimal solution and hence there must be at least one ILP in $\mathfrak{S}$ for which the same holds, contradicting our assumption.

(ii) Assume that $\mathcal{I}$ has an optimal solution S s.t. $a_1 \cdot S(x_1) + \cdots + a_n \cdot S(x_n) = \aleph_0$. In this case, we can show that the enriched system $\mathcal{S}$ once again has no solution that maximizes $a_1 \cdot x_1 + \cdots + a_n \cdot x_n$. Indeed, for each $n \geq 0$, the function $S' : V' \to \mathbb{N}$ that assigns $S'(x) = S(x)$, if $S(x) \neq \aleph_0$, and $S(x) = 1 + c + \sum_{x \in V, S(x) \neq \aleph_0} c \cdot S(x) + n$, otherwise, where c be the maximal coefficient in $\mathcal{F}$, is a solution to $\mathcal{S}$. The rest of the argument is now the same as in the previous case.

(iii) Assume that $\mathcal{I}$ has a finite optimal value b' s.t. $b' > b$. Then, $\mathfrak{I}'$ has the optimal value b', which means there is a solution S' to $\mathcal{S}$ such that $a_1 \cdot S'(x_1) \cdots a_n \cdot S'(x_n) = b'$.

This means that there must be some ILP in $\mathfrak{S}$ s.t. S' is a solution to it, which is a contradiction to b being the maximal value among the optimal values of ILPs in $\mathfrak{S}$.

By the construction of $\mathfrak{S}$ and the enriched system $\mathcal{S}$, we have that the following holds for each $\mathcal{S}' = (V'', \mathcal{E}'') \in \mathfrak{S}$:

- $|V''| = |V'| \leq 2|V|$,
- $|\mathcal{E}''| \leq |\mathcal{E}'| + |\mathcal{I}'| \leq 2|\mathcal{E}| + |I| + |V_F| + |V_I|$, and
- every constant occurring in $\mathcal{S}$ (and therefore in $\mathcal{F}$) also occurs in $\mathcal{S}'$.

In view of Proposition 6.2.4, we therefore have that the value

$$\max\{a_1 \cdot S(x_1) + \cdots + a_n \cdot S(x_n) \,:\, S \text{ is an optimal solution to some ILP in } \mathfrak{P}\}$$

is bounded by

$$(a+1)^{16(2|V|+(2|\mathcal{E}|+|I|+|V_F|+|V_I|)+2|V|\cdot(2|\mathcal{E}|+|I|+|V_F|+|V_I|)},$$

and thus also by

$$(a+1)^{16(2|\mathcal{E}|+|I|+|V_F|+|V_I|+1)\cdot(2|V|+1)},$$

where $a = \max\{|c|, |a_1|, \ldots, |a_n|, 1\}$. The same holds for the finite optimal value of $\mathfrak{I}$, if it exists.

□

6.3 The case of $\mathcal{ALCHOIQ}$

Due to its support for number restrictions, $\mathcal{ALCHOIQ}$ stands out as a natural candidate to investigate boundedness of predicates. In view of Proposition 6.1.5, we focus on the problem of FI-satisfiability in $\mathcal{ALCHOIQ}$ and we show how we can adapt the satisfiability procedure for $\mathcal{ALCHOIQ}$ with closed predicates from Chapter 3 into a worst-case complexity-optimal decision procedure for FI-SAT($\mathcal{ALCHOIQ}$). We then proceed to show that there is a function depending on a given TBox $\mathcal{T}$ and a set Σ of closed predicates, that computes for every $n \geq 0$ an upper bound on the size of extensions of bounded predicates in the models of $(\mathcal{T}, \Sigma, \mathcal{A})$, for all ABoxes $\mathcal{A}$ of size n.

Before we begin, we make a simplifying assumption. Let $(\mathcal{T}, \Sigma_F, \Sigma_I)$ be an instance of FI-SAT($\mathcal{ALCHOIQ}$). For ease of presentation, we assume that Σ_F and Σ_I contain only concept names. This is not a limitation, as role names can be eliminated from both sets in polynomial time while preserving FI-satisfiability. This is done by introducing a fresh concept name A_r, for each role name $r \in \Sigma_F \cup \Sigma_I$, and adding axioms that ensure A_r collects the domain elements in the domain and range of r. Finally, r is replaced by A_r in Σ_F and Σ_I. More precisely, let $\mathcal{T}'$ be the following TBox:

$$\mathcal{T}' = \mathcal{T} \cup \{\exists r.\top \sqcup \exists r^{-}.\top \equiv A_r : r \in \Sigma_F \cup \Sigma_I, \}$$

where A_r is a fresh concept name. Moreover, let $\Sigma'_F = (\Sigma_F \cap \mathsf{N_C}) \cup \{A_r : r \in \Sigma_F\}$ and $\Sigma'_I = (\Sigma_I \cap \mathsf{N_C}) \cup \{A_r : r \in \Sigma_I\}$. Then, $(\mathcal{T}, \Sigma_F, \Sigma_I)$ is a yes-instance of FI-SAT($\mathcal{ALCHOIQ}$) if and only if $(\mathcal{T}', \Sigma'_F, \Sigma'_I)$ is a yes-instance of FI-SAT($\mathcal{ALCHOIQ}$).

The main idea behind the FI-satisfiability procedure is the same as before: for an instance $(\mathcal{T}, \Sigma_F, \Sigma_I)$ of FI-SAT($\mathcal{ALCHOIQ}$), we want to build an enriched system of integer linear inequalities whose solutions over $\mathbb{N}^*$ correspond to the models of $\mathcal{T}$ in which all predicates in Σ_F have finite extensions and all predicates in Σ_I have infinite extensions. To this end, we rely on the results from Chapter 3 where we showed that given an $\mathcal{ALCHOIQ}$ KB $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$, we can build an enriched system $\mathcal{S}_\mathcal{K}$ whose solutions over $\mathbb{N}^*$ correspond to the models of $\mathcal{K}$ (see Definition 3.2.23 and Proposition 3.2.24). For a given $\mathcal{ALCHOIQ}$ TBox $\mathcal{T}$, we can readily reuse this characterization to define an enriched system $\mathcal{S}_\mathcal{T} = \mathcal{S}_{(\mathcal{T},\emptyset,\emptyset)}$ whose solutions over $\mathbb{N}^*$ correspond to the models of $\mathcal{T}$. However, we still need to filter out those solutions that correspond to models that violate the constraints on the predicate extensions given by Σ_F and Σ_I. To this end, we turn to the FI-enriched systems of the previous section.

Definition 6.3.1. *Let $\mathcal{T}$ be an $\mathcal{ALCHOIQ}$ TBox, and $\Sigma_F, \Sigma_I \subseteq \mathsf{N_C}(\mathcal{T})$ be two sets of concept names that occur in $\mathcal{T}$. Furthermore, let $\mathcal{K}$ denote the following $\mathcal{ALCHOIQ}$ KB with closed predicates: $\mathcal{K} = (\mathcal{T}, \emptyset, \emptyset)$ and let $\mathcal{S}_\mathcal{K} = (V, \mathcal{E}, I)$ be the enriched system as given in Definition 3.2.23. We denote by $\mathcal{F}_{(\mathcal{T},\Sigma_F,\Sigma_I)}$ the FI-enriched system $(V, \mathcal{E}, V_F, V_I, I)$, where:*

- $V_F = \{x_{(T,\rho)} : (T, \rho) \in \mathit{Tiles}(\mathcal{K})$ *and* $T \cap \Sigma_F \neq \emptyset\}$, *and*
- $V_I = \{\{x_{(T,\rho)} : (T, \rho) \in \mathit{Tiles}(\mathcal{K})$ *and* $A \in T\} : A \in \Sigma_I\}$.

Proposition 6.3.2. *Let $\mathcal{T}$ be an $\mathcal{ALCHOIQ}$ TBox, and $\Sigma_F, \Sigma_I \subseteq \mathsf{N_C}(\mathcal{T})$. The FI-enriched system $\mathcal{F}_{(\mathcal{T},\Sigma_F,\Sigma_I)}$ has the following properties:*

- *$\mathcal{F}_{(\mathcal{T},\Sigma_F,\Sigma_I)}$ is exponential in the size of $\mathcal{T}$,*
- *$\mathcal{F}_{(\mathcal{T},\Sigma_F,\Sigma_I)}$ has a solution S over $\mathbb{N}^*$ if and only if there exists an FI-model $\mathcal{I}$ of $(\mathcal{T}, \Sigma_F, \Sigma_I)$.*

Proof. The proof of the proposition above follows from the results presented in Chapter 3. More precisely, the underlying enriched system of $\mathcal{F}_{(\mathcal{T},\Sigma_F,\Sigma_I)}$ is simply the enriched system $\mathcal{S}_\mathcal{K}$ for the $\mathcal{ALCHOIQ}$ KB $\mathcal{K} = (\mathcal{T}, \emptyset, \emptyset)$, where the set of closed predicates and the ABox are empty. Recall that $\mathcal{S}_\mathcal{K}$ was obtained by translating the conditions placed on the mosaics for $\mathcal{K}$ into an enriched system, and due to Proposition 3.2.24, there is a each solution to $\mathcal{S}_\mathcal{K}$ can be seen as a mosaic for $\mathcal{K}$ and vice versa. Furthermore, Theorem 3.2.12 states that $\mathcal{K}$ is satisfiable if and only if it has a mosaic. Moreover, due to Observation 3.2.25, we know that $\mathcal{S}_\mathcal{K}$ has the following property: if $\mathcal{K}$ has a model $\mathcal{I}$, then $\mathcal{S}$ has a solution S in which $|A^\mathcal{I}| = \sum_{(T,\rho)\in \mathrm{Tiles}(\mathcal{K}), A\in T} S(x_{(T,\rho)})$, for

all $A \in \mathsf{N_C}(\mathcal{T})$. Moreover, if $\mathcal{S}_\mathcal{K}$ has a solution S then $\mathcal{K}$ has a model $\mathcal{I}$ in which $|A^\mathcal{I}| = \sum_{(T,\rho)\in \mathrm{Tiles}(\mathcal{K}), A\in T} S(x_{(T,\rho)})$, for all $A \in \mathsf{N_C}(\mathcal{T})$.

This already explains the intuition behind the sets V_F and V_I. Let A be an arbitrary concept name in Σ_F. In order to filter out the solutions that lead to models in which A has an infinite extension, we need to make sure that the solution S of $\mathcal{F}_{(\mathcal{T},\Sigma_F,\Sigma_I)}$ assigns a finite value to *every* variable representing a tile with A in its unary type. This is done by adding all such variables to V_F. Furthermore, let B be an arbitrary concept name in Σ_I. We ensure that we only consider solutions to $\mathcal{F}_{(\mathcal{T},\Sigma_F,\Sigma_I)}$ which correspond to models of $\mathcal{T}$ in which B has an infinite extension by making sure that all solutions assign $\aleph_0$ to *at least one* of the variables that represents a tile with B in its unary type. To this end, we add the set $\{x_{(T,\rho)} : (T,\rho) \in \mathrm{Tiles}(\mathcal{K}) \text{ and } B \in T\}$ to V_I. It is now easy to see that $\mathcal{F}_{(\mathcal{T},\Sigma_F,\Sigma_I)}$ has a solution S over $\mathbb{N}^*$ if and only if there exists an FI-model $\mathcal{I}$ of $(\mathcal{T},\Sigma_F,\Sigma_I)$.

Regarding the size of the FI-system, we already know that the enriched system $\mathcal{S}_{(\mathcal{K})}$ is exponential in the size of $\mathcal{K}$ (i.e., in the size of $\mathcal{T}$). As we have that $|V_F| \leq |V|$ and $|V_I| \leq \Sigma_I$ as well as $|X| \leq |V|$, for each set $X \in V_I$, we have that $\mathcal{F}_{(\mathcal{T},\Sigma_F,\Sigma_I)}$ is also exponential in the size of $\mathcal{K}$. □

Theorem 6.3.3. *FI-SAT($\mathcal{ALCHOIQ}$) is* NExpTime*-complete.*
*Boundedness($\mathcal{ALCHOIQ}$) is co-*NExpTime*-complete.*

Proof. Let $\mathcal{T}$ be an $\mathcal{ALCHOIQ}$ TBox, and $\Sigma_F, \Sigma_I \subseteq \mathsf{N_C}(\mathcal{T})$. In view of Proposition 6.3.2, we know that there is an enriched system $\mathcal{F}_{(\mathcal{T},\Sigma_F,\Sigma_I)}$ exponential in the size of $\mathcal{T}$ s.t. $\mathcal{F}_{(\mathcal{T},\Sigma_F,\Sigma_I)}$ has a solution over $\mathbb{N}^*$ if and only if $(\mathcal{T},\Sigma_F,\Sigma_I)$ is a yes-instance of FI-SAT($\mathcal{ALCHOIQ}$). Due to Proposition 6.2.6, we can decide whether $\mathcal{F}_{(\mathcal{T},\Sigma_F,\Sigma_I)}$ has a solution over $\mathbb{N}^*$ in nondeterministic polynomial time in the size of $\mathcal{F}_{(\mathcal{T},\Sigma_F,\Sigma_I)}$, i.e., in nondeterministic exponential time in the size of $\mathcal{T}$. Hence, FI-SAT($\mathcal{ALCHOIQ}$) is feasible in NExpTime. The matching lower bound comes from the fact that deciding ordinary satisfiability is already a NExpTime-complete problem in $\mathcal{ALCHOIQ}$ [Tob00]. Finally, due to Corollary 6.1.6, we have that Boundedness($\mathcal{ALCHOIQ}$) is a co-NExpTime-complete problem. □

6.3.1 Size of Bounded Extensions

The main advantage of predicates that are bounded w.r.t. a given TBox $\mathcal{T}$ and a set Σ of predicates occurring in $\mathcal{T}$ is that we can readily compute an upper bound on the number of different objects that can participate in their extensions that is dependent on the shape of $\mathcal{T}$ and the number of objects in the extensions of the predicates in Σ. The remainder of this section is dedicated to exactly that – we compute a function $f_{\mathcal{T},\Sigma} : \mathbb{N} \to \mathbb{N}$ s.t. $\sum_{p\in \mathsf{B_P}(\mathcal{T},\Sigma)}(|p^\mathcal{I}|) \leq f_{\mathcal{T},\Sigma}(n)$, for every model $\mathcal{I}$ of $\mathcal{T}$ in which, for all concept names $A \in \Sigma$, there are at most n domain elements participating in A. In the first line, we only consider the case where bounded predicates and predicates in Σ are concept names, as roles can be eliminated using the same trick as that was used in the previous section. To

compute $f_{\mathcal{T},\Sigma}$, we proceed as follows. We once again rely on our results from Chapter 3 and we consider the enriched system $\mathcal{S}_{(\mathcal{T},\emptyset,\emptyset)}$ whose solutions over $\mathbb{N}^*$ correspond to the models of $\mathcal{T}$. Furthermore, let $A_1, \ldots, A_{|\Sigma|}$ be an arbitrary enumeration of the concept names in Σ and assume that we are given an vector $\vec{b} = (b_1, \ldots, b_{|\Sigma|})$. We next show that we can build an FI-enriched ILP $\mathfrak{I}_{(\mathcal{T},\Sigma,\vec{b})}$ whose optimal value is the upper bound on the cumulative size of extensions of concept names that are strongly bounded w.r.t. $\mathcal{T}$ and Σ in every model $\mathcal{I}$ of $\mathcal{T}$ in which $|A_i|^{\mathcal{I}} = b_i$, for every $1 \leq i \leq |\Sigma|$. We formalize this in the proposition below.

Proposition 6.3.4. *Let $\mathcal{T}$ be an $\mathcal{ALCHOIQ}$ TBox, $\Sigma \subseteq \mathsf{N_C}(\mathcal{T})$, $\vec{b} = (b_1, \ldots, b_{|\Sigma|})$ be a vector over $\mathbb{N}$. We can build an FI-enriched system $\mathcal{F}_{(\mathcal{T},\Sigma,\vec{b})} = (V, \mathcal{E}, V_F, V_I, I)$ and an FI-enriched ILP $\mathfrak{I}_{(\mathcal{T},\Sigma,\vec{b})} = (f, \mathcal{F}_{(\mathcal{T},\Sigma,\vec{b})})$ with the following properties:*

- *$\mathfrak{I}_{(\mathcal{T},\Sigma,\vec{b})}$ has a finite optimal value $b_{(\mathcal{T},\Sigma,\vec{b})}$.*
- *for every model of $\mathcal{I}$ of $\mathcal{T}$ in which, for all $1 \leq i \leq |\Sigma|$, $|A_i^{\mathcal{I}}| = b_i$, we have that $\sum_{B \in \mathsf{B_C}(\mathcal{T},\Sigma)}(|B^{\mathcal{I}}|) \leq b_{(\mathcal{T},\Sigma,\vec{b})}$.*
- *$|V|, |\mathcal{E}|, |I| \leq 3(2^{l_{\mathcal{T}}} \cdot (|\mathsf{N_I}(\mathcal{T})| + 1))^{m_{\mathcal{T}} \cdot c_{\mathcal{T}} + 2}$ and $|V_F| = |V_I| = 0$, where $l_{\mathcal{T}} = |\mathsf{N_C}(\mathcal{T})| + |\mathsf{N_R^+}(\mathcal{T})|$, $c_{\mathcal{T}}$ is the maximum integer occurring in $\mathcal{T}$.*
- *$c \leq \max\{1, c_{\mathcal{T}}, b_1, \ldots, b_{|\Sigma|}\}$, where c is a coefficient occurring in $\mathcal{F}_{(\mathcal{T},\Sigma,\vec{b})}$ such that $|c| \geq |c'|$, for all coefficients c' of $\mathcal{F}_{(\mathcal{T},\Sigma,\vec{b})}$.*

Proof. We first give a definition of isomorphic ABoxes w.r.t. a given TBox $\mathcal{T}$, as well as a lemma that shows that if some concept name B has only extensions of bounded size in models of $(\mathcal{T}, \Sigma, \mathcal{A})$, for some ABox $\mathcal{A}$ over Σ, the same bound applies to the extensions of B in models of $(\mathcal{T}, \Sigma, \mathcal{A}')$, for any ABox $\mathcal{A}'$ that is isomorphic to $\mathcal{A}$ w.r.t. $\mathcal{T}$.

Definition 6.3.5. *Let $\mathcal{T}$ be an $\mathcal{ALCHOIQ}$ TBox, $\Sigma \subseteq \mathsf{N_C}(\mathcal{T})$ and $\mathcal{A}_1, \mathcal{A}_2$ be two ABoxes over the signature of $\mathcal{T}$. We say that $\mathcal{A}_1$ and $\mathcal{A}_2$ are isomorphic w.r.t. $\mathcal{T}$ if there exists a bijection $f : \mathsf{N_I}(\mathcal{A}_1) \to \mathsf{N_I}(\mathcal{A}_2)$ s.t. $\mathcal{A}_2 = \{A(f(c)) : A(c) \in \mathcal{A}_1)\} \cup \{r(f(c), f(d)) : r(c,d) \in \mathcal{A}_1)\}$, and if $c \in \mathsf{N_I}(\mathcal{T})$, then $f(c) = c$.*

Lemma 6.3.6. *Let $\mathcal{T}$ be an $\mathcal{ALCHOIQ}$ TBox, $\Sigma \subseteq \mathsf{N_C}(\mathcal{T})$, $\mathcal{A}_1, \mathcal{A}_2$ be two ABoxes over the signature of $\mathcal{T}$ that are isomorphic w.r.t. $\mathcal{T}$. Let B be a concept name and assume that there is a natural number b s.t. for every model $\mathcal{I}$ of $(\mathcal{T}, \Sigma, \mathcal{A}_1)$, $|B^{\mathcal{I}}| \leq b$. Then $|B^{\mathcal{J}}| \leq b$ also holds for every model $\mathcal{J}$ of $(\mathcal{T}, \Sigma, \mathcal{A}_2)$.*

Let now $\mathcal{S}_{(\mathcal{T},\emptyset,\emptyset)} = (V, \mathcal{E}, I)$ be an enriched system obtained from $\mathcal{T}$ that has the properties given in Definition 3.2.23. Let $\mathcal{F}_{(\mathcal{T},\Sigma,\vec{b})} = (V, \mathcal{E}', \emptyset, \emptyset, I)$ be an FI-enriched system, where $\mathcal{E}'$ is obtained from $\mathcal{E}$ by adding the following set of inequalities, for each $1 \leq i \leq |\Sigma|$:

$$\sum_{\substack{(T,\rho) \in \mathrm{Tiles}(\mathcal{T}),\\ A_i \in T}} x_{(T,\rho)} \leq b_i \qquad \text{and} \qquad \sum_{\substack{(T,\rho) \in \mathrm{Tiles}(\mathcal{T}),\\ A_i \in T}} x_{(T,\rho)} \geq b_i.$$

It is not hard to see that the following holds:

- for every model $\mathcal{I}$ of $\mathcal{T}$ s.t. $|A_i^{\mathcal{I}}| = b_i$, for every $1 \leq i \leq |\Sigma|$, there is a solution S of $\mathcal{F}_{(\mathcal{T},\Sigma,\vec{b})}$ s.t. $\sum_{\substack{(T,\rho)\in \mathrm{Tiles}(\mathcal{T}),\\ A_i \in T}} S(x_{(T,\rho)}) = b_i$, for all $1 \leq i \leq |\Sigma|$, and
- for every solution S of $\mathcal{F}_{(\mathcal{T},\Sigma,\vec{b})}$, there is a model $\mathcal{I}$ of $\mathcal{T}$ s.t. $|A_i^{\mathcal{I}}| = b_i$ and $\sum_{\substack{(T,\rho)\in \mathrm{Tiles}(\mathcal{T}),\\ A_i \in T}} S(x_{(T,\rho)}) = |A_i^{\mathcal{I}}|$, for every $1 \leq i \leq |\Sigma|$.

We next define an FI-enriched ILP $\mathfrak{I}_{(\mathcal{T},\Sigma,\vec{b})}$ as follows:

$$\mathfrak{I}_{(\mathcal{T},\Sigma,\vec{b})} = (\sum_{\substack{(T,\rho)\in \mathrm{Tiles}(\mathcal{T}),\\ T\cap \mathsf{B_C}(\mathcal{T},\Sigma)\neq\emptyset}} x_{(T,\rho)}, \mathcal{F}_{(\mathcal{T},\Sigma,\vec{b})}).$$

- We first prove that $\mathfrak{I}_{(\mathcal{T},\Sigma,\vec{b})}$ has a finite optimal value. To this end, let $\Pi_{\vec{b}}$ be the following set of ABoxes over Σ:

  $$\Pi_{\vec{b}} = \{\mathcal{A} : \mathcal{A} \text{ is an ABox over } \Sigma \text{ s.t. } |\mathcal{A}|_{A_i}| = b_i, \text{ for all } 1 \leq i \leq |\Sigma|\}.$$

  There are only finitely many (say n) different ABoxes in $\Pi_{\mathcal{A}}$ (up to isomorphism w.r.t. $\mathcal{T}$). As every $B \in \mathsf{B_C}(\mathcal{T},\Sigma)$ is bounded w.r.t. $\mathcal{T}$ and Σ, by definition of boundedness and due to Lemma 6.3.6, there exist some $l_1, \ldots, l_n \in \mathbb{N}$ s.t. for every $\mathcal{A} \in \Pi_{\vec{b}}$ the following holds: if $\mathcal{J}$ is a model of the KB $(\mathcal{T},\Sigma,\mathcal{A})$ with closed predicates, $\sum_{B\in \mathsf{B_C}(\mathcal{T},\Sigma)}(|B^{\mathcal{I}}|) \leq l_i$, for some $1 \leq i \leq n$. Let $l_{\vec{b}} = \max\{l_1, \ldots, l_n\}$. Obviously, for every $\mathcal{A} \in \Pi$ and every model $\mathcal{J}$ of $(\mathcal{T},\Sigma,\mathcal{A})$, $\sum_{B\in \mathsf{B_C}(\mathcal{T},\Sigma)}(|B^{\mathcal{I}}|) \leq l_{\vec{b}}$. Now, let $\mathcal{I}$ be an arbitrary model of $\mathcal{T}$ s.t. $|A_i^{\mathcal{I}}| = b_i$, for all $1 \leq i \leq |\Sigma|$. We can easily extract from $\mathcal{I}$ an ABox $\mathcal{A}$ over Σ s.t. $\mathcal{I}$ is a model of $(\mathcal{T},\Sigma,\mathcal{A})$. Note that $\mathcal{A} \in \Pi_{\vec{b}}$ and therefore, $\sum_{B\in \mathsf{B_C}(\mathcal{T},\Sigma)}(|B^{\mathcal{I}}|) \leq l_{\vec{b}}$. Thus, we conclude that in every model $\mathcal{I}$ of $\mathcal{T}$ s.t. $|A_i^{\mathcal{I}}| = b_i$, for every $1 \leq i \leq |\Sigma|$, $\sum_{B\in \mathsf{B_C}(\mathcal{T},\Sigma)}(|B^{\mathcal{I}}|) \leq l_{\vec{b}}$ holds. This in turn means that for every solution S of $\mathcal{F}_{(\mathcal{T},\Sigma,\vec{b})}$, $\sum_{\substack{(T,\rho)\in \mathrm{Tiles}(\mathcal{T}),\\ T\cap \mathsf{B_C}(\mathcal{T},\Sigma)\neq\emptyset}} S(T,\rho) \leq l_{\vec{b}}$ and so $\mathfrak{I}_{(\mathcal{T},\Sigma,\vec{b})}$ has the finite optimal value $b_{(\mathcal{T},\Sigma,\vec{b})}$ which does not exceed $l_{\vec{b}}$.
- We next prove that the optimal value of $\mathfrak{I}_{(\mathcal{T},\Sigma,\vec{b})}$ represents an upper bound on $\sum_{B\in \mathsf{B_C}(\mathcal{T},\Sigma)}(|B^{\mathcal{I}}|)$ for each model $\mathcal{I}$ of $\mathcal{T}$ in which $|A_i^{\mathcal{I}}| = b_i$, for all $1 \leq i \leq |\Sigma|$. This is almost immediate. Let $\mathcal{I}$ be an arbitrary such model of $\mathcal{T}$. Due to the way $\mathcal{F}_{(\mathcal{T},\Sigma,\vec{b})}$ was constructed, we know that there exists a solution S of $\mathcal{F}_{(\mathcal{T},\Sigma,\vec{b})}$ s.t. $\sum_{B\in \mathsf{B_C}(\mathcal{T},\Sigma)}(|B^{\mathcal{I}}|) = \sum_{(T,\rho)\in \mathrm{Tiles}(\mathcal{T}),T\cap \mathsf{B_C}(\mathcal{T},\Sigma)\neq\emptyset} S(x_{(T,\rho)})$ (see Observation 3.2.25). By definition of optimal values, we have that $\sum_{(T,\rho)\in \mathrm{Tiles}(\mathcal{T}),T\cap \mathsf{B_C}(\mathcal{T},\Sigma)\neq\emptyset} S(x_{(T,\rho)}) \leq b_{(\mathcal{T},\Sigma,\vec{b})}$ and thus $\sum_{B\in \mathsf{B_C}(\mathcal{T},\Sigma)}(|B^{\mathcal{I}}|) \leq b_{(\mathcal{T},\Sigma,\vec{b})}$.

- Finally, we show the desired bounds on the size of $\mathcal{F}_{(\mathcal{T},\Sigma,\vec{b})}$. By construction, $|V_F| = |V_I| = 0$. From Observations 3.2.26 and 3.2.28, using simple calculations we can see that both $|V|$ and $|I|$ are less than or equal to $3(2^{l_\mathcal{T}} \cdot (|\mathsf{N_I}(\mathcal{T})| + 1))^{|\mathcal{T}|\cdot c_\mathcal{T}+2}$. From Observation 3.2.27 and the fact that we only add at most $2n_\mathcal{T}$ new inequalities, we obtain the same bound on $|\mathcal{E}|$.

  To show that for the maximal coefficient c of $\mathcal{F}_{(\mathcal{T},\Sigma,\vec{b})}$, $c \leq \max\{c_\mathcal{T}, b_1, \ldots, b_{|\Sigma|}, 1\}$ holds, we rely on Observation 3.2.29 telling us that every coefficient c' of $\mathcal{S}_{(\mathcal{T},\emptyset,\emptyset)}$ is in $\{0, \pm 1, \ldots, \pm c_\mathcal{T}\}$. As we obtain $\mathcal{F}_{(\mathcal{T},\Sigma,\vec{b})}$ by adding to $\mathcal{S}_{(\mathcal{T},\emptyset,\emptyset)}$ the inequalities whose coefficients are in $\{0, \pm 1, \pm b_1, \ldots, \pm b_{|\Sigma|}\}$, the result follows.

□

Before we state the main result of this subsection, we still need to make a couple final observations.

Observation 6.3.7. *Given two vectors $\vec{b} = (b_1, \ldots, b_n)$ and $\vec{a} = (a_1, \ldots, a_n)$, FI-enriched ILPs $\mathfrak{I}_{(\mathcal{T},\Sigma,\vec{a})}$ and $\mathfrak{I}_{(\mathcal{T},\Sigma,\vec{b})}$ differ only in terms of coefficients that occur in the underlying FI-enriched systems $\mathcal{F}_{(\mathcal{T},\Sigma,\vec{b})}$ and $\mathcal{F}_{(\mathcal{T},\Sigma,\vec{a})}$.*

In particular, together with Proposition 6.2.7, the previous observation implies the following.

Observation 6.3.8. *Given two vectors $\vec{b} = (b_1, \ldots, b_n)$ and $\vec{a} = (a_1, \ldots, a_n)$, if $\max\{|a_1|, \ldots, |a_n|\} = \max\{|b_1|, \ldots, |b_n|\}$, then the optimal values of $\mathfrak{I}_{(\mathcal{T},\Sigma,\vec{a})}$ and $\mathfrak{I}_{(\mathcal{T},\Sigma,\vec{b})}$ coincide.*

Finally, this observation in conjunction with Propositions 6.2.7 and 6.3.4 leads us to the following result:

Theorem 6.3.9. *Let $\mathcal{T}$ be an $\mathcal{ALCHOIQ}$ TBox and $\Sigma \subseteq \mathsf{N_C}(\mathcal{T})$. Furthermore, let $f_{\mathcal{T},\Sigma} : \mathbb{N} \to \mathbb{N}$ s.t. $f_{\mathcal{T},\Sigma}(n) = (a+1)^{16\cdot 12\cdot e^2}$, where $e = 3(2^{l_\mathcal{T}} \cdot (|\mathsf{N_I}(\mathcal{T})| + 1))^{m_\mathcal{T}\cdot c_\mathcal{T}+2}$, $l_\mathcal{T} = |\mathsf{N_C}(\mathcal{T})| + |\mathsf{N_R^+}(\mathcal{T})|$ and $a = \max\{1, c_\mathcal{T}, n\}$.*

For every ABox $\mathcal{A}$ over the signature of $\mathcal{T}$ with $\max\{|\mathcal{A}|_A| : A \in \Sigma\} = n$ the following holds: in every model $\mathcal{I}$ of $(\mathcal{T}, \Sigma, \mathcal{A})$,

$$\sum_{B \in \mathsf{B_C}(\mathcal{T},\Sigma)} (|B^\mathcal{I}|) \leq f_{\mathcal{T},\Sigma}(n).$$

Theorem 6.3.9 shows that the desired function $f_{\mathcal{T},\Sigma}(n)$ that computes the upper bound on the cumulative size of extensions of bounded concept names w.r.t. $\mathcal{T}$ and Σ in models in which the extensions of predicates in Σ do not exceed n is doubly-exponential in the size of $\mathcal{T}$ but only polynomial in the size of the data (i.e., n). Note that the bound computed by $f_{\mathcal{T},\Sigma}$ is asymptotically optimal in the sense that there exists a TBox that creates a binary tree of exponential depth, whose every node belongs to a bounded predicate. Such a TBox is later used in the proof of Theorem 6.4.7.

Bringing roles back. We remark that the results stated above are only formulated for concept names but similar bounds hold for bounded role names. Namely, recall that we decide whether a role name r is bounded w.r.t. a TBox $\mathcal{T}$ and a set of predicates Σ by deciding whether the concept name A_r that subsumes its domain and range is bounded w.r.t. $\mathcal{T}$ and Σ. We can use the function $f_{\mathcal{T},\Sigma}$ from the previous theorem to compute the bound on the extensions of A_r in models of $\mathcal{T}$ in which the size of extensions of predicates in Σ does not exceed n. It is easy to see that in such models, the number of pairs of domain elements participating in the extensions of r cannot exceed $f_{\mathcal{T},\Sigma}(n) \cdot f_{\mathcal{T},\Sigma}(n)$.

Finally, we make one last observation on the number of distinct domain elements that can occur in the extensions of bounded predicates.

Theorem 6.3.10. *For a given $\mathcal{T}$ and a set of predicates $\Sigma \subseteq \mathsf{N_C}(\mathcal{T}) \sqcup \mathsf{N_R}(\mathcal{T})$, let $\mathcal{T}' = \mathcal{T} \cup \{\exists r.\top \sqcup \exists r^-.\top \equiv A_r : r \in \mathsf{N_R}(\mathcal{T})\}$ and $\Sigma' = (\Sigma \cap \mathsf{N_C}) \cup \{A_r : r \in \Sigma \cap \mathsf{N_R}\}$. Furthermore, let $\mathcal{A}$ be an arbitrary ABox over the signature of $\mathcal{T}$ with $|\mathsf{N_I}(\mathcal{A})| = n$. Then, in every model $\mathcal{I}$ of $(\mathcal{T}, \Sigma, \mathcal{A})$ we have the following:*

$$|\{d \text{ occurs in } p^{\mathcal{I}} : p \in \mathsf{B_P}(\mathcal{T}, \Sigma)\}| \leq f_{\mathcal{T}',\Sigma'}(n).$$

Proof. First of all, it is easy to see that a concept name $B \in \mathsf{N_C}(\mathcal{T})$ is bounded w.r.t $\mathcal{T}$ and Σ if and only if B is bounded w.r.t. $\mathcal{T}'$ and Σ'. Furthermore, A_r is bounded w.r.t. $\mathcal{T}'$ and Σ' if and only if r is bounded w.r.t. $\mathcal{T}$ and Σ (resp. $\mathcal{T}'$ and Σ').

Let $\mathcal{A}'$ be an ABox obtained from $\mathcal{A}$ as follows $\mathcal{A}' = \mathcal{A}|_{\mathsf{N_C}(\mathcal{T})} \cup \{A_r(a) : \text{ there exists } b \in \mathsf{N_I}(\mathcal{A}) \text{ s.t. } r(a,b) \in \mathcal{A} \text{ or } r(b,a) \in \mathcal{A}\}$. Now, let $\mathcal{I}$ be an arbitrary model of $(\mathcal{T}, \Sigma, \mathcal{A})$. It is easy to see that this model can be extended into a model $\mathcal{I}'$ of $(\mathcal{T}', \Sigma', \mathcal{A}')$ by simply interpreting the fresh predicate A_r as $(A_r)^{\mathcal{I}'} = \{e \in \Delta^{\mathcal{I}} : \text{ there exists } d \in \Delta^{\mathcal{I}} \text{ s.t. } (e,d) \in r^{\mathcal{I}} \text{ or } (d,e) \in r^{\mathcal{I}}\}$. In particular, the extension of A_r in $\mathcal{I}'$ contains all domain elements of $\Delta^{\mathcal{I}}$ that occur in the extension of r in $\mathcal{I}$. This together with the observation that $\mathsf{B_C}(\mathcal{T}, \Sigma) = \mathsf{B_C}(\mathcal{T}', \Sigma')$ as well as that r is bounded w.r.t. $\mathcal{T}$ and Σ if and only if A_r is bounded w.r.t. $\mathcal{T}'$ and Σ' implies the following:

$$\begin{aligned}
&|\{d \text{ occurs in } p^{\mathcal{I}} : p \in \mathsf{B_P}(\mathcal{T}, \Sigma)\}| = \\
&|\{d \text{ occurs in } r^{\mathcal{I}} : r \in \mathsf{B_R}(\mathcal{T}, \Sigma)\} \cup \{d \in B^{\mathcal{I}} : B \in \mathsf{B_C}(\mathcal{T}, \Sigma)\}| = \\
&|\{d \text{ occurs in } r^{\mathcal{I}'} : A_r \in \mathsf{B_C}(\mathcal{T}', \Sigma')\} \cup \{d \in B^{\mathcal{I}'} : B \in \mathsf{B_C}(\mathcal{T}', \Sigma')\}| = \\
&|\{d \in B^{\mathcal{I}'} : B \in \mathsf{B_C}(\mathcal{T}', \Sigma')\}|.
\end{aligned}$$

Obviously, from the way that $\mathcal{A}'$ was defined, we have $|\mathsf{N_I}(\mathcal{A}')| = n$. It now follows from the results above as well as Theorem 6.3.9 that

$$|\{d \text{ occurs in } p^{\mathcal{I}} : p \in \mathsf{B_P}(\mathcal{T}, \Sigma)\}| = |\{d \in B^{\mathcal{I}'} : B \in \mathsf{B_C}(\mathcal{T}', \Sigma')\}| \leq f_{\mathcal{T}',\Sigma'}(n).$$

□

Weak boundedness So far we have focused on showing results for strongly bounded predicates w.r.t. some TBox $\mathcal{T}$ and a set of predicates Σ. Recall that a predicate is strongly bounded w.r.t. $\mathcal{T}$ and Σ, if it is bounded w.r.t. all KBs $(\mathcal{T}, \Sigma, \mathcal{A})$, where $\mathcal{A}$ is an arbitrary ABox over the signature of $\mathcal{T}$. However, it is possible to have a predicate p that is not strongly bounded w.r.t. $\mathcal{T}$ and Σ, but p is nonetheless bounded in some concrete KB $(\mathcal{T}, \Sigma, \mathcal{A})$. We illustrate this on the following example.

Example 6.3.11. *Let $\mathcal{T}$ be a TBox consisting of the following axioms:*

$$\{o\} \sqsubseteq A \sqcup B,$$
$$A \sqcap B \sqsubseteq \bot,$$
$$\top \sqsubseteq \exists r^{-}.\{o\},$$
$$A \sqcap \{o\} \sqsubseteq \leq 1 r.\top.$$

Furthermore, let $\Sigma = \emptyset$. In general no predicates are bounded w.r.t. $\mathcal{T}$ and Σ. To see this, we show that for the empty ABox, there is a model $\mathcal{I}$ of $(\mathcal{T}, \emptyset, \emptyset)$ in which $A^{\mathcal{I}}, B^{\mathcal{I}}$ and $r^{\mathcal{I}}$ are all infinite and therefore unbounded. Indeed, let $\Delta^{\mathcal{I}} = \mathbb{N} \cup \{o\}$ and let $\cdot^{\mathcal{I}}$ be defined as follows:

$$A^{\mathcal{I}} = \{n \in \mathbb{N} : n \text{ is even}\},$$
$$B^{\mathcal{I}} = \{n \in \mathbb{N} : n \text{ is odd}\} \cup \{o\}, \text{ and}$$
$$r^{\mathcal{I}} = \{(o, d) : d \in \Delta^{\mathcal{I}}\}.$$

Consider now the ABox $\mathcal{A}$ consisting of a single fact $A(o)$. Due to the last two axioms of $\mathcal{T}$, it follows that the KB $(\mathcal{T}, \emptyset, \mathcal{A})$ only has models whose domain consists of a single individual o. Thus, all predicates are bounded w.r.t. $(\mathcal{T}, \emptyset, \mathcal{A})$.

We once again use the same role elimination strategy as before and we focus only on bounded concept names. Given a KB $\mathcal{K}$, it is straightforward to devise a procedure that decides whether every concept name in a given a set of concept names Γ is bounded w.r.t. $\mathcal{K}$, and if so, provides an upper bound on the size of $\sum_{B \in \Gamma}(|B^{\mathcal{I}}|)$, in all models $\mathcal{I}$ of $\mathcal{K}$. This is done by taking the enriched system $\mathcal{S}_{\mathcal{K}}$ from Definition 3.2.23 whose solutions correspond to the models of $\mathcal{K}$ and adding an objective function that maximizes the sum of extensions of the concepts in Γ.

Proposition 6.3.12. *Let $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$ be a KB, where $\mathcal{T}$ is an $\mathcal{ALCHOIQ}$ TBox, Σ is a set of predicates occurring in $\mathcal{T}$, and $\mathcal{A}$ is an ABox over the signature of $\mathcal{T}$. Consider a set of concept names $\Gamma \subseteq \mathsf{N_C}(\mathcal{T})$. We can build an FI-enriched ILP $\mathfrak{I}^{\Gamma}_{\mathcal{K}}$ that has a finite optimal value if and only if each $B \in \Gamma$ is bounded w.r.t. $\mathcal{K}$. Moreover, if b is the finite optimal value of $\mathfrak{I}^{\Gamma}_{\mathcal{K}}$, then $\sum_{B \in \Gamma}(|B^{\mathcal{I}}|) \leq b$ for all models $\mathcal{I}$ of $(\mathcal{T}, \Sigma, \mathcal{A})$.*

Proof. Let $\mathcal{S}_{\mathcal{K}} = (V, \mathcal{E}, I)$ be the system from Definition 3.2.23 and let $\mathfrak{I}^{\Gamma}_{\mathcal{K}}$ be the following FI-enriched ILP:

$$\mathfrak{I}^{\Gamma}_{\mathcal{K}} = \Big(\sum_{\substack{(T,\rho) \in \mathrm{Tiles}(\mathcal{T}), \\ T \cap \Gamma \neq \emptyset}} x_{(T,\rho)}, (V, \mathcal{E}, \emptyset, \emptyset, I) \Big).$$

Due to Observation 3.2.25, the result above is immediate. □

Furthermore, from the previously-made observations on the size of $\mathcal{K}$ as well as the fact that we can compute an upper bound on the finite optimal solution of a given FI-enriched ILP, should one exist, we can formulate the following theorem.

Theorem 6.3.13. *W-Boundedness($\mathcal{ALCHOIQ}$) is co-NExpTime-complete. Furthermore, for a given KB $\mathcal{K}$, we can compute an integer bound b that is doubly-exponential in the size of $\mathcal{K}$ s.t. for all models $\mathcal{I}$ of $\mathcal{K}$, $\sum_{p \in \mathsf{B}_P(\mathcal{K})}(|p^{\mathcal{I}}|) \leq b$ holds, where $\mathsf{B}_P(\mathcal{K})$ is the set of all predicates p that are bounded w.r.t. $\mathcal{K}$.*

6.4 Boundedness in Ontology-Mediated Query Answering

We next discuss how the results presented in the previous section can be applied in the context of ontology-mediated query answering. We have already mentioned in the introduction to this thesis that there are very few results on query answering in $\mathcal{ALCHOIQ}$, even without closed predicates. In Chapter 4, we presented a generalization of ontology-mediated instance queries called safe-range OMQs. Moreover, we showed that these queries are Datalog$^{\neg}$-rewritable and we provided tight complexity results for the query answering problem. The main idea behind safe-range OMQs is to allow arbitrary first-order queries as long as the quantification happens only over the known individuals, i.e., those individuals that occur in the TBox or the data. One way to ensure that this safety criterion is met by an OMQ $(\mathcal{T}, \Sigma, q)$ is to guard all non-answer variables in q by closed predicates. The procedure for answering safe-range OMQs over some ABox $\mathcal{A}$ hinges on the fact that the introduced safety criterion ensures that there is a finite upper bound on the number of different domain elements that the variables of q can take on. In order to check whether a certain model $\mathcal{I}$ of $(\mathcal{T}, \Sigma, \mathcal{A})$ is a counter-model to some tuple $\vec{a}$ of individuals being an answer to q over $(\mathcal{T}, \Sigma, \mathcal{A})$, it suffices to consider only the finite part of $\mathcal{I}$ that pertains to the elements that the variables of q range over, in this case the individuals occurring in $\mathcal{T}$ and $\mathcal{A}$. As there is a bounded number of different possibilities for what this part of $\mathcal{I}$ can look like, we can provide upper complexity bounds for the query answering problem for safe-range OMQs. We exploit the same idea to answer first-order and Datalog OMQs whose variables are not necessarily restricted to range only over the known individuals, but rather to a set of domain elements whose size is finite and known.

6.4.1 Extended Safe-Range OMQs

We next introduce the notion of *relaxed safe-range OMQs* that also allow open predicates to be used as guards, as long as they are bounded. To this end, we import all the notions defined in Section 4.2 and we adapt Definition 4.2.6 as given below.

Definition 6.4.1. *An OMQ $Q = (\mathcal{T}, \Sigma, q)$ is a relaxed safe-range OMQ if*

$$rr(\mathsf{SRNF}(q), \mathsf{B}_P(\mathcal{T}, \Sigma)) \neq \text{'fail'}.$$

We next illustrate this on a short example.

Example 6.4.2. *Consider the OMQ* $Q' = (\mathcal{T}', \Sigma, q(x))$ *similar to the query* $Q = (\mathcal{T}, \Sigma, q(x))$ *from Example 4.2.7 but whose TBox* $\mathcal{T}'$ *contains an additional axiom ensuring that each domain element participating in* B *can have at most one incoming* r*-arc. More precisely, let*

$$\begin{aligned} \mathcal{T} &= \{A \sqsubseteq \geq 1r.B, C \sqsubseteq \geq 1r.B, A \sqcap C \sqsubseteq \bot, B \sqsubseteq \leq 1r^{-}.\top\}, \\ \Sigma &= \{B\}, \\ q(x) &= \exists y \exists z.r(y,x) \wedge r(z,x) \wedge A(y) \wedge C(z). \end{aligned}$$

This query Q' *is not safe-range since the existentially quantified variables* y *and* z *do not occur in positive atoms over closed predicates from* Σ *(i.e.,* B*) and are therefore not recognized as* Σ*-range-restricted, so the procedure* $\mathsf{rr}(\mathsf{SRNF}(q), \Sigma)$ *returns 'fail'. However, notice that both* A *and* C *are bounded w.r.t.* $\mathcal{T}$ *and* Σ*. Thus,* $\mathsf{rr}(\mathsf{SRNF}(q), B_P(\mathcal{T}, \Sigma)) \neq$ *'fail' and so* Q' *is relaxed safe-range.*

We next show that, given a relaxed safe-range OMQ $Q = (\mathcal{T}, \Sigma, q(x_1, \ldots, x_n))$, an ABox $\mathcal{T}$ over the signature of $\mathcal{T}$ and a tuple $\vec{a}$ over the constants occurring in $\mathcal{T}$ and $\mathcal{A}$, we can decide whether $\vec{a}$ is a certain answer to Q over $\mathcal{A}$. Intuitively, every variable occurring in q is guarded by a positive atom over some bounded predicate, which means that in any model $\mathcal{I}$ of $(\mathcal{T}, \Sigma, \mathcal{A})$, q can only be mapped into the part of $\mathcal{I}$ that occurs in the extensions of bounded predicates. Fortunately, in view of Theorem 6.3.10, the function $f_{\mathcal{T},\Sigma}$ provides us with a concrete upper bound on the number of domain elements that this part of $\mathcal{I}$ contains. This leaves us with the following guess-and-check strategy for deciding whether $\vec{a}$ is *not* an answer to q over $(\mathcal{T}, \Sigma, \mathcal{A})$:

- Guess a set $\Delta \subseteq \mathsf{N_I}$ such that $|\Delta| \leq f_{\mathcal{T},\Sigma}(|\mathsf{N_I}(\mathcal{A})|)$ and an ABox $\mathcal{A}'$ with the following properties:
  - for each $a \in \Delta$ and concept name $C \in \mathsf{N_C}(\mathcal{T}) \setminus \Sigma$, either $C(a) \in \mathcal{A}'$ or $\neg C(a) \in \mathcal{A}'$,
  - for each $\{a, b\} \subseteq \Delta$ and role name $r \in \mathsf{N_R}(\mathcal{T}) \setminus \Sigma$, either $r(a, b) \in \mathcal{A}'$ or $\neg r(a, b) \in \mathcal{A}'$,
- Check whether (i) $\vec{a}$ is not an answer to q over $\mathcal{A}'$ and (ii) $(\mathcal{T}, \Sigma, \mathcal{A} \cup \mathcal{A}')$ is satisfiable. If so, then $\vec{a}$ is not a certain answer to Q over $\mathcal{A}$.

Regarding the first guess, in general, as the set $\mathsf{N_I}$ is countably infinite, there are also infinitely many different possibilities for Δ, however, only finitely many up to isomorphism w.r.t. the individuals from $\mathcal{A}$ and $\mathcal{T}$. This, in conjunction with the results from Chapter 3 on the complexity of consistency checking in $\mathcal{ALCHOIQ}$ with closed predicates as well as the results from Theorem 6.3.10, we obtain the following complexity results for answering relaxed safe-range OMQs in $\mathcal{ALCHOIQ}$.

Theorem 6.4.3. *The query answering problem for relaxed safe-range queries mediated by $\mathcal{ALCHOIQ}$ ontologies with closed predicates is* coNP*-complete in data complexity and in co-N2ExpTime in combined complexity.*

Proof. Given a relaxed safe-range OMQ $Q = (\mathcal{T}, \Sigma, q(\vec{x})$, where $\mathcal{T}$ is an $\mathcal{ALCHOIQ}$ TBox and $|\mathsf{N_I}(\mathcal{A})| = n$, in view of Theorem 6.3.10, the set Δ that we have to guess as well as the ABox $\mathcal{A}'$ with the properties listed above are both doubly-exponential in the size of $(\mathcal{T}, \Sigma, \mathcal{A})$, but only polynomial if $\mathcal{T}$ and Σ are considered fixed. Checking whether $\vec{a}$ is an answer to q over $\mathcal{A}'$ can be done in time polynomial in the size of $\mathcal{A}'$. Finally, from the results in Chapter 3 (see Theorem 3.2.32), we know that checking whether $(\mathcal{T}, \Sigma, \mathcal{A} \cup \mathcal{A}')$ is satisfiable can be done by a nondeterministic procedure in the amount of time that is exponential in the size of $\mathcal{T}$ and Σ, but only polynomial in the size of $\mathcal{A} \cup \mathcal{A}'$, i.e., in time that is doubly-exponential in the size of $(\mathcal{T}, \Sigma, \mathcal{A})$.

We have thus shown that deciding whether $\vec{a}$ is not a certain answer to Q over $\mathcal{A}$ is in N2ExpTime and the corresponding co-N2ExpTime upper bound for the combined complexity of the query answering problem of relaxed safe-range OMQs in $\mathcal{ALCHOIQ}$ with closed predicates follows.

Regarding the coNP upper bound on the data complexity of the same problem, we simply note that both Δ and $\mathcal{A}'$ are of size polynomial in $(\mathcal{T}, \Sigma, \mathcal{A})$, if $\mathcal{T}$ and Σ are considered fixed. The matching lower bound comes from Theorem 4.2.12. □

6.4.2 Datalog-Based OMQs

So far, our focus has mostly been on first-order OMQs, i.e., those OMQs whose database query component is given in the form of a first-order formula. As we will see in the rest of this thesis, there has also been significant interest in the DL community in even more expressive OMQ languages like those that couple DL ontologies with *Datalog queries*.

The ability of DLs to assert the existence of anonymous domain elements together with Datalog's support for recursion makes such OMQ languages very powerful and, unsurprisingly, undecidable even for much less expressive DLs than $\mathcal{ALCHOIQ}$ (see, e.g., [LR98, Ros07a]). In order to regain decidability, one needs to somehow limit the number of different domain objects that rule variables can be bound to. This can be done by applying the well-known *DL-safety* restriction on Datalog queries, which ensures that that all query variables are guarded by predicates that do not occur in the TBox [MSS05, Ros05]. Unfortunately, this significantly restricts reasoning about anonymous domain elements. Our next goal is to show that we can relax this safety condition using a strategy similar to the one we used for relaxing safe-range queries. In a nutshell, the new safety condition requires all query variables be guarded by a predicate that is bounded by the input TBox and a given set of closed predicates. This relaxation allows us to support reasoning about anonymous objects, while retaining decidability. We next make this more formal.

We assume a countably infinite set $\mathsf{N_D} \subseteq \mathsf{N_P} \setminus (\mathsf{N_C} \cup \mathsf{N_R})$ of *Datalog predicates*. Furthermore, note that every interpretation $\mathcal{I}$ can be seen as an Herbrand interpretation $I = \{p(\vec{a}) : \vec{a} \in p^{\mathcal{I}}, p \in \mathsf{N_P}\}$ and vice versa. We say that $\mathcal{I}$ is a model of a Datalog program $\mathcal{P}$ if I is a model of $\mathcal{P}$.

Definition 6.4.4. *An* ontology-mediated Datalog query (Datalog OMQ) *is a tuple* $Q = (\mathcal{T}, \Sigma, \mathcal{P}, q)$*, where* $\mathcal{T}$ *is a TBox,* $\Sigma \subseteq \mathsf{N_C} \cup \mathsf{N_R}$*, and* $(\mathcal{P}, q)$ *is a Datalog query with* $q \in \mathsf{N_D}$*. Furthermore, let* $\mathcal{A}$ *be an ABox over the predicates from* $\mathcal{T}$*. A tuple* $\vec{a}$ *of individuals from* $\mathsf{N_I}(\mathcal{T}) \cup \mathsf{N_I}(\mathcal{A}) \cup adom(\mathcal{P})$ *is a* certain answer to Q over $\mathcal{A}$*, if* $\vec{a} \in q^{\mathcal{I}}$*, for every interpretation* $\mathcal{I}$ *that is a model of* $(\mathcal{T}, \Sigma, \mathcal{A})$ *and* $\mathcal{P}$*. The problem of deciding whether* $\vec{a}$ *is a certain answer to* Q *over* $\mathcal{A}$ *is the* query answering problem *for Datalog OMQs.*

To ensure decidability of Datalog OMQs, we define a new safety condition that exploits predicate boundedness.

Definition 6.4.5 (Safe Datalog OMQs)**.** *A rule* ρ *is called* safe *for a given Datalog OMQ* $Q = (\mathcal{T}, \Sigma, \mathcal{P}, q)$ *if every variable of* ρ *occurs in the body of* ρ *in an atom* $p(\vec{t})$ *such that either (i)* $p \in \mathsf{N_D}$ *or (ii)* $p \in \mathsf{B_P}(\mathcal{T}, \Sigma)$*. We say* Q *is* safe *if every* $\rho \in \mathcal{P}$ *is safe for* Q*.*

Example 6.4.6. *Consider the following Datalog OMQ* $Q = (\mathcal{T}', \{\mathsf{Empl}\}, \{\rho\}, \mathsf{pair})$*, where* $\mathcal{T}'$ *is some company ontology that includes the TBox* $\mathcal{T}$ *from Example 6.1.1, and* ρ *is the following rule:*

$$\mathsf{pair}(X, Y) \leftarrow \mathsf{Empl}(X), \mathsf{Empl}(Y), \mathsf{Proj}(Z), \mathsf{assgndTo}(X, Z), \mathsf{assgndTo}(Y, Z)$$

Intuitively, this query computes pairs of employees working on a common project. Observe that Q *is safe as both* Proj *and* assignedTo *are bounded w.r.t.* $\mathcal{T}$ *and* Σ*, but is not DL-safe in the sense of [MSS05, Ros05] as all body atoms of* ρ *involve predicates that occur in* $\mathcal{T}$*.*

We next characterize the complexity of query answering in the proposed OMQ language.

Theorem 6.4.7. *The query answering problem for safe Datalog queries mediated by* $\mathcal{ALCHOIQ}$ *ontologies with closed predicates is co-2NExpTime-complete in combined complexity and coNP-complete in data complexity.*

Proof of Theorem 6.4.7. Assume a safe OMQ $Q = (\mathcal{T}, \Sigma, \mathcal{P}, q)$, an ABox $\mathcal{A}$, and a tuple of $\vec{a}$ of individuals occurring in $\mathcal{T}$, P or $\mathcal{A}$. The upper bound can be shown by a non-deterministic procedure to check that $\vec{a}$ *does not* belong to the answer to Q over $\mathcal{A}$, analogous to the one used in the proof of Theorem 6.4.3. By Theorem 6.3.9, the value $f_{\mathcal{T},\Sigma}(|\mathsf{N_I}(\mathcal{A})|)$ provides an upper bound on the number of distinct elements that can participate in extensions of predicates bounded w.r.t. $\mathcal{T}$ and Σ in any model of $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$. This value is double exponential in the size of $\mathcal{K}$, but polynomial if the size of $\mathcal{T}$ is considered fixed. Let $\mathcal{I}$ be a model of $\mathcal{K}$ and $\mathcal{P}$ with $\vec{a} \notin q^{\mathcal{I}}$ and let $B \subseteq \Delta^{\mathcal{I}}$ be the set of all elements that occur in $\mathcal{I}$ in the extension of some predicate that is bounded

by $\mathcal{T}$ and Σ. Due to our safety condition, there also exists a model of $\mathcal{J}$ of $\mathcal{K}$ and $\mathcal{P}$ such that (i) $\vec{a} \notin q^{\mathcal{J}}$, and (ii) $a_1, \ldots, a_n \in B$ for all $p \in \mathsf{N_D}$ of arity n and all $(a_1, \ldots, a_n) \in p^{\mathcal{J}}$. In other words, if $\vec{a}$ is not an answer to Q over $\mathcal{A}$, then there is a model of $\mathcal{K}$ and P in $\vec{a}$ is not in the extension of q and in which the extensions of program predicates are restricted to elements that occur in the extensions of bounded predicates For this reason, we can use the following guess-and-check procedure to decide whether $\vec{a}$ is not an answer to Q over $\mathcal{A}$:

- Step 1: Guess an interpretation $\mathcal{I}$ such that $\Delta^{\mathcal{I}} \subseteq \mathsf{N_I}$, $|\Delta^{\mathcal{I}}| \leq f_{\mathcal{T},\Sigma}(|\mathsf{N_I}(\mathcal{A})|)$ and $w^{\mathcal{I}} = \emptyset$ for every $w \in \mathsf{N_P}$ that appears neither in $\mathcal{K}$ nor in $\mathcal{P}$.
- Step 2: Construct a new ABox

$$\begin{aligned}\mathcal{A}' = &\{p(\vec{a}) : \vec{a} \in p^{\mathcal{I}}, p \in \mathsf{sig}(\mathcal{T})\} \\ &\cup \{\neg A(c) : c \in \Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}, A \in \mathsf{N_C}(\mathcal{K})\} \\ &\cup \{\neg r(c,d) : (c,d) \in (\Delta^{\mathcal{I}})^2 \setminus r^{\mathcal{I}}, r \in \mathsf{N_R}(\mathcal{T})\}.\end{aligned}$$

- Step 3: Check whether the following hold: (i) $\mathcal{I}$ is a model of $\mathcal{P}$, (ii) $\vec{a} \notin q^{\mathcal{I}}$, and (iii) $\mathcal{K}' = (\mathcal{T}, \Sigma, \mathcal{A} \cup \mathcal{A}')$ is satisfiable. If so, then return *"yes"*.

It is not difficult to verify that $\vec{a}$ is not a certain answer to Q over $\mathcal{A}$ if and only if and only if there exists a guess for $\mathcal{I}$ such that Step 3 of the procedure above returns *"yes"*. In terms of complexity, note that the size of $\mathcal{I}$ produced in Step 1 is bounded by a doubly-exponential function in the size of $\mathcal{K}$. By Theorem 3.2.32, we can decide consistency of $\mathcal{K}' = (\mathcal{T}, \Sigma, \mathcal{A} \cup \mathcal{A}')$ produced in Step 3 non-deterministically in time that is exponential in the size of $\mathcal{T}$ but only polynomial in the size of $\mathcal{A} \cup \mathcal{A}'$, i.e. in the amount of time that is doubly-exponential in size of $\mathcal{K}$. Further, if the size of $\mathcal{T}$ is bounded by a constant, then $\mathcal{I}$ is of polynomial size in the size of $\mathcal{K}$, and thus we can decide consistency of $\mathcal{K}' = (\mathcal{T}, \Sigma, \mathcal{A} \cup \mathcal{A}')$ non-deterministically in polynomial time in the size of $\mathcal{K}$. Finally, checking that $\mathcal{I}$ is a model of $\mathcal{P}$ and that $\vec{a} \notin q^{\mathcal{I}}$ are both polynomial time checks. Thus, the complexity results follow.

co-N2ExpTime Lower Bound. For the lower bound of combined complexity, we provide a reduction from the domino tiling problem for grids of double exponential size.

Definition 6.4.8. *A* domino system *is a tuple $D = (T, H, V)$, where T is a finite set of tiles, and $H, V \subseteq T \times T$. A* tiling *of an $m \times m$ grid w.r.t. to a domino system $D = (T, H, V)$ and an initial condition $(t_1, \ldots, t_n) \in T^n$ is a mapping $\pi : \{0, \ldots, m-1\} \times \{0, \ldots, m-1\} \to T$ such that:*

1. *$((i,j), (i, j+1 \bmod m)) \in V$ for all $0 \leq i, j < m$;*
2. *$((i,j), (i+1 \bmod m, j)) \in H$ for all $0 \leq i, j < m$;*

3. $\pi(0,0) = t_1, \cdots, \pi(n-1,0) = t_n$.

It is a well-known fact that deciding given an domino system D and an initial condition condition $c = (t_1, \ldots, t_n)$ whether there exists a tiling of the $2^{2^n} \times 2^{2^n}$ grid w.r.t. D and c is a N2ExpTime-hard problem [BGG97].

We now provide a reduction from the above problem to the query answering problem for safe Datalog OMQs. For this, assume an arbitrary domino system D and an initial condition condition $c = (t_1, \ldots, t_n)$. We will now construct a TBox $\mathcal{T}$ and a safe program $\mathcal{P}$ such that $\mathcal{T}$ and $\mathcal{P}$ have a model if and only if there exists a tiling of the $2^{2^n} \times 2^{2^n}$ grid w.r.t. D and c. Our goal now is to use inclusions and rules in order to generate a $2^{2^n} \times 2^{2^n}$ grid stored in binary relations H and V. Actually, H and V will encode a torus, i.e. the last column will be connected with the first column, while the last row will be connected to the first row.

The tiling conditions can then be expressed quite easily. Overall the construction consists of 4 parts:

1. (Part A) Using TBox axioms we generate a binary tree of depth 2^n. Using rules, the 2^{2^n} leaves of this tree are linearly ordered and stored in relations First, Last, and Succ. One can see this ordering a counter of rows in our grid.
2. (Part B) For each point in Succ, we again create a separate binary tree of depth 2^n, with its 2^{2^n} leaves again linearly ordered and stored in relations First′, Last′, and Succ′. This provides us 2^{2^n} rows of length 2^{2^n}.
3. (Part C) The different rows are aligned into a grid.
4. (Part D) The assignment of compatible tiles is expressed.

(Part A) The successor relation is populated by enforcing a binary tree of exponential depth, and then collecting its leaf nodes of which we have a double exponential number. We call that we want to create an order of length 2^{2^n}. We will construct a TBox $\mathcal{T}$, and a program $\mathcal{P}$ that force the successor relation to store 2^{2^n} elements in the binary Datalog relation Succ, where in addition the unary relations First and Last are used for the first and the last elements of the ordering, respectively.

We start with the construction of the desired $\mathcal{T}$. Let L, R, E be role names and $B_1, F_1, \ldots, B_n, F_n$ be concept names. Let Node, Leaf be concept names.

To enforce a tree of exponential depth, we use an n-bit counter encoded at an element using the concept names $B_1, \ldots, B_n$. We will use the role E for the child relation. We use Node to indicate a node in our tree. We first state that the individual root corresponds to the root of the tree, which stores the initial counter value (all n bits set to 0). The root node is the only node that is allowed to store this value.

$$\{root\} \sqsubseteq \mathsf{Node} \sqcap \neg B_1 \sqcap \ldots \sqcap \neg B_n \sqcap \neg \exists E^-.\top \qquad \neg B_1 \sqcap \ldots \sqcap \neg B_n \sqsubseteq \{root\}$$

We use Leaf to capture the leaves of our exponentially deep tree. We require that the objects in Leaf are exactly the objects whose counter value is $2^n - 1$ (all bits set to 1). We also require such nodes to not have child nodes.

$$B_1 \sqcap \ldots \sqcap B_n \equiv \mathsf{Leaf} \qquad \mathsf{Leaf} \sqsubseteq \neg\exists E.\top$$

We need to enforce a binary tree. For this, every non-leaf node must have precisely two children, one given via the role L and one via the role R.

$$\neg\mathsf{Leaf} \sqsubseteq {=}1L.\top \sqcap {=}1R.\top \qquad L \sqsubseteq E \qquad R \sqsubseteq E \qquad \exists E^-.\top \sqsubseteq \mathsf{Node} \qquad \mathsf{Node} \sqsubseteq {\leq}2E.\top$$

We require that every object in Node that does not correspond to the root appears somewhere in the tree, which is done by requiring it to be an L-child or an R-child of some node.

$$\mathsf{Node} \sqcap \neg\{\mathit{root}\} \sqsubseteq \exists L^-.\mathsf{Node} \sqcup \exists R^-.\mathsf{Node} \qquad \mathsf{Node} \sqsubseteq {\leq}1E^-.\top$$

It remains now to implement the counter. Assume that a non-leaf node has a counter value c. We need to make sure that its two child nodes have the counter value $c+1$. For this we employ the auxiliary concept names $F_1, \ldots, F_n$. Intuitively, if F_i holds at a point, it means that the i-th bit of the counter needs to be flipped to obtain the next counter value. When performing addition by one, the least significant bit needs to be flipped.

$$\mathsf{Node} \sqsubseteq F_n$$

If the ith bit is 1 and it needs to be flipped, then the $(i-1)$-th bit needs to be flipped as well.

$$\mathsf{Node} \sqcap B_i \sqcap F_i \sqsubseteq F_{i-1} \qquad \text{for all } 1 < i \leq n$$

If the ith bit is 0, then the $(i-1)$-th bit must not be flipped.

$$\mathsf{Node} \sqcap \neg B_i \sqsubseteq \neg F_{i-1} \qquad \text{for all } 1 < i \leq n$$

If the ith bit must not be flipped, then the $(i-1)$-th bit must not be flipped.

$$\mathsf{Node} \sqcap \neg F_i \sqsubseteq \neg F_{i-1} \qquad \text{for all } 1 < i \leq n$$

Once the decision is made on which bits need to be inverted, we can now form the new counter value at the two child nodes:

$$\mathsf{Node} \sqcap B_i \sqcap F_i \sqsubseteq \forall E.\neg B_i \qquad \mathsf{Node} \sqcap \neg B_i \sqcap F_i \sqsubseteq \forall E.B_i \qquad \text{for all } 1 \leq i \leq n$$

$$\mathsf{Node} \sqcap B_i \sqcap \neg F_i \sqsubseteq \forall E.B_i \qquad \mathsf{Node} \sqcap \neg B_i \sqcap \neg F_i \sqsubseteq \forall E.\neg B_i \qquad \text{for all } 1 \leq i \leq n$$

This completes the construction of the TBox $\mathcal{T}$. It is not difficult to see that in any model $\mathcal{I}$ of this TBox, we will have $|\mathsf{Leaf}^{\mathcal{I}}| = 2^{2^n}$, i.e. Leaf has double exponential number of element but is still bounded by $\mathcal{T}$ and $\Sigma = \emptyset$. For this observe that every element in $e \in \mathsf{Leaf}^{\mathcal{I}}$ is identified by a unique word $\sigma_1 \cdots \sigma_{2^n}$ with $\sigma_i \in \{L, R\}$. Observe also that Node is also bounded by $\mathcal{T}$ and $\Sigma = \emptyset$.

We now turn our attention to constructing the program $\mathcal{P}$, which will populate the Succ, First and Last Datalog relations. We need a small addition to the TBox $\mathcal{T}$, for which we use fresh concept names AllL and AllF. We use the following inclusions that will help us to identify the left most leaf node and the right most leaf node in our tree:

$$\{root\} \sqsubseteq \mathsf{AllL} \sqcap \mathsf{AllF} \qquad \mathsf{AllL} \sqsubseteq \forall L.\mathsf{AllL} \qquad \mathsf{AllR} \sqsubseteq \forall R.\mathsf{AllR}$$

We can now state the rules for First and Last:

$$\mathsf{First}(X) \leftarrow \mathsf{Leaf}(X), \mathsf{AllL}(X) \qquad \mathsf{Last}(X) \leftarrow \mathsf{Leaf}(X), \mathsf{AllR}(X)$$

We use the following rule to order the two child nodes of a given node.

$$\mathsf{AuxSucc}(Y, Y') \leftarrow \mathsf{Node}(X), \mathsf{Node}(Y), \mathsf{Node}(Y'), L(X, Y), R(X, Y')$$

If a pair a, b of non-leaf nodes are related by the $\mathsf{AuxSucc}$ relation, then the R-child of a and the L-child of b are related by this relation as well.

$$\mathsf{AuxSucc}(Z, Z') \leftarrow \mathsf{AuxSucc}(Y, Y'), R(Y, Z), L(Y', Z'), \mathsf{Node}(Z), \mathsf{Node}(Z')$$

At the leaf level, $\mathsf{AuxSucc}$ contains the desired successor relation:

$$\mathsf{Succ}(Z, Z') \leftarrow \mathsf{AuxSucc}(Z, Z'), \mathsf{Leaf}(Z), \mathsf{Leaf}(Z')$$

One can check that the program $\mathcal{P}$ is safe. It is also not difficult to see that in any model $\mathcal{I}$ of the constructed TBox $\mathcal{T}$ and the program $\mathcal{P}$, the relation Succ will encode a sequence of 2^{2^n} nodes.

(Part B) We now add further axioms and rules to generate the second level of trees. We take a copy of the axioms and rules obtained in Part A and perform the following steps :

1. Replace every concept name A and role name r by the "primed" version A' of the concept name and r' of the role name, respectively.
2. Replace $\{root\}$ by Leaf.

The effect of these additions is that at every point in Succ, we have a tree hanging whose leaves are linearly ordered and stored in relations First', Last', and Succ'.

(Part C) We now need to align the different rows stored in the Succ' predicate into a grid. This is done using the following rules:

$$\mathsf{hasParent}(X', X) \leftarrow E'(X, X')$$

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

$$\mathsf{ver}(X, X') \leftarrow \mathsf{First}'(X), \mathsf{hasParent}(X, Z), \mathsf{Succ}(Z, Z'), \mathsf{First}'(X'), \mathsf{hasParent}(X', Z')$$

$$\mathsf{ver}(X', X) \leftarrow \mathsf{First}'(X), \mathsf{hasParent}(X, Z), \mathsf{First}(Z), \mathsf{First}'(X'), \mathsf{hasParent}(X', Z'), \mathsf{Last}(Z')$$

$$\mathsf{ver}(Y, Y') \leftarrow \mathsf{ver}(X, X'), \mathsf{Succ}'(X, Y), \mathsf{Succ}'(X', Y')$$

The above populates the relation hor. We can now populate the relation ver, which is simply done by copying the content of Succ', and connecting the last element of the order with the first element of the order (to achieve a torus):

$$\mathsf{hor}(Z, Z') \leftarrow \mathsf{Succ}'(Z, Z')$$

$$\mathsf{SuccTC}'(Z, Z') \leftarrow \mathsf{Succ}(Z, Z')$$

$$\mathsf{SuccTC}'(Z, Z') \leftarrow \mathsf{SuccTC}'(Z, X), \mathsf{Succ}(X, Z')$$

$$\mathsf{hor}(Z', Z) \leftarrow \mathsf{First}'(Z), \mathsf{SuccTC}'(Z, Z'), \mathsf{Last}'(Z')$$

(Part D) We can now express the tiling conditions. For every tile $t \in T$, let A_t be a fresh concept name. We add the following to $\mathcal{T}$ and $\mathcal{P}$ to expressed that one tile needs to be assigned to every element of the grid, and that adjacent tiles must obey the vertical and horizontal compatibility restrictions given in H and V:

$$\mathsf{Leaf}' \sqsubseteq \bigsqcup_{t \in T} \left(A_t \sqcap \bigsqcap_{t' \in T \setminus \{t\}} \neg A_{t'}\right)$$

$$\mathsf{q}(err) \leftarrow \mathsf{Leaf}'(X), A_t(X), \mathsf{Leaf}'(Y), A_{t'}(Y), \mathsf{hor}(X, Y) \qquad \text{for all } (t, t') \in T \times T \setminus H$$

$$\mathsf{q}(err) \leftarrow \mathsf{Leaf}'(X), A_t(X), \mathsf{Leaf}'(Y), A_{t'}(Y), \mathsf{ver}(X, Y) \qquad \text{for all } (t, t') \in T \times T \setminus V$$

Finally, it remains express the initial condition $(t_1, \ldots, t_n)$. We add the following rules for all $1 \leq i \leq n$ and all $t \in T \setminus \{t_i\}$:

$$\mathsf{q}(err) \leftarrow \mathsf{First}'(X_1), \mathsf{Succ}'(X_2), \ldots, \mathsf{Succ}'(X_n), \mathsf{hasParent}(X_1, Z), \mathsf{First}(Z), A_t(X_i).$$

We are now finished with the construction of $\mathcal{T}$ and $\mathcal{P}$. One can verity that a proper tiling for the $2^{2^n} \times 2^{2^n}$ grid w.r.t. D and c grid exists if and only if there exists a model $\mathcal{I}$ of $\mathcal{T}$ and $\mathcal{P}$ such that $\mathcal{I}$ does not satisfy $err \notin \mathsf{q}$, i.e., if err is not a certain answer to $(\mathcal{T}, \emptyset, \mathcal{P}, \mathsf{q})$ over the empty ABox.

□

6.5 Discussion

In this chapter we have presented a method to reason about the number of anonymous objects in the models of KBs written in the expressive DL $\mathcal{ALCHOIQ}$ with closed predicates. This provides a new tool to aid the design of ontologies, also opening the way for sophisticated yet decidable reasoning tasks for data management.

One challenging task left for future work is to provide an implementation of our method for checking boundedness. Due to the large size of the inequality systems, it is clear that we cannot explicitly build them and reuse existing integer programming solvers – this would require (best-case) exponential time. In fact, in Chapter 7, we present a procedure that approximates bounded concept names for $\mathcal{ALCHOIQ}$ (two other less expressive DLs) by simply looking at the shapes of the axioms in the TBox. This procedure yields a subset of strongly bounded concept names for a given TBox $\mathcal{T}$ and a set Σ of closed predicates, their extensions being bounded by a single-exponential function depending on the size of $\mathcal{T}$ and Σ. Another promising way to efficiently recognize predicate boundedness is to consider small systems of inequalities that provide a sound (but incomplete) approximation of the full systems defined here.

In the previous section, we showed that predicates boundedness has application is relaxing the notions of safety for safe-range and rule-based queries over ontologies expressed in the DL $\mathcal{ALCHOIQ}$ with closed predicates. Another application, which we briefly discuss next is verification of temporal properties of evolving graph databases. Namely, we believe that using the same ideas that we used in Section 6.4, we can identify new settings with decidable verification tasks, e.g., by developing methods to identify *state bounded* systems in the sense of [BHCDG$^+$13].

Decidable Verification of Temporal Properties In this setting, we consider databases that evolve over time due to the execution of data-manipulating *actions* by various agents. Effective methods to verify *temporal properties* of such systems are currently unavailable, but would be extremely useful in the design of data-centric applications. The key challenge here is to deal with actions that may introduce *fresh values*, which may result in an *unbounded growth* of the database during the passage of time. Technically speaking, we are interested in the *model checking problem* for (temporal) logic formulae in *infinite-state* transition systems, where each state corresponds to a possible legal database. This problem is undecidable already for very simple action languages and temporal properties, but decidability can be regained, e.g., for the so-called *state bounded* systems, even for rich specifications of temporal properties based on μ-calculus [BHCDG$^+$13]. The latter systems are defined by imposing a global upper bound on the size of the active domain of the database during its evolution. Recognizing state bounded systems is undecidable though, which raises the challenge to find sound (but necessarily incomplete) methods to identify state boundedness. Our results immediately yield one such method, when we consider evolving (graph) databases $\mathcal{G}$ equipped with integrity constraints expressed as a TBox $\mathcal{T}$, assuming a set Σ of relations that are required to be read-only (i.e., acting as master data), and requiring all the remaining

relations to be bounded by $\mathcal{T}$ and Σ (in the sense of Definition 6.1.2). Here $\mathcal{G}$ is a DL interpretation that is required to be a model of $\mathcal{T}$ at each point in time. If the extension of the predicates in Σ is assumed fixed for the whole evolution (or allowed to vary while obeying some predefined bound on its size), then in all legal databases (i.e., those satisfying $\mathcal{T}$) the remaining predicates will have bounded size, which leads to state boundedness and the transfer of results from [BHCDG$^+$13]. We remark that the idea to exploit cardinality constraints (available, e.g., in UML) to identify cases with decidable verification can be found in [CMET14, MC16], but they do not study DLs.

CHAPTER 7

Resilient Logic Programs

As we have already seen, *rule-based languages*—especially those supporting non-monotonic negation—and description logics offer complementary modeling and reasoning capabilities. Indeed, rule-based languages like Datalog and its extansions are tailored to provide powerful *closed-world* reasoning about *known* objects, and features like the default negation are important when modeling dynamic domains, e.g., in reasoning about actions and change. On the other hand, DLs are suitable for *open-world* reasoning, especially for reasoning about *anonymous objects*, i.e., objects whose identity is unknown but whose existence is implied.

Motivated by this contrast, combining rule-based languages and DLs into *Hybrid Knowledge Bases* (HKBs) is a well-established research topic in KR&R [Ros05, EIL$^+$08, MR10, KAH11]. Such hybrid languages can be divided into two classes: the *world-centric* and the *entailment-centric* approaches. The languages in [Ros05, Ros06, BOŠ18] are world-centric because an intended structure (i.e., an answer set) of a HKB is a *single* first-oder structure that is "acceptable" both to the rule and to the DL component of that HKB. Intuitively, in such HKBs the rules base their inferences on a given model of the DL component, rather than accessing the *knowledge* that is entailed. In other words, this means that inferences via rules must only be consistent with the DL component, which is a rather weak way of using the knowledge stored there. The entailment-centric approaches like [EIL$^+$08, MR10] are the other extreme: when ontological reasoning is considered, rules can base their inferences only on the logical consequences of the DL component, which means that rules have very limited access to *individual models* of the DL component.

There are many KR problems where both extremes are inadequate, since solutions must be resilient to a range of possible scenarios. For a simple (synthetic) example, assume we are given a set of nodes and we want to generate a directed graph G such that removing any single node from G will always result in a strongly connected graph. In this example, an ontology can model the possible choices of nodes to be removed. Intuitively, in order

to validate our choice of edges for G, we have to make sure that every possible induced subgraph G', obtained by removing a single node from G, is strongly connected. However, the reachability relation in G' will be different for different choices of G'. This and similar examples reveal the need for a new approach that blurs the lines between the world-centric and the entailment-centric approaches. We thus study HKBs that may process different models of the input ontology in different ways, in the spirit of world-centric approaches, but the intended answer sets, which must be resilient to the different scenarios, are defined via a universal quantification over the models of the ontology, in the spirit of entailment-centric approaches.

Contributions and Relevant Publications. Our contributions can be summarized as follows:

- We introduce *resilient logic programs (RLPs)*, a formalism in which a standard $\mathsf{Datalog}^{\neg}$ program $\mathcal{P}$ is paired with a first-oder theory (or a DL ontology) $\mathcal{T}$ and the predicates are divided into *output*, *response*, and *open-world* predicates. The semantics is defined via a "negotiation" between $\mathcal{P}$ and $\mathcal{T}$: the two components need to agree on an *answer set* I over the output signature, so that no matter how I is extended into a model of $\mathcal{T}$ (by interpreting the open-world predicates), the program $\mathcal{P}$ can give a matching and justified interpretation to the response predicates. Both $\exists\forall\exists$-QBFs and disjunctive Datalog with negation under the stable model semantics ($\mathsf{Datalog}^{\vee,\neg}$) are naturally captured by resilient programs, and in fact, the QBF reduction shows that reasoning in RLPs is Σ_3^P-hard in data complexity, setting them apart form previous hybrid languages. We also illustrate the power of RLPs for configuration problems with incomplete information.
- Inevitably, reasoning in RLPs is undecidable unless restrictions are imposed on how rules are allowed to manipulate anonymous objects. We argue that by applying some natural restrictions, including a rule safety condition reminiscent of the well-known DL-safety [MSS05, Ros05], decidability of reasoning can be achieved. We provide a general complexity upper bound that applies to very expressive FO fragments like the *guarded negation fragment (GFNO)*.
- We further introduce a slightly more restricted fragment of RLPs, in which theories are given as sets of positive disjunctive rules and the use of default negation in front of response predicates is restricted. These restrictions cause a decrease in computational complexity of RLPs and allow us to provide a translation into disjunctive $\mathsf{Datalog}$, opening up a perspective for implementation.
- We then turn to RLPs where the theory is a DL ontology, and show decidability of reasoning under the relaxed rule safety condition based on predicate boundedness, similar to the safety condition introduced in Chapter 6. Finally, for the case where ontologies are written in the well-known DLs $\mathcal{ALCHOIQ}$, $\mathcal{ALCHI}$ and $\mathsf{DL\text{-}Lite}_{\mathcal{F}}$, we provide algorithms and complexity results.

The results from this chapter have been published in:

> [LOŠ20] **Sanja Lukumbuzya**, Magdalena Ortiz, Mantas Šimkus. "Resilient Logic Programs: Answer Set Programs Challenged by Ontologies". In Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, pp. 2917-2924. 2020,

as well as the workshop version of the work above:

> [OPŠ19] Magdalena Ortiz, **Sanja Pavlović**, and Mantas Šimkus. "Answer Set Programs Challenged by Ontologies". In Proceedings of the 32nd International Workshop on Description Logics, DL 2019, CEUR-WS, 2019.

Organization The rest of this chapter is organized as follows. In Section 7.1 we introduce the syntax and the semantics of our formalism and provide some illustrating examples, including a translation of regular $\mathsf{Datalog}^{\vee,\neg}$ programs into RLPs. As RLPs are generally undecidable, in Section 7.2, we identify a large fragment of RLPs for which we can guarantee decidability. Moreover, in Section 7.2.1, we show that by further constraining this fragment, we get RLPs that can be translated into disjunctive Datalog. In Section 7.3, we turn to RLPs whose theory component is given as a DL TBox. In this setting, we define a relaxed safety condition based on bounded predicates and we provide complexity results. Finally, we conclude this chapter with a brief discussion of our results in Section 7.4.

7.1 Resilient Logic Programs

Before we present our formalism, a few remarks are in order.

Preliminaries For convenience, in this chapter, we only consider Herbrand interpretations and we denote them by uppercase letters I and J. Given that we also talk about FO theories, we specify what we mean when we say that an Herbrand interpretation is a model of a FO theory. Let I be an Herbrand interpretation. Then, I induces the FO interpretation $\tilde{I} = (\Delta^{\tilde{I}}, \cdot^{\tilde{I}})$, where:

- $\Delta^{\tilde{I}} = \mathsf{N_I}(I)$, and
- $p^{\tilde{I}} = \{\vec{a} : p(\vec{a}) \in I\}$, for every $p \in \mathsf{N_P}$.

We say that I is a model of a FO theory $\mathcal{T}$, if the induced FO interpretation $\tilde{I}$ is a model of $\mathcal{T}$.

Given an interpretation I and a set Σ of predicates, $I|_\Sigma$ denotes the restriction of I to the predicates in Σ, i.e., $I|_\Sigma = \{p(\vec{a}) \in I : p \in \Sigma\}$. Similarly, for a rule ρ, we denote by $\rho|_\Sigma$ the rule obtained from ρ by deleting all atoms over predicates that are not in Σ.

We next present the syntax and the semantics of our *resilient logic programs*. Syntactically, resilient logic programs consist of an ordinary $\mathsf{Datalog}^\neg$ program equipped with a FO theory and a partition of the signature. Recall that we denote the predicate symbols that occur in a program $\mathcal{P}$ or a theory $\mathcal{T}$ by $\mathsf{sig}(\mathcal{P})$ and $\mathsf{sig}(\mathcal{T})$, respectively.

Definition 7.1.1 (Syntax)**.** *A* resilient logic program (RLP) *is a tuple*

$$\Pi = (\mathcal{P}, \mathcal{T}, \Sigma_{\mathsf{out}}, \Sigma_{\mathsf{owa}}, \Sigma_{\mathsf{re}}),$$

where $\mathcal{P}$ is a program, $\mathcal{T}$ is an FO theory, and the sets Σ_{out}, Σ_{owa}, Σ_{re} are a partition of $\mathsf{sig}(\mathcal{P}) \cup \mathsf{sig}(\mathcal{T})$ with $\Sigma_{\mathsf{re}} \cap \mathsf{sig}(\mathcal{T}) = \emptyset$. We call Σ_{out} the set of output predicates*, Σ_{owa} the set of* open predicates*, and Σ_{re} the set of* response predicates *of Π. The predicates in $\Sigma_{\mathsf{out}} \cup \Sigma_{\mathsf{re}}$ are called* closed predicates *of Π.*

Defining the semantics of RLPs involves the same general steps as defining the semantics of ordinary $\mathsf{Datalog}^\neg$ programs. To this end, we introduce the notion of a *reduct* of a program w.r.t a given interpretation and a set of predicates, that is a generalization of the notion presented in Definition 2.4.8.

Definition 7.1.2. *The* reduct *of a $\mathsf{Datalog}^\neg$ program $\mathcal{P}$ w.r.t. to an interpretation I and a set of predicates $\Sigma \subseteq \mathsf{N_P}$ is the following ground positive program $\mathcal{P}^{I,\Sigma}$:*

$$\begin{aligned}\mathcal{P}^{I,\Sigma} = \{&(\mathit{head}(\rho) \leftarrow \mathit{body}^+(\rho))|_{\mathsf{sig}(\mathcal{P})\setminus\Sigma} : \mathit{body}^+(\rho)|_\Sigma \subseteq I,\\ &\mathit{head}(\rho)|_\Sigma \cap I = \emptyset,\ \mathit{body}^-(\rho) \cap I = \emptyset, \rho \in \mathit{ground}(\mathcal{P}, \mathsf{N_I})\}.\end{aligned}$$

In other words, $\mathcal{P}^{I,\Sigma}$ is obtained from $\mathit{ground}(\mathcal{P}, \mathsf{N_I})$ as follows:

1. *Delete every rule ρ that contains a literal $p(\vec{u})$ such that:*
   - *$p(\vec{u}) \in \mathit{body}^+(\rho)$, $p(\vec{u}) \notin I$, and $p \in \Sigma$,*
   - *$p(\vec{u}) \in \mathit{head}(\rho)$, $p(\vec{u}) \in I$, and $p \in \Sigma$, or*
   - *$p(\vec{u}) \in \mathit{body}^-(\rho)$ and $p(\vec{u}) \in I$.*
2. *In the remaining rules, delete all negated atoms and all atoms $p(\vec{u})$ with $p \in \Sigma$.*

The definition above is inherited from *Clopen KBs* [BOŠ18], which in turn borrow the principle from *r-hybrid KBs* [Ros05]. Intuitively, $\mathcal{P}^{I,\Sigma}$ is the result of partially evaluating $\mathcal{P}$ according to the facts in I, interpreting the predicates in Σ as open-world. Note that in order to compute the regular reduct $\mathcal{P}^I$ we evaluate only the negated atoms in $\mathcal{P}$. In contrast, the generalized reduct requires us to additionally evaluate all atoms $p(\vec{u})$ with $p \in \Sigma$, so that the remaining program contains no predicates from Σ. Observe that if $\Sigma = \emptyset$, $\mathcal{P}^I$ coincides with $\mathcal{P}^{I,\Sigma}$.

We are now ready to define the semantics of RLPs.

Definition 7.1.3 (Semantics). *Let* $\Pi = (\mathcal{P}, \mathcal{T}, \Sigma_{\mathsf{out}}, \Sigma_{\mathsf{owa}}, \Sigma_{\mathsf{re}})$ *be an RLP, and let* I *be an interpretation over* Σ_{out}. *Then* I *is an* answer set *of* Π *if:*

(i) there exists some model J *of* $\mathcal{T}$ *such that* $I = J|_{\Sigma_{\mathsf{out}}}$, *and*

(ii) for each model J *of* $\mathcal{T}$ *with* $I = J|_{\Sigma_{\mathsf{out}}}$, *there is an interpretation* H *such that* $J|_{\Sigma_{\mathsf{out}} \cup \Sigma_{\mathsf{owa}}} = H|_{\Sigma_{\mathsf{out}} \cup \Sigma_{\mathsf{owa}}}$ *and* $H|_{\Sigma_{\mathsf{out}} \cup \Sigma_{\mathsf{re}}}$ *is a minimal model of* $\mathcal{P}^{H, \Sigma_{\mathsf{owa}}}$.

We call H *a* response to J w.r.t. I *and* Π.

Intuitively, an answer set of an RLP $\Pi = (\mathcal{P}, \mathcal{T}, \Sigma_{\mathsf{out}}, \Sigma_{\mathsf{owa}}, \Sigma_{\mathsf{re}})$ is an interpretation I over the output predicates that fulfills the following two conditions:

(i) I is consistent with the theory, i.e., we can extend I into a model of $\mathcal{T}$ by interpreting the open predicates, and

(ii) no matter how we extend I into a model of $\mathcal{T}$, we can always find a matching interpretation for the response predicates that, together with I, will be justified by the program $\mathcal{P}$.

RLPs provide an easy way of modeling and solving problems with an underlying exist-forall-exist structure. More precisely, RLPs are suitable for problems where we have control over some parameters that might, to some extent, influence the environment that we can otherwise not control and is unknown to us a priori, but to which we have to be able to respond adequately. Note that different states of the environment will likely require different responses. In such scenarios, we use the theory to describe the possible states of the environment, and the rules of the program to process a given state. The partitioning of the predicates into three different sets can be intuitively explained as follows. The output predicates are the predicates whose extensions can be controlled and that do not depend on the unknown environment, but rather might influence the set of possible states of the environment one has to consider. The predicates whose extensions we have absolutely no control over are the open predicates. These predicates are used to describe the unknown parts of the environment that we must react to (e.g., things related to user input). Finally, the response predicates are the predicates used in the rules for computing responses to the environment and their extensions are dependent on the specific extensions of the open predicates.

We next illustrate RLPs through a few examples.

Example 7.1.4. *We show how to express the graph problem from the introduction as an RLP. Given nodes* $n_1, \ldots, n_k$, *let* $\Pi = (\mathcal{P}, \mathcal{T}, \{V, E\}, \{\mathsf{in}, \mathsf{out}\}, \{\overline{E}, R\})$, *where*

$$\mathcal{T} = \{\exists x \mathsf{out}(x), \forall x(V(x) \rightarrow \mathsf{in}(x) \vee \mathsf{out}(x)),$$
$$\forall x(V(x) \rightarrow \neg \mathsf{in}(x) \vee \neg \mathsf{out}(x)), \forall x \forall y \mathsf{out}(x) \wedge \mathsf{out}(y) \rightarrow x = y\}$$

$$\begin{aligned}\mathcal{P} = \{&V(n_1),\ \cdots\ V(n_k),\\ &E(x,y) \leftarrow V(x), V(y), \mathit{not}\ \overline{E}(x,y),\\ &\overline{E}(x,y) \leftarrow V(x), V(y), \mathit{not}\ E(x,y),\\ &\mathsf{R}(x,z) \leftarrow R(x,y), R(y,z),\\ &\mathsf{R}(x,y) \leftarrow E(x,y), \mathit{not}\ \mathsf{out}(x), \mathit{not}\ \mathsf{out}(y),\\ &\leftarrow V(x), V(y), x \neq y, \mathit{not}\ \mathsf{out}(x), \mathit{not}\ \mathsf{out}(y), \mathit{not}\ R(x,y).\}\end{aligned}$$

In each answer set of Π*,* E *defines the edge relation of a directed graph* G *such that removing any single node* n_i*,* $i \in \{1, \ldots, k\}$*, from* G *(i.e.,* n_i *is the only node in* out*), results in a graph that is still strongly connected. For example, for* $k = 4$ *we have that*

$$\begin{aligned}I = \{\ &V(n_1), V(n_2), V(n_3), V(n_4),\\ &E(n_1,n_2),\ E(n_2,n_3),\ E(n_3,n_4),\ E(n_4,n_1),\\ &E(n_1,n_3), E(n_3,n_1), E(n_2,n_4), E(n_4,n_2),\}\end{aligned}$$

is an intended answer set as removing any single node from this graph (and all edges relating to this node) yields a strongly connected graph. The set obtained from I *by removing, e.g.,* $E(n_1, n_2)$ *is not an intended answer set. In such a graph, removing* n_3 *results in* n_2 *not being reachable from* n_1*.*

Furthermore, the next example shows that RLPs can elegantly capture $\exists\forall\exists$-quantified Boolean formulas (QBFs).

Example 7.1.5. *Consider the evaluation problem for QBFs of the form*

$$\Phi = \exists X_1, \ldots, X_n \forall Y_1, \ldots, Y_m, \exists Z_1, \ldots, Z_k \varphi,$$

where φ *is in 3-CNF, i.e.,* φ *is of the form* $\bigwedge_{1 \leq i \leq l} \bigvee_{1 \leq j 3} L_{ij}$ *and each* L_{ij} *is either a Boolean variable (= a predicate with arity* 0*) or a negated Boolean variable. We define an RLP* Π *whose answer sets directly correspond to the truth-value assignments for* $X_1, \ldots, X_n$ *for which* Φ *evaluates to true.*

Note that Π *reflects the quantifier alternation in* Φ*: we want to output an assignment for the* X_i *(i.e., an interpretation over* T_X *and* F_X*) such that for every assignment for the open-world* Y_i *we can respond with an assignment for the* Z_i *that satisfies the constraints, i.e., the clauses in* Φ*. We let* $\Pi = (\mathcal{P}, \mathcal{T}, \{V_X, V_Y, V_Z, T_X, F_X\}, \{T_Y, F_Y\}, \{T_Z, F_Z\})$*,*

$$\begin{aligned}\mathcal{T} = \{&\forall x(V_Y(x) \rightarrow T_Y(x) \vee F_Y(x)),\\ &\forall x(V_Y(x) \wedge T_Y(x) \wedge F_Y(x) \rightarrow \bot)\}\end{aligned}$$

$$\begin{aligned}\mathcal{P} = \{&V_\alpha(c^\alpha_1), \qquad \ldots \qquad V_\alpha(c^\alpha_{n_\alpha}),\\ &T_X(x) \leftarrow V_X(x), \mathit{not}\ F_X(x)\\ &F_X(x) \leftarrow V_X(x), \mathit{not}\ T_X(x)\\ &T_Z(x) \leftarrow V_Z(x), \mathit{not}\ F_Z(x)\\ &F_Z(x) \leftarrow V_Z(x), \mathit{not}\ T_Z(x)\\ &\leftarrow \sigma(L_{i,1}), \sigma(L_{i,2}), \sigma(L_{i,3}),\ \textit{for each clause } C_i \textit{ in } \varphi\}\end{aligned}$$

where, given a literal l, $\sigma(l)$ is defined as:

$$\sigma(l) = \begin{cases} T_\alpha(c_i^\alpha) & \text{if } l = \neg\alpha_i, i = 1, \ldots, n_\alpha \\ F_\alpha(c_i^\alpha) & \text{if } l = \alpha_i, i = 1, \ldots, n_\alpha \end{cases}$$

where $\alpha \in \{X, Y, Z\}$, $n_X = n, n_Y = m$, and $n_Z = k$.

Intuitively, each Boolean variable X_i (resp. Y_i, Z_i) in Φ is represented by a constant c_i^X (resp. c_i^Y, c_i^Z). The predicate symbols V_X, V_Y, and V_Z are used to partition these constants into three sets that correspond to the partitioning of variables into different quantifier blocks in Φ. For example, $V_X(c_i^X)$ means that variable X_i is in the first quantifier block. We use predicate symbols T_X, F_Y (resp., $T_Y, F_Y/T_Z, F_Z$) to indicate the truth value assigned to X (resp., Y/Z) variables. Namely, the expression $T_X(c_i^X)$ is understood as 'variable X_i is assigned true' and $F_X(c_i^X)$ as 'X_i is assigned false'. The meaning of other predicates is defined analogously. The theory component $\mathcal{T}$ makes sure that we only consider interpretations that can be seen as well-defined truth-value assignments to variables $Y_1, \ldots, Y_m$. In particular, this means that in any model I of $\mathcal{T}$, exactly one of $T_Y(c_i^Y) \in I$ and $F_Y(c_i^Y) \in I$ holds, for each $i = 1, \ldots, m$. Rules r_X and r_Y serve a similar purpose. We also add a rule $\leftarrow \sigma(L_{i,1}), \sigma(L_{i,2}), \sigma(L_{i,3})$ for each clause $C_i = L_{i,1} \vee L_{i,2} \vee L_{i,3}$ in φ. Intuitively, this ensures that in every model of $\mathcal{P}$, at least one literal per clause must be satisfied. Finally, we choose our output predicates to be V_Y, T_X, F_X. Notice that we can interpret candidate answers sets of Π as truth-value assignments to $X_1, \ldots, X_n$, models of $\mathcal{T}$ as truth-value assignments to $Y_1, \ldots, Y_n$, and interpretations over the response predicates as truth-value assignments to $Z_1, \ldots, Z_k$. Additionally, $\mathcal{P}$ is defined in a way that given a candidate answer set I, a model of the theory J and an interpretation H over the response predicates, $H \cup I$ is a minimal model of $\mathcal{P}^{H \cup I \cup J,\ \Sigma_{\text{owa}}}$ if $H \cup I \cup J$ defines an assignment to the variables of φ that satisfies φ. Clearly, the answer sets of Π are in direct correspondence with the truth-value assignments to $X_1, \ldots, X_n$ for which Φ evaluates to true.

As mentioned above, RLPs can be used to solve problems in which one needs to come up with robust settings that allow for a successful processing of all events that may come in many possible configurations. In such scenarios, the role of the theory is to describe all possible configurations of events and the rules of the program are used to process the given configuration. Our semantics then ensures that the answer sets of such RLPs coincide with the sought-after settings. We next present an example that illustrates this.

Example 7.1.6. *Assume a company has to process a fixed amount of customer orders per day. The company does not know what the exact configuration of these orders will be, but it knows that each of them consists of up to 5 tasks and each task requires one service offered by the company. The company has a task of selecting which services to offer so that no matter what the actual configuration of the orders is, the tasks can be scheduled to employees in a way that each task will be completed by the end of the day.*

This problem is solved by an RLP in which the offered services are captured by the output predicates, models of the theory correspond to possible configurations of orders, and the

models of the program define viable schedules of tasks to employees. The answer sets of such an RLP then correspond to sets of services that, if offered, guarantee that for every configuration of orders there exists a schedule in which each task is completed by the end of the workday.

We define a program $\mathcal{P}_1$ *consisting of the rules that model the timeline of the workday:*

$$\begin{aligned}
&\mathit{Next(i,i+1)},\ \textit{for } 0 \leq i < t_{max},\\
&\mathit{Time}(y) \leftarrow \mathit{Next}(x, y),\\
&\mathit{Time}(x) \leftarrow \mathit{Next}(x, y),\\
&\mathit{ltHour}(x_0, x_n) \leftarrow \mathit{Next}(x_0, x_1), \ldots, \mathit{Next}(x_{n-1}, x_n),\ 0 \leq n < 60
\end{aligned}$$

Assume that the employees work eight hours per day and the granularity of Next *is one minute. We set* $t_{max} = 480$.

As facts, we store the employees, the services, as well as which employee can provide which service. For simplicity assume that each service takes the same amount of time to be completed, e.g., 60 minutes, and that the company needs to process two orders per day. We also encode this information using facts. Consider, for demonstration purposes, the following set of facts:

$$\begin{aligned}
\mathcal{P}_2 = \{&\mathit{Service}(s_1), \mathit{Service}(s_2), \mathit{Service}(s_3),\\
&\mathit{Employee}(e_1), \mathit{Employee}(e_2), \mathit{Order}(o_1), \mathit{Order}(o_2),\\
&\mathit{Provides}(e_1, s_1), \mathit{Provides}(e_1, s_2),\\
&\mathit{Provides}(e_2, s_1), \mathit{Provides}(e_2, s_2), \mathit{Provides}(e_2, s_3)\}
\end{aligned}$$

We further introduce two binary predicates hasTask *and* Req *for specifying which orders have which tasks associated to them and which tasks require which offered services, respectively. Assume we have an FO theory* $\mathcal{T}$ *expressing the following information: (i) each order has at least one and at most five tasks associated to it, (ii) each task is associated to exactly one order and (iii) each task requires exactly one offered service. We show later that such a theory can be elegantly expressed using description logics.*

The rules that select services are defined as follows:

$$\begin{aligned}
\mathcal{P}_3 = \{&\mathit{OfferedService}(x) \leftarrow \mathit{Service}(x), \mathit{not}\ \overline{\mathit{OfferedService}}(x)\\
&\overline{\mathit{OfferedService}}(x) \leftarrow \mathit{Service}(x), \mathit{not}\ \mathit{OfferedService}(x)\}
\end{aligned}$$

Next, the set $\mathcal{P}_4$ *of rules generates a viable schedule (*Sched*) consisting of tuples* (x, y, z),

assigning task y to employee x to be performed starting at time point z.

$$\begin{aligned}
\mathcal{P}_4 = \{ \mathit{Sched}(x,y,z) &\leftarrow \mathit{Task}(y), \mathit{Time}(z), \mathit{Req}(y,u), \mathit{Provides}(x,u), \mathit{not}\ \mathit{Illegal}(x,y,z) \\
\mathit{Illegal}(x,y,z) &\leftarrow \mathit{Task}(y), \mathit{Time}(z), \mathit{ltHour}(z, t_{max}), \mathit{Employee}(x) \\
\mathit{Illegal}(x,y,z) &\leftarrow \mathit{Sched}(x',y,z'), Time(z), \mathit{Employee}(x), x \neq x' \\
\mathit{Illegal}(x,y,z) &\leftarrow \mathit{Sched}(x',y,z'), Time(z), \mathit{Employee}(x), z \neq z' \\
\mathit{Illegal}(x,y,z) &\leftarrow \mathit{Task}(y), \mathit{Sched}(x,y',z'), \mathit{ltHour}(z',z), z \neq z' \\
\mathit{Illegal}(x,y,z) &\leftarrow \mathit{Task}(y), \mathit{Sched}(x,y',z'), \mathit{Time}(z), z = z', y \neq y' \\
\mathit{OKTask}(y) &\leftarrow \mathit{Sched}(x,y,z) \\
&\leftarrow \mathit{not}\ \mathit{OKTask}(y), Task(y)\}
\end{aligned}$$

Let $\Sigma_{\mathsf{out}} = \{\mathit{Order}, \mathit{OfferedService}\}$, $\Sigma_{\mathsf{owa}} = \mathsf{sig}(\mathcal{T}) \setminus \Sigma_{\mathsf{out}}$, and $\Sigma_{\mathsf{re}} = \mathsf{sig}(\mathcal{P}) \setminus (\Sigma_{\mathsf{out}} \cup \Sigma_{\mathsf{owa}})$. The answer sets of the RLP $(\mathcal{P}, \mathcal{T}, \Sigma_{\mathsf{out}}, \Sigma_{\mathsf{owa}}, \Sigma_{\mathsf{re}})$, where $\mathcal{P} = \mathcal{P}_1 \cup \mathcal{P}_2 \cup \mathcal{P}_3 \cup \mathcal{P}_4$, represent sets of services that can be offered and completed within the given time constraints regardless of the type of received orders. One example of such an answer set is $I = \{\mathit{Order}(o_1), \mathit{Order}(o_2), \mathit{OfferedService}(s_1)\mathit{OfferedService}(s_2)\}$. It can be verified that whatever the configurations o_1 and o_2 might be, we can always find a schedule in which all tasks are completed on time.

Note that, unlike traditional $\mathsf{Datalog}^{\neg}$, RLPs can have comparable answer sets. For example, the set $\{\mathit{OfferedService}(s_1)\mathit{Order}(o_1), \mathit{Order}(o_2)\} \subset I$ is also an answer set of Π.

Let $J = \{\mathit{OfferedService}(s_3), \mathit{Order}(o_1), \mathit{Order}(o_2)\}$ and consider a model of $\mathcal{T}$ in which each order consist of five tasks associated with the service s_3. Since only employee e_2 can perform s_3, she needs to perform this service ten times. However, this takes more than the 480 minutes and so no valid schedule can be found. This means that J is not an answer set of Π.

7.1.1 Encoding $\mathsf{Datalog}^{\vee,\neg}$ into RLPs

There is a strong connection between RLPs and $\mathsf{Datalog}^{\vee,\neg}$, i.e., *disjunctive logic programs with negation under the stable model semantics* [EGM97]. Recall that $\mathsf{Datalog}^{\vee,\neg}$ programs extend $\mathsf{Datalog}^{\neg}$ rules by allowing disjunctions of atoms in rule heads. Namely, $\mathsf{Datalog}^{\vee,\neg}$ rules are of the form

$$h_1 \vee \ldots \vee h_l \leftarrow b_1, \ldots, b_n, \mathit{not}\ b_{n+1}, \ldots, \mathit{not}\ b_m,$$

where $l > 1, n, m \geq 0$, $h_1, \ldots, h_l, b_1, \ldots, b_m$ are atoms and every variable occurring in $h_1, \ldots, h_l, b_{n+1}, \ldots, b_m$ must occur in some $b_1, \ldots, b_n$. We next show that $\mathsf{Datalog}^{\vee,\neg}$ programs are fully captured by RLPs.

Theorem 7.1.7. *Every $\mathsf{Datalog}^{\vee,\neg}$ program $\mathcal{P}$ can be translated in polynomial time into a resilient logic program $\Pi = (\mathcal{P}', \{\top\}, \Sigma_{\mathsf{out}}, \Sigma_{\mathsf{owa}}, \Sigma_{\mathsf{re}})$ whose answer sets coincide with those of $\mathcal{P}$.*

Proof. The core idea of the translation is to use two copies of the signature of $\mathcal{P}$, both encoding possible interpretations over the signature and the constants of $\mathcal{P}$. As the theory component plays no role in this translation we simply use $\top$. Our strategy is the following. We first define rules that generate a candidate answer set I of $\mathcal{P}$ and use constraints to ensure that I satisfies all the rules of $\mathcal{P}$. For this we use the signature of $\mathcal{P}$ – our set of output predicates, and one of its copies – the response predicates. The other copy constitutes the set of open predicates and is used to verify that I is indeed an answer set of $\mathcal{P}$. To do this, we need to make sure that there is no interpretation $J \subsetneq I$ that is a model of $\mathcal{P}^I$. In other words, we need to check that *for all possible interpretations* J, either (i) $J = I$, (ii) there is an atom h such that $h \in J$ and $h \notin I$, i.e., $J \not\subset I$, or (iii) J violates some rule in $\mathcal{P}^I$. We define the rules, using some additional response predicates, to check for a given J if one of these conditions holds. Since we have a universal quantification over interpretations, we use open predicates to encode J. Our semantics then ensures that if I is an answer set of Π then for any possible interpretation J at least one of (i)-(iii) holds. Hence, the answer sets of $\mathcal{P}$ and Π coincide. We next present our translation below.

Assume that $p_1, \ldots, p_n$ are the predicate symbols in $\mathcal{P}$. For each p_i, we take two fresh predicate symbols $\overline{p}_i$ and p'_i of the same arity as p_i.

The set of rules $\mathcal{P}'$ is now defined as follows. First, we add all the facts of $\mathcal{P}$ together with the rule $\mathsf{Adom}(x_i) \leftarrow p(x_1, \ldots, x_k)$ for each k-ary $p \in \Sigma$ with $k > 0$, and each $1 \leq i \leq k$. For any $k > 0$ and tuple $(t_1, \ldots, t_k)$ of terms, in rule bodies we may write $\mathsf{Adom}^k(t_1, \ldots, t_k)$ instead of $\mathsf{Adom}(t_1), \ldots, \mathsf{Adom}(t_k)$. For each k-ary predicate $p \in \Sigma$, we take a fresh k-ary predicate $\overline{p}$ and we add:

$$p(\vec{x}) \leftarrow \mathsf{Adom}^k(\vec{x}), \mathit{not}\ \overline{p}(\vec{x})$$
$$\overline{p}(\vec{x}) \leftarrow \mathsf{Adom}^k(\vec{x}), \mathit{not}\ p(\vec{x}),$$

where $\vec{x}$ is a tuple of k variables. For every rule

$$h_1(\vec{t_1}) \vee \ldots \vee h_k(\vec{t_k}) \leftarrow b_1(\vec{u}_1), \ldots, b_n(\vec{u}_n), \mathit{not}\ b_{n+1}(\vec{u}_{n+1}), \ldots, \mathit{not}\ b_m(\vec{u}_m)$$

in $\mathcal{P}$, we add the following constraint to $\mathcal{P}'$:

$$\leftarrow \overline{h}_1(\vec{t_1}), \ldots, \overline{h}_k(\vec{t_k}), b_1(\vec{u}_1), \ldots, b_n(\vec{u}_n), \overline{b}_{n+1}(\vec{u}_{n+1}), \ldots, \overline{b}_m(\vec{u}_m).$$

We set $\Sigma_{\mathsf{out}} = \{p_1, \ldots, p_n\}$. These rules guess a structure I that corresponds exactly to a classical model of $\mathcal{P}$, i.e. a model of $\mathcal{P}$ seen as a theory in classical logic, with "*not*" interpreted as the standard negation operator $\neg$. To guarantee that such I is an answer set to $\mathcal{P}$, it remains to make sure that there is no $J \subsetneq I$ such that J is a model of $\mathcal{P}^I$. In other words, we have to make sure that for all interpretations J, we have that (i) $J = I$, (ii) there is an atom h such that $h \in J$ and $h \notin I$, or (iii) there is a rule in $\mathcal{P}^I$ that is violated in J. We use the theory $\mathcal{T} = \{\top\}$ and the open predicates $\Sigma_{\mathsf{owa}} = \{p'_1, \ldots, p'_n\}$ to set up a universal quantification over the candidate interpretations J. We now define

the rules to check if one of (i)-(iii) holds, which is indicated by justifying a 0-ary predicate ok.

To deal with (i), we add $\mathsf{ok} \leftarrow \mathit{not}\ \mathit{neq}$ and the following rules for all predicates p of $\mathcal{P}$:

$$\mathsf{neq} \leftarrow \overline{p}(\vec{x}), p'(\vec{x}) \qquad \mathsf{neq} \leftarrow p(\vec{x}), \mathit{not}\ p'(\vec{x}),$$

with $\vec{x}$ an arbitrary k-tuple of variables and k the arity of p.

To deal with (ii), we add for every predicate p of $\mathcal{P}$, the rule $\mathsf{ok} \leftarrow \overline{p}(\vec{x}), p'(\vec{x})$, where $\vec{x}$ is an arbitrary k-tuple of variables with k the arity of p.

Finally, to deal with (iii), for every rule

$$h_1(\vec{t_1}) \vee \ldots \vee h_k(\vec{t_k}) \leftarrow b_1(\vec{u}_1), \ldots, b_n(\vec{u}_n), \mathit{not}\ b_{n+1}(\vec{u}_{n+1}), \ldots, \mathit{not}\ b_m(\vec{u}_m)$$

in $\mathcal{P}$, we add the following rule to $\mathcal{P}'$:

$$\mathsf{ok} \leftarrow \mathit{not}\ h'_1(\vec{t_1}), \ldots, \mathit{not}\ h'_k(\vec{t_k}), b'_1(\vec{u}_1), \ldots, b'_n(\vec{u}_n), \overline{b}_{n+1}(\vec{u}_{n+1}), \ldots, \overline{b}_m(\vec{u}_m).$$

Intuitively, ok will be justified in all possible models of $\mathcal{T}$ iff (i)-(iii) hold, i.e., any interpretation I over Σ_{out} does not have an interpretation $J \subsetneq I$ such that J is a model of P^I. To finish the translation, we add to $\mathcal{P}'$ the constraint

$$\leftarrow \mathit{not}\ \mathsf{ok}.$$

One can easily verify that the answer sets of $\mathcal{P}$ and Π coincide. □

7.2 Decidable RLPs

The main reasoning task for RLPs is deciding the *answer set existence*, i.e., given an RLP Π, decide whether there exist an answer set I of Π. We note that other common reasoning problems like *skeptical* (resp., *brave*) *entailment* can be easily reduced to checking non-existence (resp., existence) of an answer set. In particular, a ground atom $p(\vec{c})$ is true in all answer sets of Π (i.e., $p(\vec{c})$ is a skeptical consequence) iff the result of adding $\leftarrow p(\vec{c})$ to (the program component of) Π does not have an answer set. Similarly, a ground atom $p(\vec{c})$ is true in some answer set of Π (i.e., $p(\vec{c})$ is a brave consequence) iff Π augmented with $\leftarrow \mathit{not}\ p(\vec{c})$ does have an answer set. With this in mind, the rest of this chapter is focused on the problem of deciding answer set existence.

Unsurprisingly, this task is undecidable for RLPs in their general form. This is not hard to show adapting analogous results for some well-known hybrid languages [LR98, Ros07a]. Thus, the main goal of this section is to identify a class of RLPs for which decidability can be regained.

The first step is to introduce a *safety* condition for RLPs that ensures that variables in the program range only over a finite number of constants. To this end, we use the notion of safety presented in [BOŠ18] which was inspired by the well-known *DL-safety* [MSS05, Ros05]:

Definition 7.2.1. *An RLP* $\Pi = (\mathcal{P}, \mathcal{T}, \Sigma_{\mathsf{out}}, \Sigma_{\mathsf{owa}}, \Sigma_{\mathsf{re}})$ *is* safe *if for each rule* $\rho \in \mathcal{P}$*, each variable in* ρ *occurs in some* $p(\vec{u}) \in body^{+}(\rho)$*, where* $p \in \Sigma_{\mathsf{out}} \cup \Sigma_{\mathsf{re}}$*.*

Intuitively, for RLPs that fulfill this condition, variables in the program $\mathcal{P}$ range only over the active domain of the program, i.e., the set of constants explicitly mentioned in $\mathcal{P}$. This is exploited to devise a terminating algorithm for answer set existence of RLPs.

An obvious further requirement for decidability is that the theory component of RLPs must belong to a fragment of FO in which satisfiability is decidable. However, we need something stronger: the theory must be in a logic for which the problem of *satisfiability in the presence of closed predicates* is decidable, i.e., given a theory $\mathcal{T}$, a set of predicates Σ regarded as closed, and an interpretation I, we can decide whether there is a model J of $\mathcal{T}$ s.t. $J|_{\Sigma} = I$.

Theorem 7.2.2. *The problem of answer set existence is decidable for safe RLPs whose theory is in a logic for which satisfiability under closed predicates is decidable.*

Proof. Algorithm 7.1 decides the answer set existence for such RLPs. In a nutshell, the algorithm takes an RLP $\Pi = (\mathcal{P}, \mathcal{T}, \Sigma_{\mathsf{out}}, \Sigma_{\mathsf{owa}}, \Sigma_{\mathsf{re}})$, guesses an interpretation I and subsequently uses two oracle calls to check whether I is an answer set of Π. The first call checks whether there is a model J of $\mathcal{T}$ s.t. $J|_{\Sigma_{\mathsf{out}}} = I$. The second call tries to verify whether each such model of $\mathcal{T}$ has a response w.r.t. I and Π by attempting to guess a model J' of $\mathcal{T}$ for which this does not hold. In order to check that J' has no response, another oracle call is made which tries to guess a response. If this fails, there is a counter example to I being an answer set of Π. Note that, due to our safety condition, it suffices to make the guesses only over $adom(\mathcal{P})$. Further, the subroutine isConsistent$(\mathcal{T}, I, \Sigma)$ checks whether there exists a model J of $\mathcal{T}$ s.t. $J|_{\Sigma} = I$. For RLPs that fulfill the conditions from Theorem 7.2.2 this problem is decidable and the subroutine terminates, yielding a decision procedure. □

A naive analysis of Algorithm 7.1 yields the following upper bound in terms of computational complexity:

Theorem 7.2.3. *Let* $\mathcal{L}$ *be a logic in which satisfiability under closed predicates is in some complexity class* C *that contains NP. Deciding answer set existence of safe RLPs with theories in* $\mathcal{L}$ *is in* NExpTime $^{C^{NP}}$*.*

One of the most expressive decidable fragments of FO is the so-called *guarded negation fragment (GNFO)* [BCS15], which extends the guarded fragment of FO and also captures most description logics. As in GNFO satisfiability under closed predicates is decidable in 2ExpTime [BBtCP16, BOŠ18], by Theorem 7.2.3, answer set existence for safe RLPs with GNFO theories is decidable and belongs to the class NExpTime $^{2\text{ExpTime}^{NP}}$ = NExpTime $^{2\text{ExpTime}}$.

Algorithm 7.1: Answer set existence for safe RLPs.

Algorithm hasAnswerSet(Π)
- **Input** : RLP $\Pi = (\mathcal{P}, \mathcal{T}, \Sigma_{\mathsf{out}}, \Sigma_{\mathsf{owa}}, \Sigma_{\mathsf{re}})$
- **Output** : *true* iff Π has an answer
- Guess an interpretation I over Σ_{out} and $adom(\mathcal{P})$
- **return** isConsistent$(\mathcal{T}, \Sigma_{\mathsf{out}}, I)$ and *not* hasCE$(I, \Pi, adom(\mathcal{P}), \Sigma_{\mathsf{out}})$

Subroutine hasCE($I, \Pi, \Delta, \mathcal{B}$)
- **Input** : Interpretation I, RLP $\Pi = (\mathcal{P}, \mathcal{T}, \Sigma_{\mathsf{out}}, \Sigma_{\mathsf{owa}}, \Sigma_{\mathsf{re}})$, $\Delta \subseteq \mathsf{N_I}$, and $\mathcal{B} \subseteq \mathsf{sig}(\mathcal{T})$
- **Output** : *true* iff there is a counter example to I being an answer set of Π
- Guess an interpretation J over $\mathsf{sig}(\mathcal{P}) \cap \mathsf{sig}(\mathcal{T}) \cup \Sigma_{\mathsf{out}}$ and the constants from Δ s.t. $J|_{\Sigma_{\mathsf{out}}} = I|_{\Sigma_{\mathsf{out}}}$
- $J' \leftarrow J \cup \{\neg p(\vec{c}) : \vec{c} \in \Delta^{art(p)}, p \in \mathsf{sig}(\mathcal{P}) \cap (\mathsf{sig}(\mathcal{T}) \setminus \mathcal{B}),$ and $p(\vec{c}) \notin J\}$
- **return** isConsistent$(\mathcal{T}, \mathcal{B}, J')$ and *not* hasResp(J, Π)

Subroutine hasResp(J, Π)
- **Input** : Interpretation J, RLP $\Pi = (\mathcal{P}, \mathcal{T}, \Sigma_{\mathsf{out}}, \Sigma_{\mathsf{owa}}, \Sigma_{\mathsf{re}})$
- **Output** : *true* iff there exists a response to J w.r.t. $J|_{\Sigma_{\mathsf{out}}}$ and Π
- Guess an interpretation H over $\mathsf{sig}(\mathcal{P})$ and the constants from J and $adom(\mathcal{P})$ s.t. $H|_{\Sigma_{\mathsf{out}} \cup \Sigma_{\mathsf{owa}}} = J$
- **return** $H|_{\Sigma_{\mathsf{out}} \cup \Sigma_{\mathsf{re}}}$ *is a min. model of* $\mathcal{P}^{H, \Sigma_{\mathsf{owa}}}$

7.2.1 Translation into Disjunctive Datalog

We now identify a fragment of RLPs that can be translated into disjunctive Datalog. First, we require that all RLPs in this fragment are safe, as described in the previous section.

To this end, we disallow default negation in front of response predicates in rules other than constraints. This restriction has as an effect that, given a candidate answer set I and a model of the theory J that agrees with I on the output predicates, deciding whether there is a response to J can be done deterministically, which reduces the computational complexity of reasoning with such RLPs. We note that many problems, like Example 7.1.4 and Example 7.1.6, can be captured in this fragment.

Furthermore, we restrict the theory component of RLPs in this fragment essentially to a set of positive disjunctive rules which allows us to easily encode formulas of the theory as rules of the program component. More formally, we consider FO formulas of the form $\forall x_1, \ldots, x_l\, \varphi$, where φ is a disjunction of literals using only constants and variables $x_1, \ldots, x_l$ in which every variable occurring in a positive literal involving an output predicate also occurs in a negative literal involving an output predicate. This fragment of FO is particularly important for DL research as the formulas in it can be

seen as positive disjunctive rules and many standard DLs are rewritable into positive disjunctive Datalog (see, e.g., [HMS07, BtCLW14]).

We next show that the considered fragment of RLPs is captured by disjunctive logic programs with negation under the stable model semantics.

Theorem 7.2.4. *Let* $\Pi = (\mathcal{P}, \mathcal{T}, \Sigma_{\mathsf{out}}, \Sigma_{\mathsf{owa}}, \Sigma_{\mathsf{re}})$ *be an RLP for which:*

- *for every rule* $\rho \in \mathcal{P}$*, if* ρ *is not a constraint, then default negation does not occur in front of response predicates, and*
- *each formula in* $\mathcal{T}$ *is of the form* $\forall x_1, \ldots, x_l\, \varphi$*, where* φ *is a disjunction of literals using only constants and variables* $x_1, \ldots, x_l$ *and in which every variable occurring in a positive literal over an output predicate also occurs in a negative literal over a an output predicate.*

We can obtain a $\mathsf{Datalog}^{\vee,\neg}$ *program* $\mathcal{P}^{\Pi}_{solve}$ *whose answer sets restricted to* Σ_{out} *coincide with the answer sets of* Π*.*

The rest of this section serves as the proof of the theorem above. To this end, we employ the technique from [EP03] that transforms two non-disjunctive programs $\mathcal{P}_1$ and $\mathcal{P}_2$ into a single disjunctive program $\mathcal{P}_3$ whose answer sets correspond to the answer sets S of $\mathcal{P}_1$ for which $\mathcal{P}_2 \cup S$ does not have an answer set.

As a first step, we can obtain from an RLP $\Pi = (\mathcal{P}, \mathcal{T}, \Sigma_{\mathsf{out}}, \Sigma_{\mathsf{owa}}, \Sigma_{\mathsf{re}})$ two programs $\mathcal{P}^{\Pi}_{guess}$ and $\mathcal{P}^{\Pi}_{check}$ that do the following:

- $\mathcal{P}^{\Pi}_{guess}$ guesses a structure over the output predicates and makes sure that it can be extended into a model of $\mathcal{T}$ by interpreting the open predicates. Answer sets of $\mathcal{P}^{\Pi}_{guess}$ consist of such structures, extended into models of $\mathcal{T}$.
- $\mathcal{P}^{\Pi}_{check}$ checks, given an answer set S of $\mathcal{P}^{\Pi}_{guess}$, whether there is a model of $\mathcal{T}$ that agrees with S on the output predicates and has no response. Such models constitute the answer sets of $\mathcal{P}^{\Pi}_{check} \cup S$.

The second step is to use the above-mentioned technique to obtain a disjunctive program $\mathcal{P}^{\Pi}_{solve}$ whose answer sets correspond to the answer sets S of $\mathcal{P}^{\Pi}_{guess}$ for which $\mathcal{P}^{\Pi}_{check} \cup S$ does not have an answer set. This results in the program $\mathcal{P}^{\Pi}_{solve}$ having answer sets that correspond to structures S over output predicates (extended into models of $\mathcal{T}$ in possibly multiple different ways) for which there is no model of $\mathcal{T}$ that agrees with S on the output predicates and has no response. Thus, we have that the answer sets of $\mathcal{P}^{\Pi}_{solve}$ (restricted to the output predicates) and the answer sets of Π coincide.

We now make the intuitions above more precise. Consider an RLP $\Pi = (\mathcal{P}, \mathcal{T}, \Sigma_{\mathsf{out}}, \Sigma_{\mathsf{owa}}, \Sigma_{\mathsf{re}})$. The construction of $\mathcal{P}^{\Pi}_{guess}$ is rather straightforward. We first add the fact

$\mathsf{Adom}(c)$, for every $c \in adom(\mathcal{P})$ and $\mathsf{Const}(c)$ for every $c \in \mathsf{N_I}$ occurring in $\mathcal{T}$ or $adom(\mathcal{P})$. Next, we guess the structure over the output predicates by adding the following for every $p \in \Sigma_{\mathsf{out}}$ of arity $k \geq 0$:

$$p(\vec{x}) \leftarrow \mathsf{Adom}^k(\vec{x}), \mathit{not}\ \overline{p}(\vec{x}),$$
$$\overline{p}(\vec{x}) \leftarrow \mathsf{Adom}^k(\vec{x}), \mathit{not}\ p(\vec{x}),$$

where $\vec{x}$ is k-tuple of variables from $\mathsf{N_V}$.

We then check whether there is a model of $\mathcal{T}$ in which the extensions of output predicates correspond exactly to the ones we guessed using the previous rule. We do this by adding, for each $q \in \Sigma_{\mathsf{owa}}$ of arity $k \geq 0$, the rule

$$q(\vec{x}) \leftarrow \mathsf{Const}^k(\vec{x}), \mathit{not}\ \overline{q}(\vec{x}),$$
$$\overline{q}(\vec{x}) \leftarrow \mathsf{Const}^k(\vec{x}), \mathit{not}\ q(\vec{x}),$$

where $\vec{x}$ is k-tuple of variables from $\mathsf{N_V}$.

Finally, for every formula $\forall x_1, \ldots, x_l(a_1 \vee \ldots \vee a_j \vee \neg a_{j+1} \vee \ldots \vee \neg a_h) \in \mathcal{T}$, we add the constraint:

$$\leftarrow a_{j+1}, \ldots, a_h, \mathit{not}\ a_1, \ldots, \mathit{not}\ a_j, \mathsf{Const}^l(x_1, \ldots, x_l).$$

This completes our construction of the program $\mathcal{P}^{\Pi}_{guess}$.

Let us denote by $ground(\mathcal{T}, \Delta)$ the grounding of the theory $\mathcal{T}$ with respect to the constants in $\Delta \subseteq \mathsf{N_I}$, i.e., the quantifier-free theory obtained from $\mathcal{T}$ by taking each formula $\varphi \in \mathcal{T}$, uniformly replacing all variables in it by all possible elements from $\Delta \subseteq \mathsf{N_I}$ and then removing all universal quantifiers. Once $\mathcal{P}^{\Pi}_{guess}$ guesses the output structure I, it checks whether there is a model J of $\mathcal{T}$ s.t. $J|_{\Sigma_{\mathsf{out}}} = I$. As explained above, this check is implemented by further guessing the extensions of the open predicates over the constants from $adom(\mathcal{P})$ and $\mathcal{T}$, and transforming formulas in $\mathcal{T}$ into constraints in the program. If there is an answer set of $\mathcal{P}^{\Pi}_{check}$, these constraints ensure that our guesses for the output and the open predicates define a model of $ground(\mathcal{T}, \Delta)$, where Δ is the set of constants from $adom(\mathcal{P})$ and $\mathcal{T}$.

The correctness of this approach relies on the following lemma.

Lemma 7.2.5. *Let $\Pi = (\mathcal{P}, \mathcal{T}, \Sigma_{\mathsf{out}}, \Sigma_{\mathsf{owa}}, \Sigma_{\mathsf{re}})$ be an RLP in the considered class and let I be an interpretation over $\Sigma_{\mathsf{out}} \cup \Sigma_{\mathsf{owa}}$ using only the constants from $\Delta \subseteq \mathsf{N_I}$ s.t. all constants occurring in $\mathcal{T}$ are also in Δ. If I is a model of $ground(\mathcal{T}, \Delta)$, then there is a model J of $ground(\mathcal{T}, \mathsf{N_I})$ s.t. $J|_{\Sigma_{\mathsf{out}}} = I|_{\Sigma_{\mathsf{out}}}$ and $\{p(\vec{c}) : p(\vec{c}) \in J, \vec{c} \in \Delta^{art(p)}\} = I$.*

Proof. Assume I is a model of $ground(\mathcal{T}, \Delta)$ and consider the following theory:

$$\begin{aligned}\mathcal{T}' = {} & ground(\mathcal{T}, \mathsf{N_I}) \cup I \cup \\ & \{\neg p(\vec{c}) : p(\vec{c}) \notin I, \vec{c} \in \Delta^{art(p)}\} \cup \\ & \{\neg p(\vec{c}) : p \in \Sigma_{\mathsf{out}}, \vec{c} \in \mathsf{N_I}^{art(p)} \setminus \Delta^{art(p)}\}\end{aligned}$$

It is easy to verify that there is a model J of $ground(\mathcal{T}, \mathsf{N_I})$ s.t. $\{p(\vec{c}) : p(\vec{c}) \in J, \vec{c} \in \Delta^{art(p)}\} = I$ and $J|_{\Sigma_{\mathsf{out}}} = I|_{\Sigma_{\mathsf{out}}}$ if and only if $\mathcal{T}'$ is consistent.

We now proceed as follows. First, we show that the following theory is consistent:

$$\mathcal{T}'' = ground(\mathcal{T}, \mathsf{N_I}) \cup I \ \cup \{\neg p(\vec{c}) : p(\vec{c}) \notin I, \vec{c} \in \Delta^{art(p)}\}$$

Suppose towards a contradiction that this is not the case. Then, as $\mathcal{T}''$ is a set of clauses, we use resolution to obtain a proof of $\bot$. Denote the set of the initial clauses in this proof by S. First note that at least one of the clauses in S must contain an atom that uses a constant from $\mathsf{N_I} \setminus \Delta$, otherwise this proof would be valid also in $ground(\mathcal{T}, \Delta) \cup I \ \cup \{\neg p(\vec{c}) : p(\vec{c}) \notin I, \vec{c} \in \Delta^{art(p)}\}$ which is a contradiction to I being a model of $ground(\mathcal{T}, \Delta)$. We now take an arbitrary constant $d \in \Delta$ and for every constant $d' \notin \Delta$, we replace each occurrence of d' in this proof by d. Note that this procedure still yields a proof of $\bot$, but from a different set of initial clauses, call it S'. Consider a clause C from S that contains atoms using constants from $\mathsf{N_I} \setminus \Delta$. Then $C \in ground(\mathcal{T}, \mathsf{N_I})$. As these constants do not occur in $\mathcal{T}$, C must have been obtained during grounding from some $\varphi \in \mathcal{T}$ by replacing the variables in φ were replaced by these constants. This however means that when we performed grounding to obtain $ground(\mathcal{T}, \Delta)$, we must have also replaced these variables with d and hence there is a clause in $ground(\mathcal{T}, \Delta)$ that corresponds to C in which all occurrences of constants that are not in Δ are replaced by d. Hence, each clause in S' is in $ground(\mathcal{T}, \Delta)$, which means that the proof obtained by this procedure is a proof of $\bot$ in $ground(\mathcal{T}, \Delta) \cup I \ \cup \{\neg p(\vec{c}) : p(\vec{c}) \notin I, \vec{c} \in \Delta^{art(p)}\}$. This is however a contradiction to I being a model of $ground(\mathcal{T}, \Delta)$. Hence, $\mathcal{T}''$ must be consistent.

Let H be a model of $\mathcal{T}''$ and consider an interpretation H' obtained from H by deleting every atom that uses a predicate from Σ_{out} and a constant from $\mathsf{N_I} \setminus \Delta$. Assume H' is not a model of $\mathcal{T}'$. Then, there must be a clause in $ground(\mathcal{T}, \mathsf{N_I})$ that H' does not satisfy. Additionally, such a clause must contain a positive literal that uses a predicate from Σ_{out} and constants from $\mathsf{N_I} \setminus \Delta$. As these constants do not occur in $\mathcal{T}$, this clause must have been obtained from some $\varphi' \in \mathcal{T}$ by replacing variables in φ' with these constants. But as each variables that occur in a positive literal using predicates from Σ_{out} must be guarded by a negative literal also using a predicate from Σ_{out}, this clause contains at least one negative literal that uses a constant from from $\mathsf{N_I} \setminus \Delta$. Due to our construction of H', this clause is satisfied, which is a contradiction to H' not being a model of $\mathcal{T}'$. Thus $\mathcal{T}'$ is consistent. It follows that there is a model J of $ground(\mathcal{T}, \mathsf{N_I})$ s.t. $\{p(\vec{c}) : p(\vec{c}) \in J, \vec{c} \in \Delta^{art(p)}\} = I$ and $J|_{\Sigma_{\mathsf{out}}} = I|_{\Sigma_{\mathsf{out}}}$. □

Intuitively, the lemma above tells us that, in the considered fragment of RLPs, we can extend a model I of $ground(\mathcal{T}, \Delta)$, where Δ consists of all constants from $adom(\mathcal{P})$ and $\mathcal{T}$, into a model J of $ground(\mathcal{T}, \mathsf{N_I})$ such that (i) J agrees with I on the output predicates and (ii) if $p(\vec{c})$, $p \in \Sigma_{\mathsf{owa}}$, is in J but not in I, then no constants from $adom(\mathcal{P})$ and $\mathcal{T}$ occur in $\vec{c}$. Roughly, the second condition ensures that I fully specifies those parts of extensions of open predicates that involve known constants. Well-known results from

first order logic then tell us that J is a model of $\mathcal{T}$, which means that there is a model of $\mathcal{T}$ that agrees with I on closed predicates.

Before we provide the construction of $\mathcal{P}^{\Pi}_{check}$, we define the notion of a *partial reduct* $\mathcal{P}^{I,\Sigma,\Sigma'}$ of $\mathcal{P}$ w.r.t. an interpretation I and two sets Σ and Σ' of predicates.

Definition 7.2.6. *Let $\mathcal{P}$ be a $\mathsf{Datalog}^{\neg}$ program, I be an interpretation and $\Sigma, \Sigma' \subseteq \mathsf{N_P}$. The* partial reduct $\mathcal{P}^{I,\Sigma,\Sigma'}$ *of $\mathcal{P}$ w.r.t. I, Σ and Σ' is obtained from ground$(\mathcal{P}, \mathsf{N_I})$ as follows:*

1. *Delete every rule ρ that contains a literal $p(\vec{u})$ such that:*
   - *$p(\vec{u}) \in body^{+}(\rho)$, $p(\vec{u}) \notin I$, and $p \in \Sigma$,*
   - *$p(\vec{u}) \in head(\rho)$, $p(\vec{u}) \in I$, and $p \in \Sigma$, or*
   - *$p(\vec{u}) \in body^{-}(\rho)$ and $p(\vec{u}) \in I$.*
2. *In the remaining rules, delete negated atoms over the predicates from $\mathsf{sig}(\mathcal{T}) \setminus \Sigma'$.*

Observe that if predicates from Σ' occur negated only in constrains, the program $\mathcal{P}^{I,\Sigma,\Sigma'}$ either has no answer sets of it has a *unique* answer set. Indeed, we can see this program as a union of a plain Datalog program (i.e, a program that contains no negation) and a set of constraints. Recall that the latter cannot be used to derive any new information but only to eliminate potential answer sets. Thus, $\mathcal{P}^{I,\Sigma,\Sigma'}$ either has no answer sets or its answer set is the unique minimal model of its plain Datalog component (see Proposition 2.4.6).

Lemma 7.2.7. *Let $\Pi = (\mathcal{P}, \mathcal{T}, \Sigma_{\mathsf{out}}, \Sigma_{\mathsf{owa}}, \Sigma_{\mathsf{re}})$ be an RLP in the considered class. If I is an answer set of Π, each model J of $\mathcal{T}$ with $J|_{\Sigma_{\mathsf{out}}} = I$ has a unique response M w.r.t. I and Π. Moreover, M coincides with the unique answer set of $\mathcal{P}^{J,\Sigma_{\mathsf{owa}},\Sigma_{\mathsf{re}}}$.*

Proof. Consider an arbitrary response H to J w.r.t. I and Π. From $H|_{\Sigma_{\mathsf{out}}\cup\Sigma_{\mathsf{owa}}} = J$ it follows that $\mathcal{P}^{H|_{\Sigma_{\mathsf{out}}\cup\Sigma_{\mathsf{owa}}},\Sigma_{\mathsf{owa}},\Sigma_{\mathsf{re}}} = \mathcal{P}^{J,\Sigma_{\mathsf{owa}},\Sigma_{\mathsf{re}}}$. That means that $\mathcal{P}^{H,\Sigma_{\mathsf{owa}}}$ is a subset of $\mathcal{P}^{J,\Sigma_{\mathsf{owa}},\Sigma_{\mathsf{re}}}$. Moreover, the only difference between the two is that $\mathcal{P}^{J,\Sigma_{\mathsf{owa}},\Sigma_{\mathsf{re}}}$ might contain additional constraints $\rho_{c_1}, \ldots, \rho_{c_n}$ that have negated atoms using response predicates. Recall that a constraint is a rule of the form $p \leftarrow \alpha, not\ p$ (abbreviated as $\leftarrow \alpha$), where p is a fresh propositional atom that does not occur elsewhere in $\mathcal{P}$.

Let a be an arbitrary atom in M. By definition of constraints, $a \notin head(\rho_{c_i})$, for all $i \geq 1$. Thus, in order for M to be the minimal model, a must have a derivation from facts using rules of $\mathcal{P}^{J,\Sigma_{\mathsf{owa}},\Sigma_{\mathsf{re}}} \setminus \{\rho_{c_1}, \ldots \rho_{c_n}\}$. As all rules in the derivation of a are also present in $\mathcal{P}^{H,\Sigma_{\mathsf{owa}}}$, a must be contained it the minimal model of this program. Furthermore, as H is a response to J w.r.t. I, $H|_{\Sigma_{\mathsf{out}}\cup\Sigma_{\mathsf{re}}}$ is the minimal model of $\mathcal{P}^{H,\Sigma_{\mathsf{owa}}}$, so $a \in H|_{\Sigma_{\mathsf{out}}\cup\Sigma_{\mathsf{re}}}$.

Assume now that $a \in H|_{\Sigma_{\mathsf{out}}\cup\Sigma_{\mathsf{re}}}$. As $H|_{\Sigma_{\mathsf{out}}\cup\Sigma_{\mathsf{re}}}$ is the minimal model of $\mathcal{P}^{H,\Sigma_{\mathsf{owa}}}$, a must have a derivation from facts using rules in $\mathcal{P}^{H,\Sigma_{\mathsf{owa}}}$. As $\mathcal{P}^{H,\Sigma_{\mathsf{owa}}} \subseteq \mathcal{P}^{J,\Sigma_{\mathsf{owa}},\Sigma_{\mathsf{re}}}$, then a is also derivable using the facts and rules in $\mathcal{P}^{J,\Sigma_{\mathsf{owa}},\Sigma_{\mathsf{re}}}$, and so $a \in M$. Hence, we have $M = H|_{\Sigma_{\mathsf{out}}\cup\Sigma_{\mathsf{re}}}$. □

We are now ready to build $\mathcal{P}^{\Pi}_{check}$. We again start by adding $\mathsf{Const}(c)$ for every $c \in \mathsf{N_I}$ that occurs in $\mathcal{T}$ or $adom(\mathcal{P})$. We then add, for each $q \in \Sigma_{\mathsf{owa}}$,

$$q'(\vec{x}) \leftarrow \mathsf{Const}^k(\vec{x}), not\ \overline{q}'(\vec{x})$$
$$\overline{q}'(\vec{x}) \leftarrow \mathsf{Const}^k(\vec{x}), not\ q'(\vec{x}),$$

where $\vec{x}$ is k-tuple of variables from $\mathsf{N_V}$ and k is the arity of q.

Intuitively, these rules extend our previously-guessed structure over the output predicates I by guessing the extensions of the open predicates. Let J denote this extension. We now need to check whether J is a counter-example to I being an answer set of Π, i.e., we check whether the following two hold:

1. $J \cup I$ can be extended into a model of $\mathcal{T}$, and
2. we cannot find an extension H of J by interpreting the response predicates such that H is a minimal model of $\mathcal{P}^{H,\Sigma_{\mathsf{owa}}}$. In other words, we cannot find a response for J.

To check the first condition, for all $\forall x_1, \ldots, x_l(a_1 \vee \ldots \vee a_j \vee \neg a_{j+1} \vee \ldots \vee \neg a_h)$ in $\mathcal{T}$, we add:

$$\leftarrow a'_{j+1}, \ldots, a'_h, not\ a'_1, \ldots, not\ a'_j, \mathsf{Const}^l(x_1, \ldots, x_l),$$

where, for $1 \leq i \leq h$, a'_i is obtained from a_i by replacing the predicate symbol p by p', if $p \in \Sigma_{\mathsf{owa}}$, otherwise $a'_i = a_i$.

We deal with the second condition as follows. In view of Lemma 7.2.7, we only need to test whether the minimal model M of the partial reduct of $\mathcal{P}$ w.r.t. J, Σ_{owa} and Σ_{re} is a response to J w.r.t I and Π, i.e. we need to check if M is a minimal model of $\mathcal{P}^{M \cup J, \Sigma_{\mathsf{owa}}}$. If not, we obtained a counter-example.

We now do the following: for every rule $\rho \in \mathcal{P}$, we add the rule ρ' to $\mathcal{P}^{\Pi}_{check}$ obtained from ρ by (i) replacing each occurrence of p in $body^+(\rho) \cup head(\rho)$ by p', for each $p \in \Sigma_{\mathsf{out}}$, (ii) replacing each occurrence of q in $head(\rho)$ by q'', for each $q \in \Sigma_{\mathsf{owa}}$, (ii) replacing all remaining occurrences of q in ρ by q', for each $q \in \Sigma_{\mathsf{owa}}$. Further, if ρ is a constraint we set $head(\rho') = \{\mathsf{Violation}\}$.

To complete our construction, for each $p \in \Sigma_{\mathsf{out}}$ we add:

$$\mathsf{not\_equal} \leftarrow p(\vec{x}), not\ p'(\vec{x}), \mathsf{Const}^k(\vec{x}) \qquad \mathsf{not\_equal} \leftarrow p'(\vec{x}), not\ p(\vec{x}), \mathsf{Const}^k(\vec{x}),$$

where $\vec{x}$ is an arbitrary k-ary tuple of variables with k the arity of p, and for each $q \in \Sigma_{\mathsf{owa}}$:

$$\mathsf{Violation} \leftarrow q''(\vec{x}), not\ q'(\vec{x}), \mathsf{Const}^k(\vec{x}),$$

where $\vec{x}$ is an arbitrary k-ary tuple of variables with k the arity of q. Finally, we add $\leftarrow not\ \mathsf{not\_equal}, not\ \mathsf{Violation}$.

	combined	bounded arities	data
DL-Lite$_\mathcal{F}$	NExpTime $^{\text{NP}}$-complete	Σ_3^P- complete	Σ_3^P- complete
$\mathcal{ALCHI}$	NExpTime $^{\text{NP}}$-complete	ExpTime-complete	Σ_3^P- complete
$\mathcal{ALCHOIQ}$	in NExpTime $^{\text{NExpTime}}$	in NP $^{\text{NExpTime}}$	Σ_3^P- complete

Table 7.1: Complexity of answer set existence for safe RLPs.

Intuitively, $\mathcal{P}^{\Pi}_{check}$ computes and "stores" the minimal model M of the partial reduct using predicate symbols in $\Sigma_{\mathsf{re}} \cup \{p' : p \in \Sigma_{\mathsf{out}}\}$. If during the computation of M a constraint in $\mathcal{P}$ is violated we know that M cannot be a response to J and so we derive the atom Violation. We also use the predicate symbols in $\{q'' : q \in \Sigma_{\mathsf{owa}}\}$ to keep track of whether there are atoms in M that use open predicates but were not a part of our guess. If such an atom is derived, then there is a violation of some constraint in $\mathcal{P}^{M \cup J, \Sigma_{\mathsf{owa}}}$. This in turn means that M is not a response to J.

If no constraint was violated we have to compare whether exactly those atoms involving output predicates were derived that are in I. If this is not the case, we derive not_equal which again means that M is not a response to J. If neither Violation nor not_equal are derived, M is a response to J. The final constraint in our construction is then used to "kill" an answer set of $\mathcal{P}_{check}$ that does not define a counter-example to I being an answer set of Π.

It can be easily verified that the following holds: if S is an answer set of Π then there is an answer set S' of $\mathcal{P}^{\Pi}_{guess}$ s.t. $S'|_{\Sigma_{\mathsf{out}}} = S$ and $\mathcal{P}^{\Pi}_{check} \cup S'$ does not have an answer set. Conversely, if S' is an answer set of $\mathcal{P}^{\Pi}_{guess}$ s.t. $\mathcal{P}^{\Pi}_{check} \cup S'$ has no answer set, $S'|_{\Sigma_{\mathsf{out}}}$ is an answer set of Π. Hence, the answer sets of Π are in one-to-one correspondence with answer sets of $\mathcal{P}^{\Pi}_{solve}$ (modulo the predicates from Σ_{owa}).

7.3 Resilient Logic Programs with DL Theories

In this section, we focus on RLPs whose theory component is given as a DL TBox in one of the considered logics and we provide some complexity results.

Description Logics Preliminaries. Recall that most DLs, in particular those considered in this chapter, are fragments of FO with only unary and binary predicate symbols, and a slightly modified syntax that allows us to write formulas more concisely. We next consider RLPs whose theories are written in one of the following DLs: the expressive $\mathcal{ALCHI}$ and $\mathcal{ALCHOIQ}$, and the lightweight DL-Lite$_\mathcal{F}$. The first two DLs have already been introduced and studied throughout this thesis. The last one is an extension of the basic lightweight DL-Lite with global role functionality. We next recall the syntax of these logics. For all three logics, roles are defined according to the following syntax:

$r := p \mid p^-$, where $p \in \mathsf{N_R}$, whereas complex concepts are defined as follows:

$$\begin{aligned} \mathsf{DL\text{-}Lite}_{\mathcal{F}}: &\ C := \bot \mid A \mid \exists r \\ \mathcal{ALCHI}: &\ C := \top \mid A \mid \neg C \mid C \sqcap C \mid C \sqcup C \mid \exists r.C \mid \forall r.C \\ \mathcal{ALCHOIQ}: &\ C := \top \mid A \mid \neg C \mid C \sqcap C \mid C \sqcup C \mid \geq nr.C \mid \leq nr.C \mid \{a\}, \end{aligned}$$

where $A \in \mathsf{N_C}$, r is a role, $a \in \mathsf{N_I}$, and $n \geq 0$. Hereinafter, C, D denote complex concepts and r, s denote roles in the considered logic. Moreover, we abbreviate $\leq nr.C \sqcap \geq nr.C$ by $= nr.C$.

In $\mathcal{ALCHI}$ and $\mathcal{ALCHOIQ}$, a TBox is a finite set concept inclusions of the form $C \sqsubseteq D$ and role inclusions of the form $r \sqsubseteq s$. In $\mathsf{DL\text{-}Lite}_{\mathcal{F}}$, a TBox consists of concept inclusions of the form $C \sqsubseteq D$ and $C \sqsubseteq \neg D$ and axioms of the form $\mathsf{func}(r)$.

7.3.1 Relaxed Safety for DL RLPs

Recall the RLP from Example 7.1.6. This RLP is considered unsafe as there are rules that contain atoms of the form $\mathsf{Task}(x)$ in which the variable x is not safeguarded by a closed predicate. However, it is easy to see that Task is a bounded predicate with respect to the given theory and the set of output predicates. According to our results from Chapter 6, in any model of the theory, there is an upper bound on the number of constants that can occur in atoms over the predicate Task that depends on the theory and the number of constants that occur in output predicates, which makes this predicate a suitable guard for program variables.

The results in Theorem 6.3.9 and Theorem 6.3.10 from the previous chapter give us a concrete upper bound on the number of distinct elements in bounded predicates, however this number is very high – doubly-exponential in the size of $\mathcal{T}$. To keep the cost of reasoning lower, especially for $\mathcal{ALCHI}$ and $\mathsf{DL\text{-}Lite}_{\mathcal{F}}$, we next present a way to obtain a certain set of bounded concept names that is incomplete, but for which we can give a better bound on the number of distinct elements that can occur in their extensions.

Approximation of bounded concept names. We begin by formally defining the particular set $\mathbf{B}_{\text{cn}}(\mathcal{T}, \Sigma)$ of concept names bounded w.r.t. a TBox $\mathcal{T}$ and a set Σ of predicates that we consider for our safety condition.

Definition 7.3.1. *Let $\mathcal{T}$ be an $\mathcal{ALCHOIQ}$ TBox, Σ be a finite set with $\Sigma \subseteq \mathsf{N_P}$, and $\mathsf{Nom}(\mathcal{T}) = \{\{c\} : c \text{ occurs in } \mathcal{T}\}$. The set $\boldsymbol{B}_{cn}(\mathcal{T}, \Sigma)$ of* bounded concept names *w.r.t. $\mathcal{T}$ and Σ is the smallest set that contains every concept name $A \in \mathsf{N_C}(\mathcal{T})$ for which at least one of the following holds:*

1. *$A \in \Sigma$,*
2. *there is a role $r \in \mathsf{N_R^+}(\mathcal{T}) \cap \Sigma$ s.t. $\mathcal{T} \models A \sqsubseteq \exists r.\top$,*
3. *$\mathcal{T} \models A \sqsubseteq \bigsqcup_{B \in \mathcal{B} \cup \mathsf{Nom}(\mathcal{T})} B$, or*

Algorithm 7.2: Algorithm for computing $\mathbf{B}_{\text{cn}}(\mathcal{T}, \Sigma)$ for a given $\mathcal{T}$ and Σ, where $\mathcal{T}$ is in $\mathcal{ALCHOIQ}$

Algorithm getBoundedCN($\mathcal{T}, \Sigma, b$)
  Input : An $\mathcal{ALCHOIQ}$ TBox $\mathcal{T}$, $\Sigma \subseteq \mathsf{N_P}$, and an integer b
  Output : A set of bounded concept names in $\mathcal{T}$ w.r.t. Σ, an integer bound b'
  $\mathcal{B} \leftarrow \{A : A \in \mathsf{N_I}(\mathcal{T}) \cap \Sigma\}$
  $\mathsf{Nom}(\mathcal{T}) \leftarrow \{\{a\} : a \in \mathsf{N_I}$ occurs in $\mathcal{T}\}$
  $n \leftarrow$ max int i s.t. $\leq i$ occurs in $\mathcal{T}$
  $m \leftarrow$ min int i s.t. $\geq i$ occurs in $\mathcal{T}$
  $b' \leftarrow b$
  $\mathcal{B}' \leftarrow \mathcal{B}$
  repeat
    $\mathcal{B} \leftarrow \mathcal{B}'$
    foreach *concept name* $A \in \Sigma(\mathcal{T}) \setminus \mathcal{B}$ **do**
      if $\mathcal{T} \models A \sqsubseteq \bigsqcup_{B \in \mathcal{B} \cup \mathsf{Nom}(\mathcal{T})} B$ **then** $f \leftarrow$ true
      if *not* f **then**
        foreach *role name* $r \in \Sigma$ **do**
          if ($\mathcal{T} \models A \sqsubseteq \exists r.\top$ or $\mathcal{T} \models A \sqsubseteq \exists r^-.\top$) **then**
            $f \leftarrow$ true
            break
      if *not* f **then**
        foreach *role* $r \in \mathsf{N}_R^+(\mathcal{T})$ **do**
          foreach $B \in \mathcal{B} \cup \mathsf{Nom}(\mathcal{T})$ **do**
            if $\mathcal{T} \models A \sqsubseteq \geq mr.B$ and $\mathcal{T} \models B \sqsubseteq \leq nr^-.A$ **then**
              $b' \leftarrow b' + \lfloor \frac{b' \cdot n}{m} \rfloor$
              $f \leftarrow$ true
              break
      if f **then** $\mathcal{B}' \leftarrow \mathcal{B}' \cup \{A\}$
  until $\mathcal{B} \neq \mathcal{B}'$
  return $(\mathcal{B}, b')$

4. *there exists* $B \in \boldsymbol{B}_{cn}(\mathcal{T}, \Sigma) \cup \mathsf{Nom}(\mathcal{T})$, *a role* $r \in \mathsf{N}_R^+(\mathcal{T})$, *and integers* $n, m \geq 0$ *s.t.* $\mathcal{T} \models A \sqsubseteq \geq mr.B$ *and* $\mathcal{T} \models B \sqsubseteq \leq nr^-.A$.

If $\mathcal{T}$ *is in* $\mathcal{ALCHI}$, *item 4. in the definition above is omitted and* $\mathsf{Nom}(\mathcal{T}) = \emptyset$. *Similarly, if* $\mathcal{T}$ *is in DL-Lite*$_{\mathcal{F}}$, *item 3. is omitted and item 4. is replaced by the following:*

4a. *there exists* $B \in \boldsymbol{B}_{cn}(\mathcal{T}, \Sigma)$ *and a role* $r \in \mathsf{N}_R^+(\mathcal{T})$ *s.t.* $\mathcal{T} \models A \sqsubseteq \exists r^-$, $\mathcal{T} \models \exists r \sqsubseteq B$, *and* $\mathsf{func}(r) \in \mathcal{T}$.

Algorithm 7.2 computes the set $\mathbf{B}_{\text{cn}}(\mathcal{T}, \Sigma)$ for a given $\mathcal{ALCHOIQ}$ TBox $\mathcal{T}$ and a set of predicate symbols $\Sigma \subseteq \mathsf{N_P}$. This algorithm can easily be modified to cater to both

$\mathcal{ALCHI}$ and DL-Lite$_\mathcal{F}$. Indeed, for these logics, we replace line 2 with $\mathsf{Nom}(\mathcal{T}) \leftarrow \emptyset$. Further, for $\mathcal{ALCHI}$, lines 17-22 are omitted from Algorithm 7.2. For DL-Lite$_\mathcal{F}$, lines 19-20 are replaced by the following:

19 **if** $\mathcal{T} \models A \sqsubseteq \exists r^-$ and $\mathcal{T} \models \exists r \sqsubseteq B$ and $\mathsf{func}(r) \in \mathcal{T}$ **then**
20 $\quad b' \leftarrow b' + b$

As reasoning in selected logics is decidable, the algorithm is guaranteed to terminate and hence $\mathbf{B}_{\mathrm{cn}}(\mathcal{T}, \Sigma)$ can be effectively computed. In fact, given $\mathcal{T}$ and Σ, computing the set $\mathbf{B}_{\mathrm{cn}}(\mathcal{T}, \Sigma)$ requires a polynomial number of steps, some of which use an entailment test as an oracle.

Our algorithm also takes as input an upper bound b on the number of different constants that can occur in extensions of predicates from Σ and computes an upper bound b' on the number of different constants that can occur in extensions of predicates in $\mathbf{B}_{\mathrm{cn}}(\mathcal{T}, \Sigma)$. By closely inspecting Algorithm 7.2 we get that the bound b' is at most exponential for $\mathcal{ALCHOIQ}$ and polynomial in DL-Lite$_\mathcal{F}$ and $\mathcal{ALCHI}$. In fact, for $\mathcal{ALCHI}$ $b' = b$. This is observation is formalized in the following proposition.

Proposition 7.3.2. *Let $\mathcal{T}$ be a TBox in one of the DLs above, $\Sigma \subseteq \mathsf{N_P}$, and $b \geq 0$. We can compute a constant b' s.t. $|\{c : A(c) \in J, A \in \boldsymbol{B}_{cn}(\mathcal{T}, \Sigma)\}| \leq b'$ holds in every model J of $\mathcal{T}$ in which $|\{c : p(\vec{c}) \in J, p \in \Sigma, c \text{ occurs in } \vec{c}\}| \leq b$. If $\mathcal{T}$ is in $\mathcal{ALCHOIQ}$, b' has the value that is at most exponential in the size of $\mathcal{T}$. Otherwise, b' has the value that is polynomial in the size of $\mathcal{T}$.*

Proof. Let J be an arbitrary model of $\mathcal{T}$ s.t. $|\{\vec{c} : p(\vec{c}) \in J, p \in \Sigma\}| \leq b$. Further, let $\Delta_0 = \{\vec{c} : p(\vec{c}) \in J, p \in \Sigma\} \cup \mathsf{N_I}(\mathcal{T})$ and let $\mathbf{B}^0_{cn}(\mathcal{T}, \Sigma)$ be the set of all the concept names that we can infer are in $\mathbf{B}_{\mathrm{cn}}(\mathcal{T}, \Sigma)$ using only rules 1-3 in Def. 7.3.1. Obviously, if a concept name is in $\mathbf{B}^0_{cn}(\mathcal{T}, \Sigma)$, then only the elements of Δ_0 can be in its extension. Hence, an upper bound on the number of different elements in the extension of any concept name in $\mathbf{B}^0_{cn}(\mathcal{T}, \Sigma)$ is $b_0 = |\Delta_0|$. Let $\mathbf{B}^1_{cn}(\mathcal{T}, \Sigma)$ be the set of all the concept names that we can infer are in $\mathbf{B}_{\mathrm{cn}}(\mathcal{T}, \Sigma)$ using only rules 1-3 in Def. 7.3.1 and a single application of rule 4. Assume we inferred that some concept name A is in $\mathbf{B}^1_{cn}(\mathcal{T}, \Sigma)$ directly by the rule 4. Then there is some $B \in \mathbf{B}^0_{cn}(\mathcal{T}, \Sigma)$, a role r in $\mathcal{T}$, and integers $n, m \geq 0$ s.t. $\mathcal{T} \models A \sqsubseteq \geq mr.B$ and $\mathcal{T} \models B \sqsubseteq \leq nr^-.A$. It is not hard to see that even though we do not know which elements occur in the extension of A, there could be at most $\lfloor \frac{b_0 \cdot n}{m} \rfloor$ of them. Then, the extension of any bounded concept names in $\mathbf{B}^1_{cn}(\mathcal{T}, \Sigma)$ contains at most $b_1 = b_0 + \lfloor \frac{b_0 \cdot n}{m} \rfloor$. We repeat these calculations, at each level obtaining a new bound $b_i = b_{i-1} + \lfloor \frac{b_{i-1} \cdot n}{m} \rfloor$, until $\mathbf{B}^n_{cn}(\mathcal{T}, \Sigma) = \mathbf{B}^{n+1}_{cn}(\mathcal{T}, \Sigma)$. Finally, b_n is an upper bound on the number of different constants that can occur in the extension of some concept name in $\mathbf{B}_{\mathrm{cn}}(\mathcal{T}, \Sigma)$. Similar observations can be made also in the case where functionality is the only allowed number restriction. □

In what follows, a predicate symbol p occurring in an RLP $\Pi = (\mathcal{P}, \mathcal{T}, \Sigma_{\mathsf{out}}, \Sigma_{\mathsf{owa}}, \Sigma_{\mathsf{re}})$ is said to be *bounded* in Π if $p \in \Sigma_{\mathsf{out}} \cup \Sigma_{\mathsf{re}} \cup \mathbf{B}_{\mathrm{cn}}(\mathcal{T}, \Sigma_{\mathsf{out}})$.

Relaxed safety. We now use an approach similar to those from the previous chapter to significantly relax the safety restriction from Definition 7.2.1. The main idea here is to guard every variable with a positive body atom over some bounded predicate which means that program variables can only range over a bounded number of constants. However, in order for our notion of boundedness to make sense, we need to ensure that the output predicates themselves are not directly or indirectly guarded by bounded open predicats, i.e., that the variables in output predicates still range only over the active domain of the program. To this end, we introduce the following auxiliary notions.

Given a program $\mathcal{P}$, for each $p \in \mathsf{sig}(\mathcal{P})$ and $1 \leq i \leq \mathit{art}(p)$, we denote by $p[i]$ the i-th *position* in the predicate p. Given a set of predicates Σ, the set $\mathsf{ap}(\mathcal{P}, \Sigma)$ of *affected positions* (see [CGK13]) in $\mathcal{P}$ w.r.t. Σ is inductively defined as follows:

- $p[i] \in \mathsf{ap}(\mathcal{P}, \Sigma)$, for $p \in \Sigma$ and $1 \leq i \leq \mathit{art}(p)$, and
- if there exists a rule $\rho \in \mathcal{P}$ s.t. a variable x appears in $\mathit{body}^+(\rho)$ only in affected positions and x appears in $\mathit{head}(\rho)$ in position π, then $\pi \in \mathsf{ap}(\mathcal{P}, \Sigma)$.

Intuitively, the set of affected positions consists of all the positions of the program predicates in which a constant that is not in the active domain of the program can appear.

Definition 7.3.3. *An RLP* $\Pi = (\mathcal{P}, \mathcal{T}, \Sigma_{\mathsf{out}}, \Sigma_{\mathsf{owa}}, \Sigma_{\mathsf{re}})$ *fulfills* the relaxed safety condition *if for each rule* $\rho \in \mathcal{P}$*: every variable in* ρ *occurs in some* $p(\vec{u}) \in \mathit{body}^+(\rho)$*, where* p *is a bounded predicate in* Π *and* $q[i] \notin \mathsf{ap}(\mathcal{P}, \boldsymbol{B}_{cn}(\mathcal{T}, \Sigma_{\mathsf{out}}) \setminus \Sigma_{\mathsf{out}})$*, for all* $q \in \Sigma_{\mathsf{out}}$ *and* $1 \leq i \leq \mathit{art}(q)$*.*

Algorithm 7.3 is an adaptation of Algorithm 7.1 that checks whether an RLP with a DL theory fulfilling the relaxed safety condition has an answer set. Specifically, we modify the subroutine hasCE to guess a candidate structure J for the counter-example to I being the answer set of Π over a larger set of constants – exponential in the size of $\mathcal{T}$ if $\mathcal{T}$ is an $\mathcal{ALCHOIQ}$ TBox, and polynomial if $\mathcal{T}$ is given in $\mathcal{ALCHI}$ or $\mathsf{DL\text{-}Lite}_{\mathcal{F}}$. We note that, in the original algorithm, this guess is limited to the constants that already appear in the input rules.

Theorem 7.3.4. *For* $\mathsf{DL\text{-}Lite}_{\mathcal{F}}$, $\mathcal{ALCHI}$ *and* $\mathcal{ALCHOIQ}$ *the procedure hasAnswerSet(*Π*) from Algorithm 7.3 correctly decides the existence of an answer set of* Π*.*

The intuition behind the result above is similar to the intuition behind Theorem 7.2.3. Namely, Algorithm 7.3 takes an RLP $\Pi = (\mathcal{P}, \mathcal{T}, \Sigma_{\mathsf{out}}, \Sigma_{\mathsf{owa}}, \Sigma_{\mathsf{re}})$, guesses an interpretation I over the output predicates and the active domain of $\mathcal{P}$. We note that our relaxed safety condition excludes the possibility that all positions of output predicates are not affected, hence these predicates range only over the constants in the program and it suffices to guess I over the active domain. After this initial guess, two oracle calls are

Algorithm 7.3: Answer set existence of RLPs with DL theories under relaxed safety, where isConsistent$(\mathcal{T}, \Sigma, I)$ checks consistency of DL KB $(\mathcal{T}, I)$ with closed predicates Σ.

Algorithm hasAnswerSet(Π)
  Input : An RLP $\Pi = (\mathcal{P}, \mathcal{T}, \Sigma_{\mathsf{out}}, \Sigma_{\mathsf{owa}}, \Sigma_{\mathsf{re}})$
  Output : *true* iff Π has an answer
  Guess an interpretation I over Σ_{out} using constants from $adom(\mathcal{P})$
  $\mathcal{B} \leftarrow \mathbf{B}_{\mathrm{cn}}(\mathcal{T}, \Sigma_{\mathsf{out}})$
  $b \leftarrow$ max number of different const. in extensions of bounded concept names
  $\Delta \leftarrow adom(\mathcal{P}) \cup \mathsf{N_I}(\mathcal{T}) \cup \{a_1, \ldots, a_{b-b'}\}$, where $b' = |adom(\mathcal{P})| + |\mathsf{N_I}(\mathcal{T})|$
  return isConsistent$(\mathcal{T}, \Sigma_{\mathsf{out}}, I)$ and *not* hasCE$(I, \Pi, \Delta, \mathcal{B})$

Subroutine hasCE($I, \Pi, \Delta, \mathcal{B}$)
  Input : An interpretation I, an RLP $\Pi = (\mathcal{P}, \mathcal{T}, \Sigma_{\mathsf{out}}, \Sigma_{\mathsf{owa}}, \Sigma_{\mathsf{re}})$, $\Delta \subseteq \mathsf{N_I}$, and $\mathcal{B} \subseteq \mathsf{sig}(\mathcal{T})$
  Output : *true* iff there is a counter example to I being an answer set of Π
  Guess an interpretation J over $\mathsf{sig}(\mathcal{P}) \cap \mathsf{sig}(\mathcal{T}) \cup \Sigma_{\mathsf{out}}$ and constants from Δ s.t. $J|_{\Sigma_{\mathsf{out}}} = I|_{\Sigma_{\mathsf{out}}}$
  $J' \leftarrow J \cup \{\neg p(\vec{c}) : \vec{c} \in \Delta^{art(p)}, p \in \mathsf{sig}(\mathcal{P}) \cap (\mathsf{sig}(\mathcal{T}) \setminus \mathcal{B}),$ and $p(\vec{c}) \notin J\}$
  return isConsistent$(\mathcal{T}, \mathcal{B}, J')$ and *not* hasResp(J, Π)

Subroutine hasResp(J, Π)
  Input : An interpretation J and an RLP $\Pi = (\mathcal{P}, \mathcal{T}, \Sigma_{\mathsf{out}}, \Sigma_{\mathsf{owa}}, \Sigma_{\mathsf{re}})$
  Output : *true* iff there is a response for J w.r.t. $J|_{\Sigma_{\mathsf{out}}}$ and Π
  Guess an interpretation H over $\mathsf{sig}(\mathcal{P})$ and the constants from J and $adom(\mathcal{P})$ s.t. $H|_{\Sigma_{\mathsf{out}} \cup \Sigma_{\mathsf{owa}}} = J$
  if $H|_{\Sigma_{\mathsf{out}} \cup \Sigma_{\mathsf{re}}}$ *is a min. model of* $\mathcal{P}^{H, \Sigma_{\mathsf{owa}}}$ **then return** true
  else return false

made to check whether I is indeed an answer set of Π. As before, the first call checks whether there is a model I can be extended into a model of $\mathcal{T}$ without modifying the interpretation of the output predicates. This involves evoking a decision procedure for KB satisfiability in the presence of closed predicates for the chosen logic. We note that for our logics this problem is known to be decidable. The second call tries to verify whether each such model of $\mathcal{T}$ has a response w.r.t. I and Π by attempting to guess a model J of $\mathcal{T}$ for which this does not hold. This guess is done over a set Δ of constants such that $\mathsf{N_I}(\mathcal{T}), adom(\mathcal{P}) \subseteq \Delta$ and $|\Delta| = b$, where b is a bound on the number of different constants that can occur in bounded concept names as computed by Algorithm 7.2. We remark that Δ contains exponentially many constants in the size of $\mathcal{T}$, if $\mathcal{T}$ is an $\mathcal{ALCHOIQ}$ TBox, and polynomially many constants in the size of $\mathcal{T}$, if $\mathcal{T}$ is written in $\mathcal{ALCHI}$ or DL-Lite$_{\mathcal{F}}$. It is easy to see that this guess is sufficient due to our safety restriction that ensures that the variables of $\mathcal{P}$ range only over the constants that appear

in bounded concept names in answer sets of Π, however, a formal proof can be found in the appendix to this thesis. Finally, to check that J has no response, another oracle call is made which tries to guess a response. If this fails, there is a counter-example to I being an answer set of Π.

Complexity results. By analyzing Algorithm 7.3 and relying on known complexity results for $\mathsf{Datalog}^\neg$ and DLs with closed predicates, we can characterize the complexity of RLPs whose theories are given as TBoxes expressed in one of the logics mentioned above under the relaxed safety. We provide results for both combined complexity, measured in terms of the size of the whole RLP, as well as for data complexity in which only the size of the facts matter, while the size of the theory and the non-factual rules of the RLP's program component are treated as constant. Additionally, as the program component can contain predicates of arbitrary arities, we also provide some results for the case where these arities are bounded by a constant.

Theorem 7.3.5. *Table 7.1 provides correct complexity results for deciding the answer set existence for RLPs fulfilling the relaxed safety condition.*

We briefly guide the reader through the entries in Table 7.1. Observe that for predicates of arbitrary arities, the guess for I over the output predicates is at most exponential in $|\mathcal{P}|$. Let us first consider the logic $\mathcal{ALCHI}$. In view of Proposition 7.3.2, Δ is polynomial in $|\Pi|$ and thus there are only exponentially many different guesses for J. As $\mathcal{ALCHI}$ with closed predicates is ExpTime-complete, we modify the algorithm to do the following in exponential time: guess I, compute $\mathbf{B}_{\text{cn}}(\mathcal{T}, \Sigma_{\mathsf{out}})$, construct Δ and check whether there is a model of $\mathcal{T}$ that agrees with I on Σ_{out}. If so, iterate through possible guesses for J and check for each whether it corresponds to some model. If so, call an oracle to determine whether there is a response H to J. Due to unbounded predicate arities, both H and the grounding $\mathit{ground}(\mathcal{P}, \Delta)$ may be at most of exponential size. Thus, we can guess H, compute the corresponding reduct and verify whether $H|_{\Sigma_{\mathsf{out}} \cup \Sigma_{\mathsf{re}}}$ is its minimal model in exponential time. Observe that, by the standard padding argument [Pap94, AB09], this oracle can be implemented as an NP oracle, and so the overall complexity of the algorithm is NExpTime $^{\text{NP}}$. If predicate arities are bounded, instead of guessing I and H we simply iterate through all the possibilities. The ExpTime upper bound follows. Data complexity follows from the fact that consistency checking in $\mathcal{ALCHI}$ with closed predicates is in NP in terms of data complexity [LSW15]. Hence, each part of the original algorithm can be implemented as an NP oracle and the complexity result is immediate. As $\mathsf{DL\text{-}Lite}_{\mathcal{F}}$ with closed predicates is in NP both in data and in combined complexity [FIS11, GGI$^+$20], arguments for $\mathsf{DL\text{-}Lite}_{\mathcal{F}}$ are virtually the same as for $\mathcal{ALCHI}$. The difference is that in the case of bounded predicate arities, we obtain the Σ_3^P upper bound due to lower combined complexity of $\mathsf{DL\text{-}Lite}_{\mathcal{F}}$ with closed predicates. For $\mathcal{ALCHOIQ}$, the constructed set of constants Δ is at most exponential in $|\mathcal{T}|$, and polynomial if $\mathcal{T}$ is considered fixed. Moreover, consistency checking of $\mathcal{ALCHOIQ}$ knowledge bases with closed predicates is NExpTime-complete [Tob00], so it suffices to iterate through possible guesses for J (exponentially many in $|\Pi|$) and have a single NExpTime oracle to check

for consistency and the existence of response to J. The upper bound follows. Similarly, for the bounded arities case, we make our first guess for $\mathcal{I}$ – this time polynomial in the size of $\mathcal{P}$ – and then we place two calls to a NExpTime oracle to check for consistency and the existence of response to J. Finally, as we have shown earlier in this thesis that consistency checking in $\mathcal{ALCHOIQ}$ with closed predicates in in NP, the data complexity bound follows by the same argument that was used for $\mathcal{ALCHI}$.

The data-complexity bound for $\mathcal{ALCHOIQ}$ as well as all bounds for DL-Lite$_\mathcal{F}$ and $\mathcal{ALCHI}$ are tight. ExpTime-hardness for $\mathcal{ALCHI}$ in the case of bounded predicate arities is due to ExpTime-completeness of $\mathcal{ALCHI}$ with closed predicates. NExpTime $^{\text{NP}}$-hardness in the case of unbounded predicate arities is obtained by a reduction from answer set existence problem for disjunctive datalog which is known to be complete for this class. Finally, Σ_3^P-hardness is shown by a reduction from $\exists\forall\exists$-QBFs to RLPs with $\mathcal{ALCHI}$/DL-Lite$_\mathcal{F}$ theories. We note that the reduction that was provided in Example 7.1.5, although natural, does not allow us to infer the desired data-complexity lower-bound, since the the rules of $\mathcal{P}$ are dependent on the given QBF – indeed, we introduce one rule per each clause in the formula. Fortunately, we can modify this reduction in a way that encodes the given formula as facts and otherwise uses a fixed set of rules. We present this reduction below.

Σ_3^P lower bound for data-complexity. Let Φ be the following QBF:

$$\Phi = \exists X_1, \ldots, X_n \forall Y_1, \ldots, Y_m, \exists Z_1, \ldots, Z_k \varphi.$$

We first explain how we obtain the program $\mathcal{P}'$.

First we add to $\mathcal{P}'$ the following facts and rules that carry the same intuitive meaning as in the Example 7.1.5:

$$\begin{aligned}
&V_X(c_1^X), \ldots, V_X(c_n^X), \qquad V_Y(c_1^Y), \ldots, V_Y(c_m^Y), \qquad V_Z(c_1^Z), \ldots, V_Z(c_k^Z),\\
&T_X(x) \leftarrow V_X(x), \mathit{not}\ F_X(x), \qquad T_Z(x) \leftarrow V_Z(x), \mathit{not}\ F_Z(x),\\
&F_X(x) \leftarrow V_X(x), \mathit{not}\ T_X(x), \qquad F_Z(x) \leftarrow V_Z(x), \mathit{not}\ T_Z(x).
\end{aligned}$$

Let "␣" be an additional constant representing a blank symbol. We use a 6-ary relation Clause and for each clause $C_i = L_{i,1} \vee L_{i,2} \vee L_{i,3}$ in Φ we add the following fact to our program $\mathcal{P}'$:

$$\mathsf{Clause}(\sigma_p(L_{i,1}), \sigma_n(L_{i,1}), \sigma_p(L_{i,2}), \sigma_n(L_{i,2}), \sigma_p(L_{i,3}), \sigma_n(L_{i,3})),$$

where, given a literal l, $\sigma_p(l)$ and $\sigma_n(l)$ are defined as:

$$\sigma_p(l) = \begin{cases} c_i^\alpha, & \text{if } l = \alpha_i, 1 \le i \le n_\alpha \\ \text{␣}, & \text{if } l = \neg\alpha_i, 1, \le i \le n_\alpha \end{cases} \qquad \sigma_n(l) = \begin{cases} \text{␣}, & \text{if } l = \alpha_i, 1 \le i \le n_\alpha \\ c_i^\alpha, & \text{if } l = \neg\alpha_i, 1 \le i \le n_\alpha \end{cases}$$

where $\alpha = X, Y, Z$, $n_X = n, n_Y = m$, and $n_Z = k$.

We add the following rules for succinctness:

$$T(x) \leftarrow T_X(x), \qquad T(x) \leftarrow T_Y(x), \qquad T(x) \leftarrow T_Z(x),$$
$$F(x) \leftarrow F_X(x), \qquad F(x) \leftarrow F_Y(x), \qquad F(x) \leftarrow F_Z(x).$$

Next, we add the rules that ensure that $\mathcal{P}'$ only admits models that define an assignment to Boolean variables that satisfies each clause in the program:

$$\begin{aligned}
&\mathsf{violation} \leftarrow \mathsf{Clause}(x, _, y, _, z, _), F(x), F(y), F(z),\\
&\mathsf{violation} \leftarrow \mathsf{Clause}(_, x, y, _, z, _), T(x), F(y), F(z),\\
&\mathsf{violation} \leftarrow \mathsf{Clause}(x, _, y, _, _, z), F(x), F(y), T(z),\\
&\mathsf{violation} \leftarrow \mathsf{Clause}(_, x, y, _, _, z), T(x), F(y), T(z),\\
&\mathsf{violation} \leftarrow \mathsf{Clause}(x, _, _, y, z, _), F(x), T(y), F(z),\\
&\mathsf{violation} \leftarrow \mathsf{Clause}(_, x, _, y, z, _), T(x), T(y), F(z),\\
&\mathsf{violation} \leftarrow \mathsf{Clause}(x, _, _, y, _, z), F(x), T(y), T(z),\\
&\mathsf{violation} \leftarrow \mathsf{Clause}(_, x, _, y, _, z), T(x), T(y), T(z).
\end{aligned}$$

As we use this reduction to show that answer set existence of RLPs is Σ_3^P-hard in data complexity for DL-Lite$_\mathcal{F}$ (and $\mathcal{ALCHI}$), we also need to write the theory as a DL-Lite$_\mathcal{F}$ TBox. We do something stronger and let our theory be empty, i.e. $\mathcal{T}' = \top$. For this reason, we add additional rules that help us filter out those truth-value assignments to $Y_1, \ldots, Y_m$ that are not well-defined. Namely, if we get a model of a theory that defines such an assignment, we simply justify it:

$$\mathsf{ok} \leftarrow V_Y(x), F_Y(x), T_Y(x), \qquad \mathsf{ok} \leftarrow V_Y(x), \mathit{not}\ F_Y(x), \mathit{not}\ T_Y(x).$$

Finally, we add:

$$\mathsf{ok} \leftarrow \mathit{not}\ \mathsf{violation}, \qquad \leftarrow \mathit{not}\ \mathsf{ok}.$$

It is easy to verify that the RLP

$$\Pi' = (\mathcal{P}', \mathcal{T}', \{V_Y, T_X, F_X\}, \{T_Y, F_Y\}, \{T_Z, F_Z, V_Z, V_X, \mathsf{ok}, \mathsf{violation}, \mathsf{Clause}\})$$

has an answer set if and only if Φ is satisfiable.

7.4 RLPs Discussion

In this chapter, we have formalized and studied resilient logic programs as a novel hybrid language that addresses some of the shortcomings of the previous world-centric and entailment-centric approaches. There are several directions for future research. First, it is important to identify syntactic restrictions to lower the complexity of reasoning. The use of stratified negation for response predicates seems to be a promising approach, which we believe will allow us to avoid an NP oracle for checking the existence of responses in the computation of answer sets (roughly, eliminating the third-level NP oracle in the complexity results of this chapter; see Theorem 7.2.2 and Table 7.1). The second direction is to study and implement various rewritings of DL-based RLPs into disjunctive Datalog,

which has efficient reasoners [LPF+06]. The presented translation of the fragment of RLPs with theories consisting of positive disjunctive rules provides the groundwork for this.

We note that, in this chapter, FO theories and DL ontologies were interpreted over Herbrand interpretations (whose domain is a fixed infinite set of constants). This was done for mathematical clarity of the semantics of RLPs, but it has some side-effects, e.g., ruling out ontology models with a finite domain. This can be easily fixed using a more complicated definition that handles two kinds of interpretations: Herbrand interpretations for rules, and ordinary first-order structures for DL ontologies.

CHAPTER 8

Summary and Conclusion

In this concluding chapter, we provide a brief summary of our results and discuss some interesting questions that still remain open and that we would like to tackle in our future work. The main goal of this thesis was to study two particular ways of mixing the open and the closed world assumption in description logics: (i) the one that involves extending standard DL with closed predicates and (ii) the one that combines standard DLs with non-monotonic rules.

In the closed predicate setting, our focus was on very expressive DLs that extend the basic $\mathcal{ALC}$ with nominals, inverses, and number restrictions, a combination that leads to the loss of convenient model-theoretic properties and for which only very few results are known. In fact, we have already remarked that the only complexity-optimal technique for reasoning in the presence of these constructors is the one in [PH05, PH09] which is based on characterizing the satisfiability problem of knowledge bases expressed in this logic as a system of integer linear inequalities. However, this technique does not take into account closed predicates. Indeed, due to the observations made in [SFdB09], if we are only interested in the combined complexity of reasoning tasks, we can simply hardcode the extensions of closed predicates as given in the ABox directly into the TBox using disjunctions of nominals. Unfortunately, this technique is not suitable for establishing data complexity, and adapting it to the closed predicate setting is a non-trivial task. To this end, in Chapter 3 we presented a new characterization of the knowledge base satisfiability problem for $\mathcal{ALCHOIQ}$ that is able to handle closed predicates without internalizing parts of the ABox. More precisely, we used *tiles*, also known as *star-types* in the DL literature, which generalize the notion of the *type* that is commonly used to design decision procedures in DLs. In addition to the unary type of the central element, a tile τ for a particular $\mathcal{ALCHOIQ}$ KB with closed predicates additionally keeps track of the part of the neighborhood of this domain element that is relevant to the satisfaction of number restrictions. We subjected tiles to certain syntactic conditions that ensure they are of "manageable" size and encode descriptions of domain elements that are

locally consistent with respect to the TBox. Moreover, we also defined *global consistency* conditions formalized in the notion of *mosaics* for $\mathcal{K}$. These conditions are given in terms of integer linear inequalities and implications that tell us how many instances of each tile for $\mathcal{K}$ we need to build a model of $\mathcal{K}$. Finally, we showed that the existence of mosaics can be decided by using common techniques to solve the induced system of integer linear inequalities with implications (a.k.a., enriched system), allowing the transfer of known complexity results from the realm of integer programming. In particular, we observed that the size of the enriched system that we obtain from $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$ is bounded by a function that grows exponentially in the size of $\mathcal{K}$, but only polynomially in the size of $\mathcal{A}$, i.e., if the size of $\mathcal{T}$ and Σ are considered constant. As a result, we were able to infer NP-completeness of KB satisfiability in $\mathcal{ALCHOIQ}$ with closed predicates. We concluded Chapter 3 with another version of the presented characterization that was specifically tailored for the weaker logic $\mathcal{ALCHOIF}$, and which serves as the basis for Chapter 5.

In Chapter 4 we presented a novel rewriting of *safe-range queries* mediated by $\mathcal{ALCHOIQ}$ ontologies with closed predicates into $\mathsf{Datalog}^{\neg}$. In a nutshell, safe-range queries are first-order queries in which all quantified variables are guarded by closed predicates, which ensures that in a model $\mathcal{I}$ of the considered knowledge base, such queries can only map into the part of $\mathcal{I}$ that relates to named individuals of this knowledge base. Safe-range queries subsume simple instance queries as well as quantifier-free unions of conjunctive queries that were recently studied in [LSW19]. The main idea behind our rewriting was to encode the satisfiability procedure presented in Chapter 3 in $\mathsf{Datalog}^{\neg}$ by defining a modular program that first computes a relational representation of the enriched system for a given knowledge base and then solves it. An important property of our rewriting is that it is (i) data-independent – the computed program depends only on the given TBox $\mathcal{T}$ and the set Σ of closed predicates whereas the ABox is considered an input to the program and (ii) it is succinct, i.e, the size of the program is only polynomial in the size of $\mathcal{T}$ and Σ. We remark that achieving succinctness was a challenging task. Namely, the part of the program that solves the computed enriched system is a guess-and-check procedure that guesses a solution to the system and checks whether it is valid. However, we are dealing with very large systems of integer linear inequalities, namely exponential in the size of $\mathcal{T}$ and Σ, whose solutions might encode structures whose domains are doubly exponential in the size of the given ontology, requiring great care to ensure that we keep within our desired bounds. As a by-product of our rewriting we obtained a novel coNP upper data complexity bound for answering safe-range queries mediated by $\mathcal{ALCHOIQ}$ ontologies with closed predicates. This bound is worst-case optimal.

We continued exploring expressiveness of OMQ formulated in these very expressive DLs extended with closed predicates in Chapter 5, however from a slightly different perspective. Here, we were interested in finding an OMQ language that is expressive enough to capture the class of all generic Boolean queries over ABoxes that are computable in coNP. To this end, we presented inexpressibility results for OMQs based on standard DLs (i.e., without closed predicates) as well as for instance queries mediated by $\mathcal{ALCHOI}$ ontologies with

closed predicates. We then presented an extension of $\mathcal{ALCHOIF}$ with closed predicates with *nominal schemata* [KMKH11] of very restricted shapes and showed that there is no complexity increase for the problem of knowledge base satisfiability compared to $\mathcal{ALCHOIQ}$. This was done by modifying the integer programming characterization from Chapter 3. Encouraged by this result, we proceeded to show that the OMQ language that couples inconsistency queries (i.e., queries that ask whether a KB has a model) with ontologies in this extended language have the desired expressive power by showing that such queries can express computations of non-deterministic Turing machines over (encoding of) ABoxes that run in polynomial time.

Briefly stepping away from expressiveness analysis, in Chapter 6 we tackled the question of whether closed predicates, whose extensions are by definition bounded in size, allow us to infer some bounds on the number of domain elements in extensions of open predicates. To this end, we proposed two notions of bounded predicates, together with worst-case optimal procedures for deciding predicate boundedness, and concrete upper bounds on the size of their extensions. These results were then used to relax the syntactic restriction in the definition of safe-range queries and as well as to introduce a new safety condition for combining $\mathcal{ALCHOIQ}$ knowledge bases with Datalog rules.

Finally, in Chapter 7 we proposed a novel hybrid formalism called *resilient logic programs* for combining ontologies (expressed in first-order logic or a description logic) and non-monotonic rules. The main motivation behind this formalism is to provide reasoning support for systems that need to react correctly in all situations. In a nutshell, RLPs are hybrid knowledge bases consisting of a theory component whose models represent potential scenarios that the system might encounter and a rule component that represents the actual system, i.e., reactions to given situations. RLPs assume that there is a set of parameters that we can control that somehow influences the set of potential scenarios (i.e., output predicates). The semantics of RLPs then has the exist–forall–exists structure. Namely, we are interested in those structures over output predicates for which the following holds: no matter how this structure is extended over the open predicates, there will be a way to further extend it into a model of the rule component. We then proceeded in a similar fashion as in the previous chapters, analysing complexity and expressiveness of our new formalism. Namely, we showed that RLPs are decidable under some natural assumptions. We considered the setting where the theory is expressed using a few concrete first-logic fragments and we provided algorithms and complexity results. We also investigated how expressive DLs are compared to Datalog and its extensions by showing that RLPs can express $\mathsf{Datalog}^{\vee,\neg}$ even when their theory component is empty as well as identifying a fragment of RLP that can be embedded back into $\mathsf{Datalog}^{\vee,\neg}$.

Open Problems & Future work Throughout the thesis, we discussed some open questions that we would like to tackle in the future. We next give a brief summary:

Unary vs binary encoding of integers.

As $\mathcal{ALCHOIQ}$ TBoxes may contain integers greater than one, the choice of how we encode integers becomes relevant. Specifically, the integer n in some number restriction

of the form $\leq nr.C$ or $\geq nr.C$ can be represented using either unary or binary encoding, and the latter results in TBoxes that are exponentially shorter. The characterization in [PH09] is invariant to the choice of number encoding. Unfortunately, ours is not. Recall that our satisfiability characterization is based on the notion of tiles $\tau = (T, \rho)$, consisting of a central type T and an encoding of the neighborhood ρ. Due to the fact that our tiles explicitly store n different neighbors in ρ, for every applicable number restriction that requires the existence of n neighbors, the number of elements in ρ depends polynomially on the value of the numbers in the TBox. Thus, if the binary encoding of integers is assumed, the Datalog$^\neg$ translation obtained in Chapter 4 is no longer polynomial.

Richer query languages in $\mathcal{ALCHOIF}$ and $\mathcal{ALCHOIQ}$.

Characterizing the complexity of answering conjunctive queries is a long-standing open problem in expressive description logics that at the same time admit nominals, inverses, and number restrictions. So far, we only know that the problem of CQ answering is decidable for $\mathcal{ALCOIQ}$ [RG10] and that it is co-N2ExpTime-hard [GKL11]. Unfortunately, the proof in [RG10] does not provide any upper bounds on the complexity of the problem. While we obtain favorable complexity results for a large class of first-order queries, we are still interested in closing this major remaining gap.

Expressive power of $\mathcal{ALCHOIF}$ and $\mathcal{ELHIF}$.

In Chapter 5 we showed that adding certain features to $\mathcal{ALCHOIF}$ with closed predicates makes inconsistency queries in this logic powerful enough to express all generic Boolean queries over ABoxes that are computable in coNP. However, it remains unclear whether these additional features are in fact necessary, in other words, it still needs to be investigated whether basic $\mathcal{ALCHOIF}$ with closed predicates can express all coNP-computable queries. It is worth noting that the argument presented in Theorem 5.2.3 cannot be directly applied here as it relies on the existence of an algorithm that determines whether a given $\mathcal{ALCHOI}$ KB is satisfiable in the time bound that is polynomial in the size of the ABox (if the TBox and the set of closed predicates are considered fixed) and the degree of the polynomial is a constant that does not depend on the knowledge base in any way. However, in our satisfiability procedure for $\mathcal{ALCHOIQ}$ with closed predicates, the degree of the polynomial actually depends on the size of the TBox.

Another open question that remains is whether we can precisely capture coNP using an OMQ language that uses a less expressive DL but a more expressive query language, specifically, conjunctive queries and $\mathcal{ALCHOI}$ with closed predicates. An important first step towards investigating the expressive power of this language would be to characterize its data complexity, as there are currently no known upper bounds.

Finally, we believe that the arguments in Chapter 5 can also be extended to standard Horn Description logics without closed predicates. For example, the OMQ language that combines inconsistency and instance queries with $\mathcal{ELHIF}$ ontologies is proven to be PTime-hard, yet it fails to express all queries computable in PTime. We believe that simply adding a built-in linear order to $\mathcal{ELHIF}$ is insufficient to capture PTime. However, we suspect that incorporating additional features similar to those of $\mathcal{ALCHOIF}^+$ leads

to a language that is still in PTime but is also powerful enough to express all PTime-computable queries. This, however, requires further investigation and is left for future work.

Using boundedness for verification.

We are confident that the boundedness of predicates can be exploited for the verification of temporal properties of graph databases that evolve over time. The major challenge here is that data-manipulating actions may introduce fresh values into the database, potentially causing the database to grow indefinitely over time. This means that verification needs to be done on an infinite-state transition system, which is known to be undecidable even for very simple action languages and temporal properties. It was shown in [BHCDG$^+$13] that this problem becomes decidable if one is able to obtain a global upper bound on the number of elements in the active domain of the database during its evolution. Transition systems with this property are referred to as *state-bounded systems*, however, recognizing whether a system is state-bounded is generally undecidable. We remark that by exploiting the results presented in Chapter 6, we can identify some cases in which state boundedness is ensured. More specifically, consider the setting in which we are given a graph database $\mathcal{G}$ with integrity constraints expressed as a TBox $\mathcal{T}$ and a set Σ of relations that are read-only. Assuming that the extension of the predicates in Σ is either fixed for the whole evolution or allowed to vary while obeying some predefined bound on its size and that all the remaining relations are strongly bounded w.r.t. $\mathcal{T}$ and Σ, we obtain state boundedness of the corresponding transition system and we can transfer the results from [BHCDG$^+$13]. We plan to make this more precise in our future work.

Implementation of RLPs.

One drawback of RLPs is that they suffer from quite high complexity which makes it important to identify syntactic restrictions for which the computational cost of reasoning is lower. Allowing only stratified negation for response predicates seems to be a good candidate – we believe this will allow us to avoid an NP oracle for checking the existence of responses in the computation of answer sets (roughly, eliminating the third-level NP oracle; see Theorem 7.2.2 and Table 7.1).

Another problem that we leave for future work is studying and implementing various rewritings of DL-based RLPs into Datalog$^{\vee,\neg}$, which has efficient reasoners [LPF$^+$06]. The presented translation of the fragment of RLPs with theories consisting of positive disjunctive rules provides the groundwork for this.

List of Figures

List of Tables

List of Algorithms

Bibliography

[AB09] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[ABW88] Krzysztof R. Apt, Howard A. Blair, and Adrian Walker. Chapter 2 - towards a theory of declarative knowledge. In *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufmann, 1988.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[ALMV18] Giovanni Amendola, Nicola Leone, Marco Manna, and Pierfrancesco Veltri. Enhancing existential rules by closed-world variables. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence, IJCAI 2018*, pages 1676–1682. ijcai.org, 2018.

[AOŠ20] Shqiponja Ahmetaj, Magdalena Ortiz, and Mantas Šimkus. Polynomial rewritings from expressive description logics with closed predicates to variants of datalog. *Artificial Intelligence*, 280:103220, 2020.

[AU79] Alfred V Aho and Jeffrey D Ullman. Universality of data retrieval languages. In *Proceedings of the 6th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 110–119, 1979.

[BBL05] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the EL envelope. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence, IJCAI 2005*, pages 364–369. Professional Book Center, 2005.

[BBR20] Bartosz Bednarczyk, Franz Baader, and Sebastian Rudolph. Satisfiability and query answering in description logics with global and local cardinality constraints. In *Proceedings of ECAI 2020*, June 2020.

[BBtCP16] Michael Benedikt, Pierre Bourhis, Balder ten Cate, and Gabriele Puppis. Querying visible and invisible information. In *Proceedings of LICS 2016*, pages 297–306. ACM, 2016.

[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, 2003.

[BCS15] Vince Bárány, Balder Ten Cate, and Luc Segoufin. Guarded negation. *Journal of the ACM (JACM)*, 62(3):22:1–22:26, June 2015.

[BGG97] Egon Börger, Erich Grädel, and Yuri Gurevich. *The Classical Decision Problem*. Perspectives in Mathematical Logic. Springer, 1997.

[BHCDG+13] Babak Bagheri Hariri, Diego Calvanese, Giuseppe De Giacomo, Alin Deutsch, and Marco Montali. Verification of relational data-centric dynamic systems with external services. In *Proceedings of PODS 2013*. ACM, 2013.

[BHLS17] Franz Baader, Ian Horrocks, Carsten Lutz, and Uli Sattler. *An Introduction to Description Logic*. Cambridge University Press, 2017.

[BKK+18] Meghyn Bienvenu, Stanislav Kikot, Roman Kontchakov, Vladimir V. Podolskii, and Michael Zakharyaschev. Ontology-mediated queries: Combined complexity and succinctness of rewritings via circuit complexity. *Journal of the (JACM)*, 65(5), aug 2018.

[BLW09] Piero A. Bonatti, Carsten Lutz, and Frank Wolter. The complexity of circumscription in description logics. *Journal of Artificial Intelligence Research*, 35:717–773, 2009.

[BO15] Meghyn Bienvenu and Magdalena Ortiz. Ontology-mediated query answering with data-tractable description logics. In *Proceedings of Reasoning Web Summer School 2015*, volume 9203 of *LNCS*, pages 218–307. Springer, 2015.

[Bor96] Alex Borgida. On the relative expressiveness of description logics and predicate logics. *Artificial Intelligence*, 82(12):353 – 367, 1996.

[BOŠ18] Labinot Bajraktari, Magdalena Ortiz, and Mantas Šimkus. Combining rules and ontologies into Clopen knowledge bases. In *Proceedings of AAAI 2018*, 2018.

[Bra79] Ronald J Brachman. On the epistemological status of semantic networks. In *Associative networks*, pages 3–50. Elsevier, 1979.

[BtCLW14] Meghyn Bienvenu, Balder ten Cate, Carsten Lutz, and Frank Wolter. Ontology-based data access: A study through disjunctive datalog, CSP, and MMSNP. *ACM Transactions on Database Systems*, 39(4):33:1–33:44, 2014.

[Cal96] Diego Calvanese. Finite model reasoning in description logics. In *Proceedings of the 5th International Conference on the Principles of Knowledge Representation and Reasoning, KR 1996*, pages 292–303. Morgan Kaufmann, 1996.

[CDK18] David Carral, Irina Dragoste, and Markus Krötzsch. The combined approach to query answering in horn-alchoiq. In *Proceedings of the 16th International Conference on Principles of Knowledge Representation and Reasoning, KR 2018*, pages 339–348, 2018.

[CDL+05] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. DL-Lite: Tractable description logics for ontologies. In *Proceedings of the 20th National Conference on Artificial Intelligence and the 17th Innovative Applications of Artificial Intelligence Conference*, pages 602–607. AAAI Press / The MIT Press, 2005.

[CDL+07] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *Journal of Automated Reasoning*, 39(3):385–429, 2007.

[CGK13] Andrea Calì, Georg Gottlob, and Michael Kifer. Taming the infinite chase: Query answering under expressive relational constraints. *Journal of Artificial Intelligence Research (JAIR)*, 48:115–174, 2013.

[CGL+13] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Data complexity of query answering in description logics. *Artificial Intelligence*, 195:335–360, 2013.

[CLN94] Diego Calvanese, Maurizio Lenzerini, and Daniele Nardi. A unified framework for class-based representation formalisms. In *Principles of Knowledge Representation and Reasoning*, pages 109–120. Elsevier, 1994.

[CMET14] Diego Calvanese, Marco Montali, Montserrat Estañol, and Ernest Teniente. Verifiable UML artifact-centric business process models. In *Proceedings of CIKM 2014*, pages 1289–1298. ACM, 2014.

[Cod70] E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, june 1970.

[Coo72] Stephen A Cook. A hierarchy for nondeterministic time complexity. In *Proceedings of the 4th annual ACM symposium on Theory of computing*, pages 187–192, 1972.

[CTS11] Alexandros Chortaras, Despoina Trivela, and Giorgos Stamou. Optimized query rewriting for owl 2 ql. In *Automated Deduction – CADE-23*, pages 192–206, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[DEGV01] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. *ACM Computing Surveys*, 33(3):374–425, 2001.

[DNR02] Francesco M. Donini, Daniele Nardi, and Riccardo Rosati. Description logics of minimal knowledge and negation as failure. *ACM Transactions on Computational Logic*, 3(2):177–225, 2002.

[EGM97] Thomas Eiter, Georg Gottlob, and Heikki Mannila. Disjunctive Datalog. *ACM Transactions on Database Systems (TODS)*, 22(3):364–418, 1997.

[EIK09] Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. Answer set programming: A primer. In *Reasoning Web*, volume 5689 of *LNCS*, pages 40–110. Springer, 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. *Artificial Intelligence*, 172(12-13):p. 1495, 2008.

[ELOŠ09] Thomas Eiter, Carsten Lutz, Magdalena Ortiz, and Mantas Šimkus. Query answering in description logics with transitive roles. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI 2009*, pages 759–764, 2009.

[EOŠ12] Thomas Eiter, Magdalena Ortiz, and Mantas Šimkus. Conjunctive query answering in the description logic SH using knots. *J. Comput. Systems Sci.*, 78(1):47–85, 2012.

[EP03] Thomas Eiter and Axel Polleres. Transforming co-np checks to answer set computation by meta-interpretation. In *Informal Proceedings of Joint Conference on Declarative Programming (AGP 2003)*, pages 410–421, 2003.

[FCS+15] Cristina Feier, David Carral, Giorgio Stefanoni, Bernardo Cuenca Grau, and Ian Horrocks. The combined approach to query answering beyond the owl 2 profiles. In *24th International Joint Conference on Artificial Intelligence*, 2015.

[FIS11] Enrico Franconi, Yazmín Angélica Ibáñez-García, and Inanç Seylan. Query answering with dboxes is hard. *Electronic Notes in Theoretical Computer Science*, 278:71–84, 2011.

[Fit96] Melvin Fitting. *First-order logic and automated theorem proving (2nd ed.)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1996.

[Flo62] Robert W. Floyd. Algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345, jun 1962.

[GBIGKK15] Víctor Gutiérrez-Basulto, Yazmín Ibáñez-García, Roman Kontchakov, and Egor V Kostylev. Queries with negation and inequalities over lightweight ontologies. *Journal of Web Semantics*, 35:184–202, 2015.

[GGBIG+19] Tomasz Gogacz, Víctor Gutiérrez-Basulto, Yazmín A Ibáñez-García, Filip Murlak, Magdalena Ortiz, and Mantas Šimkus. Ontology focusing: Knowledge-enriched databases on demand. *arXiv preprint arXiv:1904.00195*, 2019.

[GGI+20] Tomasz Gogacz, Víctor Gutiérrez-Basulto, Yazmín Ibáñez-García, Filip Murlak, Magdalena Ortiz, and Mantas Šimkus. Ontology focusing: Knowledge-enriched databases on demand. In *Proceedings of ECAI 2020*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 745–752. IOS Press, 2020.

[GKK+14] Georg Gottlob, Stanislav Kikot, Roman Kontchakov, Vladimir V. Podolskii, Thomas Schwentick, and Michael Zakharyaschev. The price of query rewriting in ontology-based data access. *Artificial Intelligence*, 213:42–59, 2014.

[GKL11] Birte Glimm, Yevgeny Kazakov, and Carsten Lutz. Status QIO: an update. In *Proceedings of the 24th International Workshop on Description Logics, DL 2011,*, volume 745 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2011.

[GL88] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Proceedings of ICLP/SLP 1988*, pages 1070–1080. MIT Press, 1988.

[GLOŠ20] Tomasz Gogacz, Sanja Lukumbuzya, Magdalena Ortiz, and Mantas Šimkus. Datalog rewritability and data complexity of ALCHOIF with closed predicates. In *Proceedings the 17th International Conference on the Principles of Knowledge Representation and Reasoning of KR 2020*, pages 434–444, 2020.

[GS12] Georg Gottlob and Thomas Schwentick. Rewriting ontological queries into small nonrecursive datalog programs. In *Proceedings of the 13th International Conference on the Principles of Knowledge Representation and Reasoning, KR 2012*. AAAI Press, 2012.

[HLSW15] Peter Hansen, Carsten Lutz, Inanç Seylan, and Frank Wolter. Efficient query rewriting in the description logic EL and beyond. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence, IJCAI 2015*, pages 3034–3040. AAAI Press, 2015.

[HMS07] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Reasoning in description logics by a reduction to disjunctive datalog. *Journal of Automated Reasoning*, 39(3):351–384, 2007.

[Hor08] Ian Horrocks. Ontologies and the semantic web. *Communications of the ACM*, 51(12):58–67, 2008.

[Imm99] Neil Immerman. *Descriptive complexity*. Graduate texts in computer science. Springer, 1999.

[KAH11] Matthias Knorr, José Júlio Alferes, and Pascal Hitzler. Local closed world reasoning with description logics under the well-founded semantics. *Artificial Intelligence*, 175(9-10):1528–1554, 2011.

[KHJR+15] Evgeny Kharlamov, Dag Hovland, Ernesto Jiménez-Ruiz, Davide Lanti, Hallstein Lie, Christoph Pinkel, Martin Rezk, Martin G. Skjæveland, Evgenij Thorstensen, Guohui Xiao, Dmitriy Zheleznyakov, and Ian Horrocks. Ontology based access to exploration data at statoil. In *The Semantic Web - ISWC 2015*, pages 93–112, Cham, 2015. Springer International Publishing.

[KM78] Ravindran Kannan and Clyde L Monma. On the computational complexity of integer programming problems. In *Optimization and Operations Research: Proceedings of a Workshop Held at the University of Bonn, October 2–8, 1977*, pages 161–172. Springer, 1978.

[KMKH11] Markus Krötzsch, Frederick Maier, Adila Krisnadhi, and Pascal Hitzler. A better uncle for OWL: nominal schemas for integrating rules and ontologies. In *Proceedings of the 20th International Conference on World Wide Web, WWW 2011*, pages 645–654. ACM, 2011.

[Kol91] Phokion G Kolaitis. The expressive power of stratified logic programs. *Information and Computation*, 90(1):50–66, 1991.

[KR14] Markus Krötzsch and Sebastian Rudolph. Nominal schemas in description logics: Complexities clarified. In *Principles of Knowledge Representation and Reasoning: Proceedings of the 14th International Conference, KR 2014*. AAAI Press, 2014.

[Lif91] Vladimir Lifschitz. Nonmonotonic databases and epistemic queries. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI 91, page 381–386. Morgan Kaufmann Publishers Inc., 1991.

[LOŠ20] Sanja Lukumbuzya, Magdalena Ortiz, and Mantas Šimkus. Resilient logic programs: Answer set programs challenged by ontologies. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence, AAAI 2020*, pages 2917–2924. AAAI Press, 2020.

[LOŠ23] Sanja Lukumbuzya, Magdalena Ortiz, and Mantas Šimkus. On the expressive power of ontology-mediated queries: Capturing conp. In *Proceedings of the 36th International Workshop on Description Logics, DL 2023*, volume 3515 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2023.

[LOŠ24] Sanja Lukumbuzya, Magdalena Ortiz, and Mantas Šimkus. Datalog rewritability and data complexity of alchoiq with closed predicates. *Artificial Intelligence*, 330:104099, 2024.

[LPF+06] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The dlv system for knowledge representation and reasoning. *ACM Transactions on Computational Logic (TOCL)*, 7(3):499–562, 2006.

[LR98] Alon Y Levy and Marie-Christine Rousset. Combining horn rules and description logics in carin. *Artificial intelligence*, 104(1-2):165–209, 1998.

[LŠ21] Sanja Lukumbuzya and Mantas Šimkus. Bounded predicates in description logics with counting. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence, IJCAI 2021*, pages 1966–1972. ijcai.org, 2021.

[LST05] Carsten Lutz, Ulrike Sattler, and Lidia Tendera. The complexity of finite model reasoning in description logics. *Information and Computation*, 199(1-2):132–171, 2005.

[LSW13] Carsten Lutz, Inanç Seylan, and Frank Wolter. Ontology-based data access with closed predicates is inherently intractable(sometimes). In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence, IJCAI 2013*, pages 1024–1030. IJCAI/AAAI, 2013.

[LSW15] Carsten Lutz, Inanç Seylan, and Frank Wolter. Ontology-mediated queries with closed predicates. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence, IJCAI 2015*, pages 3120–3126. AAAI Press, 2015.

[LSW19] Carsten Lutz, Inanç Seylan, and Frank Wolter. The data complexity of ontology-mediated queries with closed predicates. *Logical Methods in Computer Science*, 15(3), 2019.

[LTW09] Carsten Lutz, David Toman, and Frank Wolter. Conjunctive query answering in the description logic el using a relational database system. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI 2009*, volume 9, pages 2070–2075, 2009.

[Lut08] Carsten Lutz. The complexity of conjunctive query answering in expressive description logics. In *Proceedings of Automated Reasoning, 4th*

International Joint Conference, IJCAR 2008, volume 5195 of *LNCS*, pages 179–193. Springer, 2008.

[Mar06] David Marker. *Model theory: an introduction*, volume 217. Springer Science & Business Media, 2006.

[MC16] Marco Montali and Diego Calvanese. Soundness of data-aware, case-centric processes. *International Journal on Software Tools for Technology Transfer*, 18(5):535–558, 2016.

[MHRS06] Boris Motik, Ian Horrocks, Riccardo Rosati, and Ulrike Sattler. Can OWL and logic programming live together happily ever after? In *Proceedings of the Semantic Web - ISWC 2006, 5th International Semantic Web Conference, ISWC 2006*, volume 4273 of *Lecture Notes in Computer Science*, pages 501–514. Springer, 2006.

[Min74] Marvin Minsky. A framework for representing knowledge, 1974.

[MR07] Boris Motik and Riccardo Rosati. A faithful integration of description logics with logic programming. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI 2007*, pages 477–482, 2007.

[MR10] Boris Motik and Riccardo Rosati. Reconciling description logics and rules. *Journal of the ACM (JACM)*, 57(5):30:1–30:62, 2010.

[MSS05] Boris Motik, Ulrike Sattler, and Rudi Studer. Query answering for OWL-DL with rules. *Journal of Web Semantics*, 3(1):41–60, 2005.

[MTKW18] David Maier, K. Tuncay Tekle, Michael Kifer, and David Warren. *Datalog: concepts, history, and outlook*, pages 3–100. Association for Computing Machinery and Morgan & Claypool, 09 2018.

[NOŠ16] Nhung Ngo, Magdalena Ortiz, and Mantas Šimkus. Closed predicates in description logics: Results on combined complexity. In *Proceedings of the 15th International Conference on the Principles of Knowledge Representation and Reasoning, KR 2016*, pages 237–246, 2016.

[OPŠ19] Magdalena Ortiz, Sanja Pavlovic, and Mantas Šimkus. Answer set programs challenged by ontologies. In *Proceedings of the 32nd International Workshop on Description Logics, DL 2019*, volume 2373 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2019.

[ORŠ10] Magdalena Ortiz, Sebastian Rudolph, and Mantas Šimkus. Worst-case optimal reasoning for the Horn-DL fragments of OWL 1 and 2. In *Proceedings of the 12th International Conference on the Principles of Knowledge Representation and Reasoning, KR 2010*. AAAI Press, 2010.

[OWL09] W3C OWL Working Group. *OWL 2 Web Ontology Language: Document Overview*. W3C Recommendation, 27 October 2009. Available at `http://www.w3.org/TR/owl2-overview/`.

[Pap81] Christos H Papadimitriou. On the complexity of integer programming. *Journal of the ACM (JACM)*, 28(4):765–768, 1981.

[Pap94] Christos H. Papadimitriou. *Computational Complexity*. Addison Wesley Publ. Co., 1994.

[PH05] Ian Pratt-Hartmann. Complexity of the two-variable fragment with counting quantifiers. *Journal of Logic, Language and Information*, 14(3):369–395, 2005.

[PH09] Ian Pratt-Hartmann. Data-complexity of the two-variable fragment with counting quantifiers. *Information and Computation*, 207(8):867–888, 2009.

[PLC+08] Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Linking data to ontologies. *Journal on Data Semantics*, 10:133–173, 2008.

[Qui67] M Ross Quillian. Word concepts: A theory and simulation of some basic semantic capabilities. *Behavioral science*, 12(5):410–430, 1967.

[RA10] Riccardo Rosati and Alessandro Almatelli. Improving query answering over DL-Lite ontologies. In *Principles of Knowledge Representation and Reasoning: Proceedings of the 12th International Conference, KR 2010*. AAAI Press, 2010.

[RG10] Sebastian Rudolph and Birte Glimm. Nominals, inverses, counting, and conjunctive queries or: Why infinity is your friend! *Journal of Artificial Intelligence Research*, 39:429–481, 2010.

[RMKZ13] Mariano Rodrıguez-Muro, Roman Kontchakov, and Michael Zakharyaschev. Ontop at work. In *Proceedings of the 10th OWL: Experiences and Directions Workshop (OWLED 2013)*, 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 *Proceedings of 10th International Conference on the Principles of Knowledge Representation and Reasoning, KR 2006*. AAAI Press, 2006.

[Ros07a] Riccardo Rosati. The limits of querying ontologies. In *Proceedings of the Database Theory - ICDT 2007, 11th International Conference*, pages 164–178. Springer, 2007.

[Ros07b] Riccardo Rosati. On conjunctive query answering in EL. In *Proceedings of the 2007 International Workshop on Description Logics, DL 2007*, volume 250 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[Ros12] Riccardo Rosati. Query rewriting under extensional constraints in DL-Lite. In *Proceedings of the 2012 International Workshop on Description Logics, DL-2012, Rome, Italy, June 7-10, 2012*, volume 846 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2012.

[SA77] Roger C Schank and Robert P Abelson. *Scripts, plans, goals, and understanding: an inquiry into human knowledge structures*. Hillsdale, NJ: Lawrence Erlbaum Associates, 1977.

[Sav70] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970. ISSN 1439-2275.

[Sch93] Andrea Schaerf. On the complexity of the instance checking problem in concept languages with existential quantification. *Journal of Intelligent Information Systems*, 2(3):265–278, 1993.

[Sch95] JS Schlipf. The expressive powers of the logic programming semantics. *Journal of Computer and System Sciences*, 51(1):64–86, 1995.

[Sch99] Alexander Schrijver. *Theory of linear and integer programming*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1999.

[SFdB09] Inanç Seylan, Enrico Franconi, and Jos de Bruijn. Effective query rewriting with ontologies over dboxes. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI 2009*, pages 923–925, 2009.

[Sim13] František Simančík. *Consequence-based Reasoning for Ontology Classification*. PhD thesis, University of Oxford, Oxford, United Kingdom, 2013.

[SKH11] Kunal Sengupta, Adila Alfa Krisnadhi, and Pascal Hitzler. Local closed world semantics: Grounded circumscription for OWL. In *Proceedings of The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference*, pages 617–632. Springer, 2011.

[Tob00] Stephan Tobies. The complexity of reasoning with cardinality restrictions and nominals in expressive description logics. *Journal of Artificial Intelligence Research*, 12:199–217, 2000.

[Tur37] A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 1937.

[Woo75] William A Woods. What's in a link: Foundations for semantic networks. In *Representation and understanding*, pages 35–82. Elsevier, 1975.

[Žák83] Stanislav Žák. A turing machine time hierarchy. *Theoretical Computer Science*, 26(3):327–333, 1983.

Appendices

Missing Proof of Theorem 3.3.4

We next provide the proof of Theorem 3.3.4. The result states that an $\mathcal{ALCHOIF}$ KB $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$ with closed predicates is satisfiable if and only if it respects role inclusions and there exists a mosaic for $\mathcal{K}$, as given in Definition 3.3.2.

($\Rightarrow$) : Assume $\mathcal{K}$ is satisfiable. It is easy to show that $\mathcal{K}$ respects role inclusions.

1. Assume $r \in \Sigma \cap \mathsf{N_R}$ and $s(a,b) \in \mathcal{A}$. If we have $s \sqsubseteq r \in \mathcal{T}$, as $\mathcal{I} \vDash \mathcal{T}$, we have $s^{\mathcal{I}} \subseteq r^{\mathcal{I}}$. Moreover, as $\mathcal{I} \vDash_{\Sigma} \mathcal{A}$, we have $(a,b) \in s^{\mathcal{I}}$ and thus also $(a,b) \in r^{\mathcal{I}}$. Since $r \in \Sigma$, we must have $r(a,b) \in \mathcal{A}$. Similarly, if $s^- \sqsubseteq r \in \mathcal{T}$, we have that $(s^-)^{\mathcal{I}} \subseteq r$. This means that $(b,a) \in (s^-)^{\mathcal{I}}$ and thus $(b,a) \in r^{\mathcal{I}}$. Since $r \in \Sigma$, we must have $r(b,a) \in \mathcal{A}$.

2. Let r be a role with $\mathsf{func}(r) \in \mathcal{T}$ and let a be a constant in $\mathcal{A}$ and assume that $\{b : p(a,b) \in \mathcal{A}, p \sqsubseteq r \in \mathcal{T}\} \cup \{b : p(b,a) \in \mathcal{A}, p^- \sqsubseteq r \in \mathcal{T}\}$ has more than one element. It is easy to verify that there can be no interpretation that satisfies $\mathsf{func}(r)$ and so $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$ is unsatisfiable which contradicts our assumption.

We next show that we can construct a mosaic for $\mathcal{K}$ from a model $\mathcal{I}$ of $\mathcal{K}$. To this end, we extract for each element $e \in \Delta^{\mathcal{I}}$ a tile $\tau_e = (T_e, \rho_e)$ that describes it. Recall that we denote by $\mathsf{t}(e)$ the unary type of an element (i.e., the set of all basic concepts that e participates in), and we denote by $\mathsf{rt}(e,e')$ the set of all roles that the pair (e,e') participates in. We now set $T_e = \mathsf{t}(e)$. To define ρ_e we do the following. As $\mathcal{I}$ is a model of $\mathcal{K}$, $\mathcal{I}$ satisfies every existential axiom in $\mathcal{T}$. This means that for every axiom $\alpha \in \mathcal{T}$ that is of the type $A \sqsubseteq \exists r.B$, if $A \in \mathsf{t}(e)$, then there exists at least one element $e_\alpha \in \Delta^{\mathcal{I}}$ such that $(e, e_\alpha) \in r^{\mathcal{I}}$ and $e_\alpha \in B^{\mathcal{I}}$. If there are multiple choices for e_α, we pick an arbitrary one. Let $E(e) = \{e_\alpha : \alpha = A \sqsubseteq \exists r.B \in \mathcal{T}, A \in \mathsf{t}(e)\}$. Further, let $F(e) = \{e' : (e,e') \in r^{\mathcal{I}}, \mathsf{func}(r) \in \mathcal{T}\}$. We set

$$\rho_e = \{(\mathsf{rt}(e,e'), \mathsf{t}(e')) : e' \in E(e) \cup F(e)\}.$$

It is easy to verify that τ_e is a proper tile for $\mathcal{K}$.

To define a mosaic for $\mathcal{K}$, we define a function $N : \text{Tiles}(\mathcal{K}) \to \mathbb{N}^*$ as

$$N(\tau) = |\{e \in \Delta^{\mathcal{I}} : \tau_e = \tau\}|.$$

Note that if this set is infinite, we set $N(\tau) = \aleph_0$. Finally, in order to show that N is indeed a mosaic for $\mathcal{K}$, we need to show that N satisfies conditions MF1-MF5 in Definition 3.2.9.

MF1. Let a be an arbitrary constant in $\mathcal{K}$. The only domain element that participates in $\{a\}$ is the element that $\mathcal{I}$ maps a to, which is a itself. Thus, τ_a is the only tile that contains $\{a\}$ in its type, and moreover $N(\tau_a) = 1$.

MF2. As we do not allow interpretations with empty domains, there must be at at least one element $e \in \Delta^{\mathcal{I}}$. Then $N(\tau_e) \geq 1$ and thus MF2 is satisfied.

MF3. Let T, T' be two arbitrary types for $\mathcal{K}$, let R be an arbitrary subset of $\mathsf{N}_{\mathsf{R}}^{+}(\mathcal{K})$ and assume that there is a role r with $\mathsf{func}(r^-) \in \mathcal{T}$ such that $r \in R$. By construction, the number of tiles (T', ρ') where $(R^-, T) \in \rho'$ is exactly the number of pairs $(e', e) \in (r^-)^{\mathcal{I}}$, where $\mathsf{t}(e') = T'$, $\mathsf{t}(e) = T$, and $\mathsf{rt}(e', e) = R^-$. This is the value of the sum on the right-hand side of the inequality in MF3.

Consider now an element e with $\mathsf{t}(e) = T$. Note that the way we construct τ_e guarantees that each pair $(R, T') \in \rho_e$ corresponds to an element e' with $\mathsf{t}(e') = T'$ and $\mathsf{rt}(e, e') = R$. We compute the left-hand side of the inequality in MF3 as the number of tiles (T, ρ) with $(R, T') \in \rho$, which, in the light of the previous observation, can be *at most* the number of pairs $(e, e') \in r^{\mathcal{I}}$ such that $\mathsf{t}(e) = T$, $\mathsf{t}(e') = T'$, and $\mathsf{rt}(e, e') = R$. The latter is exactly the number of pairs $(e', e) \in (r^-)^{\mathcal{I}}$, where $\mathsf{t}(e') = T'$, $\mathsf{t}(e) = T$, and $\mathsf{rt}(e', e) = R^-$, which is the value of the sum of the right-hand side of the inequality. This establishes that the inequality in MF3 holds.

MF4. Assume there is a tile τ for $\mathcal{K}$ such that $N(\tau) > 0$. That means that there is at least one element $e \in \Delta^{\mathcal{I}}$ with $\tau = \tau_e$. Let (R, T') be a pair in ρ. By construction, this pair corresponds to some domain element e' with $\mathsf{t}(e') = T'$ and $\mathsf{rt}(e, e') = R$. We then have that $N(\tau_{e'}) \geq 1$, and so the inequality in MF4 holds.

MF5. Let $\{a\}$ and $\{b\}$ be two nominals in $\mathsf{N}_{\mathsf{C}}^{+}(\mathcal{K})$, $A, B \in \mathsf{N}_{\mathsf{C}}(\mathcal{K})$ and assume at least one of the conditions (a)-(d) is satisfied. In this case, it is easy to see that if $a \in A^{\mathcal{I}}$, then $b \in B^{\mathcal{I}}$. Assume that $\sum_{\substack{(T,\rho)\in\text{Tiles}(\mathcal{K}),\\ \{a\}\in T, A\in T}} N((T, \rho)) > 0$. As $\tau_a = (T_a, \rho_a)$ is the only tile that contains $\{a\}$ in its type, A must be in T_a, and so $a \in A^{\mathcal{I}}$. This means that $b \in B^{\mathcal{I}}$, and so $B \in T_b$. As τ_b is the only tile with $\{b\}$ in its type, this means that there exist no tile (T, ρ) such that $\{b\} \in T$ and $B \notin T$. Hence, MF5 holds.

We have thus shown that N is a mosaic for $\mathcal{K}$.

($\Leftarrow$) : Assume $\mathcal{K}$ respects role inclusions and let N be a mosaic for $\mathcal{K}$. We show how to construct a model $\mathcal{I}$ of $\mathcal{K}$. Intuitively, each tile can be seen as a description of a domain element and a part of its neighborhood. A mosaic for $\mathcal{K}$ tells us for each tile τ, how many domain elements that fit the description provided by τ we need in order to build a model of $\mathcal{K}$. Thus, we set our domain

$$\Delta^{\mathcal{I}} = \{\tau_i : \tau \in \text{Tiles}(\mathcal{K}) \text{ and } 0 < i \leq N(\tau)\}.$$

Intuitively, the constant τ_i corresponds to the i-th domain element described by the tile τ. Further, for every $B \in \mathsf{N}_{\mathsf{C}}^{+}(\mathcal{K}) \setminus \{\top, \bot\}$, we set

$$B^{\mathcal{I}} = \{\tau_i \in \Delta^{\mathcal{I}} : \tau = (T, \rho), B \in T, 1 \leq i \leq N(\tau)\}.$$

The extensions of roles are constructed in multiple steps.

1. We first make sure that that $\mathcal{I}$ satisfies the ABox $\mathcal{A}$. Due to MF1, for each nominal $\{a\} \in \mathsf{N}_{\mathsf{C}}^{+}\mathcal{K}$, we know that there is exactly one domain element $e_a = (T_a, \rho_a)_1$ s.t. $\{a\} \in T_a$. Hence, there is exactly one domain element participating in $\{a\}^{\mathcal{I}}$ and that is e_a. This element "represents" the constant a in $\mathcal{I}$ (and can be renamed to a to comply with SNA). Now, for each assertion $r(a, b) \in \mathcal{A}$ and each role s with $r \sqsubseteq s \in \mathcal{T}$, we set $(e_a, e_b) \in s^{\mathcal{I}}$, if s is a role name and $(e_b, e_a) \in s^{\mathcal{I}}$, otherwise.

We need to take special care when we construct the extensions of roles that are involved in functionality assertions.

2. For each role $r \in \mathsf{N}_{\mathsf{R}}^{+}(\mathcal{K})$ with $\mathsf{func}(r) \in \mathcal{T}$ and $\mathsf{func}(r^{-}) \in \mathcal{T}$ we do the following. Using the inequalities from MF3 in two directions, we have that for every pair $T, T' \in \mathsf{Types}(\mathcal{K})$ and every $R \subseteq \mathsf{N}_{\mathsf{R}}^{+}(\mathcal{K})$ with $r \in R$, the following is satisfied: Thus, the sets

$$X_{T,T',R} = \{(T, \rho)_i \in \Delta^{\mathcal{I}} : (R, T') \in \rho, 1 \leq i \leq N((T, \rho))\}$$

$$Y_{T,T',R} = \{(T', \rho')_i \in \Delta^{\mathcal{I}} : (R^{-}, T) \in \rho', 1 \leq i \leq N((T', \rho'))\}$$

have the same cardinality and so there exists a bijection $f_{T,T',R}$ between them with them.

Furthermore, the bijection $f_{T,T',R}$ has the following property for all elements $e \in X_{T,T',R}$: If r is a role name (resp. an inverse role) and $(e, e') \in r^{\mathcal{I}}$ (resp. $(e', e) \in (r^{-})^{\mathcal{I}}$) was constructed in step 1, then $f_{T,T',R}(e) = e'$.

Assume r is a role name and $(e, e') \in r^{\mathcal{I}}$ or was constructed in the first step (similarly for r an inverse role and $(e', e) \in (r^{-})^{\mathcal{I}}$). Then, e and e' correspond to some constants a and b and we must have one of the following scenarios:

a) $p(a,b) \in \mathcal{A}$, $p \sqsubseteq r \in \mathcal{T}$, or

b) $p(b,a) \in \mathcal{A}$, $p \sqsubseteq r^- \in \mathcal{T}$.

This means that $\{a\} \in T$. Due to the conditions TF8 (a) and TF8 (b), we can conclude that $\{b\} \in T'$. In the light of this, due to MF1, e is the only element in $X_{T,T',R}$ and e' is the only element in $Y_{T,T',R}$. Thus, $f_{T,T',R}(e) = e'$.

We now do the following. For every $e \in X_{T,T',R}$ and every $s \in R$, we set $(e, f_{T,T',R}(e)) \in s^{\mathcal{I}}$ if s is a role name and $(f_{T,T',R}(e), e) \in (s^-)^{\mathcal{I}}$ otherwise. Intuitively, this connects via an r-arc, every domain element of type T that requires an r-neighbor of type T' to one domain element of type T' that requires an r^--neighbor of type T.

3. Next, for every role $r \in \mathsf{N}_\mathsf{R}^+(\mathcal{K})$ such that $\mathsf{func}(r^-) \in \mathcal{T}$ but $\mathsf{func}(r^-) \notin \mathcal{T}$ we do the following. For every pair $T, T' \in \mathsf{Types}(\mathcal{K})$ and every $R \subseteq \mathsf{N}_\mathsf{R}^+(\mathcal{K})$ with $r \in R$, we get by employing MF3 that the following is satisfied: If we define the sets $X_{T,T',R}$ and $Y_{T,T',R}$ as before, we get that $|X_{T,T',R}| \leq |Y_{T,T',R}|$, for every T, T', R. Thus, there exists an injection $g_{T,T',R}$ from $X_{T,T',R}$ onto $Y_{T,T',R}$. This injection $g_{T,T',R}$ has the same property as the bijection $f_{T,T',R}$ in the previous case. Once again, for every $e \in X_{T,T',R}$ and every $s \in R$, we set $(e, g_{T,T',R}(e)) \in s^{\mathcal{I}}$ if s is a role name and $(g_{T,T',R}(e), e) \in (s^-)^{\mathcal{I}}$ otherwise. We note that, if there is a role s s.t. $\mathsf{func}(s) \in \mathcal{T}$ and $\mathsf{func}(s^-) \in \mathcal{T}$ and $s \in R$, the injection $g_{T,T',R}$ is exactly the bijection $f_{T,T',R}$ from the previous case.

4. Finally, we do the following for each $e = (T, \rho)_i \in \Delta^{\mathcal{I}}$ with $(R, T') \in \rho$ for which there is no $d \in \Delta^{\mathcal{I}}$ such that $d = (T', \rho')$ and $(e, d) \in r^{\mathcal{I}}$, for all $r \in R$. Due to MF4, there exists some ρ' such that $(T', \rho') \in \mathsf{Tiles}(\mathcal{K})$ and $N((T', \rho')) > 0$. Let e' denote the domain element $(T', \rho')_1 \in \Delta^{\mathcal{I}}$. For each $r \in R$, if $r \in \mathsf{N}_\mathsf{R}$ we set $(e, e') \in r^{\mathcal{I}}$, otherwise we set $(e', e) \in (r^-)^{\mathcal{I}}$. This concludes the construction of $\mathcal{I}$.

We next need to show that $\mathcal{I}$ is indeed a model of $\mathcal{K}$ under Σ.

Satisfaction of $\mathcal{A}$ under Σ. Let $A(c) \in \mathcal{A}$, where $A \in \mathsf{N}_\mathsf{C}$. As $\{c\} \in \mathsf{N}_\mathsf{C}^+(\mathcal{K})$, we have that $e_c \in \Delta^{\mathcal{I}}$, where, as explained above, $e_c = (T_c, \rho_c)$ is an element in $\Delta^{\mathcal{I}}$ that is associated to c and is unique. As (T_c, ρ_c) is a tile, it must satisfy the condition TF6 and hence $A \in T_c$. Since $A^{\mathcal{I}}$ is defined as a set of all domain elements $(T, \rho)_i$ such that $A \in T$, we have that $e_c \in A^{\mathcal{I}}$ and so $\mathcal{I}$ satisfies $A(c)$. Let $p(a,b) \in \mathcal{A}$, where $p \in \mathsf{N}_\mathsf{R}$. The first step in our construction of $\mathcal{I}$ involved adding $(e_a, e_b) \in r^{\mathcal{I}}$ for each assertion $p(a,b) \in \mathcal{A}$ and role $r \sqsubseteq p \in \mathcal{T}$. Due to the closure assumption for the TBox, we have $p \sqsubseteq p \in \mathcal{T}$ and thus $(e_a, e_b) \in p^{\mathcal{I}}$. Hence, $\mathcal{I}$ satisfies $r(a,b)$.

Assume that $\neg A(b) \in \mathcal{A}$ and assume towards a contradiction that $e_b \in A^{\mathcal{I}}$. Then $A \in T_b$, which contradicts condition TF7. Further, assume $\neg p(a,b) \in \mathcal{A}$ and $(e_a, e_b) \in p^{\mathcal{I}}$. In

order to construct $(e_a, e_b) \in p^{\mathcal{I}}$ it must be the case that either there is a pair (R, T_b) in ρ_a where $p \in R$, which contradicts TF8 (c) or (R, T_a) in ρ_b, where $p^- \in R$, which contradicts TF8 (d).

$\mathcal{I}$ respects closed predicates. Let $A \in \mathsf{N_C} \cap \Sigma$ and $e = (T, \rho)_i$ be an arbitrary element in $A^{\mathcal{I}}$. By construction of $A^{\mathcal{I}}$, we have that $A \in T$. Further, by condition TF9 there exists some $c \in \mathsf{N_I}$ such that $\{c\} \in T$ and $A(c) \in \mathcal{A}$. As $\{c\} \in \mathsf{N_C^+}(\mathcal{K})$, as explained above, there is exactly one domain element $e_c = (T_c, \rho_c) \in \Delta^{\mathcal{I}}$ with $\{c\} \in T_c$. Hence, $e = e_c$ and as $A(c) \in \mathcal{A}$, $\mathcal{I}$ respects closed concepts.

Let $r \in \mathsf{N_R} \cap \Sigma$ and let $e_1 = (T_1, \rho_1)_i$ and $e_2 = (T_2, \rho_2)_j$ arbitrary elements of $\Delta^{\mathcal{I}}$ for which $(e_1, e_2) \in r^{\mathcal{I}}$ holds. We now make a case distinction based on at which point in the construction of $\mathcal{I}$ we set $(e_1, e_2) \in r^{\mathcal{I}}$.

- We set $(e_1, e_2) \in r^{\mathcal{I}}$ in the first step of the construction, in order to satisfy the ABox. In this case, $e_1 = e_a$ and $e_2 = e_b$, for some a, b for which $r(a, b) \in \mathcal{A}$ (also since $\mathcal{K}$ respects RIs). Hence, the closed role is not violated.
- We set $(e_1, e_2) \in r^{\mathcal{I}}$ in any of the other cases. In this case, we can see that a prerequisite to setting $(e_1, e_2) \in r^{\mathcal{I}}$ is that there is some $(R, T_2) \in \rho_1$ such that $r \in R$. Then, due to condition TF10, we have that there exists $c \in \mathsf{N_I}$ occurring in $\mathcal{A}$ such that $\{c\} \in T_1$. Hence, $e_1 = e_c$. Further, also due to the same condition, we have that there is some $d \in \mathsf{N_I}$ occurring in $\mathcal{A}$ such that $\{d\} \in T_2$ and therefore $e_2 = e_b$. The final part of the condition TF10 requires states that $r(c, d) \in \mathcal{A}$. Hence $(e_1, e_2) \in r^{\mathcal{I}}$ does not violate the closed role.

We can conclude that $\mathcal{I}$ respects closed roles. From this and the fact that $\mathcal{I}$ satisfies all assertions in $\mathcal{A}$, it follows that $\mathcal{I} \vDash_{\Sigma} \mathcal{A}$.

Satisfaction of $\mathcal{T}$. Consider an arbitrary axiom α in $\mathcal{T}$.

- α is of the shape $B_1 \sqcap \cdots \sqcap B_{k-1} \sqsubseteq B_k \sqcup \cdots \sqcup B_m$: Let $e = (T, \rho)_i \in \Delta^{\mathcal{I}}$ and assume that $e \in (B_1 \sqcap \cdots \sqcap B_{k-1})^{\mathcal{I}}$. This means that $e \in B_j^{\mathcal{I}}$, for all $1 \leq j < k$. By construction of $\mathcal{I}$, this implies that $B_j \in T$, for all $1 \leq j < k$. By condition TF2. of Definition 3.2.5 we have that there exists l, $k \leq l \leq m$, such that $B_l \in T$. Thus, $e \in B_l^{\mathcal{I}}$ and so $e \in (B_k \sqcup \cdots \sqcup B_m)^{\mathcal{I}}$.
- α is of the shape $B_1 \sqsubseteq \forall r.B_2$: Assume $e = (T, \rho)_i \in \Delta^{\mathcal{I}}$, $e' = (T', \rho')_j \in \Delta^{\mathcal{I}}$, $(e, e') \in r^{\mathcal{I}}$, $e \in B_1^{\mathcal{I}}$, and $e_2 \notin B_2^{\mathcal{I}}$. By construction, $B_1 \in T$ and $B_2 \notin T'$. Due to items TF4 (a) and TF4 (b), $(e, e') \in r$ could not have been constructed in steps 2-4. Thus, it must be the case that e and e' represent constants. Since this contradicts condition MF5, we can conclude that this situation is impossible and t is satisfied.

- α is of the shape $B_1 \sqsubseteq \exists r.B_2$: Let $e = (T, \rho)_i \in \Delta^{\mathcal{I}}$ and assume that $e \in B_1^{\mathcal{I}}$. By construction of $\mathcal{I}$, we have that $B_1 \in T$. By condition TF3 of Definition 3.2.5 we have that there is some $(R, T') \in \rho$ such that $r \in R$ and $B_2 \in T'$. Hence, latest at step 3, we will construct $(e, e') \in r^{\mathcal{I}}$, for some $e' = (T', \rho')$. As $B_2 \in T'$, by construction we have $e' \in B_2^{\mathcal{I}}$ and so α is satisfied.

- t is of the shape $r \sqsubseteq s$: The satisfaction of these axioms is due to the condition TF4 (c) as well as that in the first step of the construction, whenever we add $(e, e') \in r^{\mathcal{I}}$, we do the same for all roles s with $r \sqsubseteq s$.

- t is of the shape $\mathsf{func}(r)$: Let $e = (T, \rho)_i$, $e_1 = (T_1, \rho_1)_j$, and $e_2 = (T_2, \rho_2)_k$ be elements in $\Delta^{\mathcal{I}}$ and assume that $(e, e_1) \in r^{\mathcal{I}}$ and $(e, e_2) \in r^{\mathcal{I}}$. Note that, steps 1-3 can together construct at most one r-arc from e to some other element, thus these steps alone cannot be responsible for the situation. The same holds for step 4. We conclude that the violation arises as follows. W.lo.g., assume that steps 1-3 construct $(e, e_1) \in r^{\mathcal{I}}$ and step 4 constructs $(e, e_2) \in r^{\mathcal{I}}$ due to a pair $(R_2, T_2) \in \rho$ with $r \in R_2$. Now, if $(e, e_1) \in r^{\mathcal{I}}$ was constructed in steps 2-3, then there must be also be a pair $(R_1, T_1) \in \rho$ with $r \in R_1$. Due to TF5, (R_1, T_1) and (R_2, T_2) must be the same pair. Thus, there already exists a witness for (R_2, T_2) and step 4. is never actually executed. On the other hand, if $(e, e_1) \in r^{\mathcal{I}}$ was constructed in step 1, then due to conditions TF8 (a) and TF8 (b) and MF1, we can conclude that $T_1 = T_2$. As e_1 represents a constant and is thus the only element with the type T_2, we can conclude that $e_1 = e_2$ and that there is no violation.

As we have shown that $\mathcal{I} \models_{\Sigma} \mathcal{A}$ and $\mathcal{I} \models \mathcal{T}$, we have that $\mathcal{I} \models \mathcal{K}$ and thus $\mathcal{K}$ is satisfiable.

Missing Proof of Theorem 5.3.6

In this section, we provide the proof of Theorem 5.3.6. The following proof relies greatly on the proof of Theorem 3.3.4 from the previous section and omits the cases can trivially adapted.

Let $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$ be an $\mathcal{ALCHOIF}^+$ KB.

($\Rightarrow$): Assume $\mathcal{K}$ is satisfiable, i.e., there is an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$. It is easy to show that $\mathcal{K}$ respects role inclusions.

We next show that we can construct a mosaic for $\mathcal{K}$ from $\mathcal{I}$. For each domain element $e \in \Delta^{\mathcal{I}}$ we define the type of e as the set of basic concepts e participates in:

$$\mathsf{t}(e) = \{\top\} \cup \{A \in \mathsf{N}_{\mathsf{C}}^{+}(\mathcal{K}) : e \in A^{\mathcal{I}}\}.$$

Further, for each pair of elements $e, e' \in \Delta^{\mathcal{I}}$, we define a role type of (e, e') as

$$\mathsf{rt}(e, e') = \{r \in \mathrm{N}_R^{+}(\mathcal{K}) : (e, e') \in r^{\mathcal{I}}\}.$$

Next, we need to extract $\pi_\exists(e)$ and $\pi_\forall(e)$. We begin with the former.

To this end, for each $e \in \Delta^{\mathcal{I}}$, we set

$$\pi_\exists(e) := \{(\varepsilon, \{a\})\},$$

if $e = a$, for some $\{a\} \in \mathsf{N}_{\mathsf{C}}^{+}(\mathcal{K})$, otherwise $\pi_\exists(e) := \emptyset$. We then go through all pairs in $\Pi_\exists(\mathcal{K})$ of the shape $(r \circ A? \circ P, C) \in \Pi_\exists(\mathcal{K})$, where $r \in \mathsf{N}_R^{+}$, $A \in \mathsf{N}_{\mathsf{C}}^{+}(\mathcal{K})$, $P \in \mathsf{Paths}_\exists(\mathcal{K})$, and $C \in \mathsf{N}_{\mathsf{C}}^{+}(\mathcal{K})$, and we do the following. For each $(e, e') \in r^{\mathcal{I}}$, if $(r \circ A? \circ P, C) \in \pi_\exists(e)$ and $A \in \mathsf{t}(e')$, we set $\pi_\exists(e') := \pi_\exists(e') \cup \{(P, C)\}$. Finally, we go through all pairs $\Pi_\exists(\mathcal{K})$ of the shape $(A? \circ R, D)$, where $A \in \mathsf{N}_{\mathsf{C}}^{+}(\mathcal{K})$, R is a complex role and $D \in \mathsf{N}_{\mathsf{C}}^{+}(\mathcal{K})$, and for each $e \in A^{\mathcal{I}}$, if $(R, D) \in \pi_\exists(e)$ then $(A \circ R, D) \in \pi_\exists(e)$. This concludes the construction of $\pi_\exists(e)$.

We next extract $\pi_\forall(e)$, for all $e \in \Delta^{\mathcal{I}}$. We begin by setting

$$\pi_\forall(e) := \{(P, \{a\}) : (A? \circ P, \{a\}) \in \Pi_\forall(\mathcal{K}), A \in \mathsf{t}(e)\},$$

if $e = a$ for some $\{a\} \in \mathsf{N}_{\mathsf{C}}^{+}(\mathcal{K})$. Otherwise, $\pi_\forall(e) := \emptyset$. Further, for each $e \in \Delta^{\mathcal{I}}$ s.t. $(a, e) \in r^{\mathcal{I}}$, for some $\{a\} \in \mathsf{N}_{\mathsf{C}}^{+}(\mathcal{K})$, $A \in \mathsf{t}(e)$ and $(r \circ A? \circ P, \{a\}) \in \pi_\forall(a)$, we set $\pi_\forall(e) := \pi_\forall(e) \cup \{(P, \{a\})\}$. Finally, for all $e \in \Delta^{\mathcal{I}}$ and each $P \in \mathsf{Paths}_\forall(\mathcal{K})$ s.t. there are two nominals $\{a\}, \{b\} \in \mathsf{N}_{\mathsf{C}}^{+}(\mathcal{K})$ with $(P, \{a\}), (P, \{b\}) \in \pi_\forall(e)$, we set $\pi_\forall(e) := \pi_\forall(e) \setminus \{(P, C) : (P, C) \in \pi_\forall(e)\} \cup \{(P, \bot)\}$.

Observation 1. *Given an arbitrary domain element $e \in \Delta^{\mathcal{I}}$, if $(r \circ A? \circ R, \bot) \in \pi_\forall(e)$, then $(R, \bot) \in \pi_\forall(e')$, for all e' with $(e, e') \in r^{\mathcal{I}}$ and $e' \in A^{\mathcal{I}}$.*

Finally, for each element $e \in \Delta^{\mathcal{I}}$, we extract a tile $\tau_e = (\mathsf{t}(e), \rho(e), \pi_\exists(e), \pi_\forall(e))$ that describes it. Sometimes we write T_e as an abbreviation for $\mathsf{t}(e)$. To define $\rho(e)$ we do the following. As $\mathcal{I}$ is a model of $\mathcal{K}$, $\mathcal{I}$ satisfies every existential axiom in $\mathcal{T}$. This means that for every axiom $\alpha \in \mathcal{T}$ that is of the type $A \sqsubseteq \exists r.B$, if $A \in \mathsf{t}(e)$, then there exists at least one element $e_\alpha \in \Delta^{\mathcal{I}}$ such that $(e, e_\alpha) \in r^{\mathcal{I}}$ and $e_\alpha \in B^{\mathcal{I}}$. If there are multiple choices for e_α, we pick an arbitrary one. Let $E(e) = \{e_\alpha : \alpha = A \sqsubseteq \exists r.B \in \mathcal{T}, A \in \mathsf{t}(e)\}$. Further, let $F(e) = \{e' : (e, e') \in r^{\mathcal{I}}, \mathsf{func}(r) \in \mathcal{T}\}$. We set

$$\rho(e) = \{(\mathsf{rt}(e, e'), \mathsf{t}(e'), \pi_\exists(e'), \pi_\forall(e')) : e' \in E(e) \cup F(e)\}.$$

Observation 2. *By construction of ρ_e, if $(R, T', \pi'_\exists, \pi'_\forall) \in \rho_e$, then there is an element $e' \in \Delta^{\mathcal{I}}$ s.t. $\mathsf{t}(e') = T'$, $\mathsf{rt}(e, e') = R$, $\pi'_\exists = \pi_\exists(e')$ and $\pi'_\forall = \pi_\forall(e')$.*

We next need to verify that τ_e is a proper tile. We only give details for the newly-introduced conditions , i.e., TF$^+$11-TF$^+$21.

Notational remark: For $P = r_1 \circ A_1? \circ \cdots \circ r_n \circ A_n? \in \mathsf{Paths}_\exists \cup \mathsf{Paths}_\forall$, we define the length of P as $len(P) = n$. Furthermore, $\mathsf{len}(\varepsilon) = 0$.

TF$^+$11. $|\pi_\exists| \leq |\mathsf{Paths}_\exists(\mathcal{K})|$

By construction, for each $e \in \Delta^\mathcal{I}$ and each $P \in \mathsf{Paths}_\exists(\mathcal{K})$, if $(P, \{a\}) \in \pi_\exists(e)$ then $(e, \{a\}) \in P^\mathcal{I}$. Moreover, by definition of $\mathsf{Paths}_\exists(\mathcal{K})$, P consists of only tests and functional roles, which means that there there is at most one domain element $e' \in \Delta^\mathcal{I}$ s.t. $(e, e') \in P^\mathcal{I}$. Thus, for each $P \in \mathsf{Paths}_\exists(\mathcal{K})$ there is at most one $\{a\} \in \mathsf{N}_\mathsf{C}^+(\mathcal{K})$ with $(P, \{a\}) \in \pi_\exists(e)$ and so $|\pi_\exists| \leq |\mathsf{Paths}_\exists(\mathcal{K})|$.

TF$^+$12. $|\pi_\forall| \leq |\mathsf{Paths}_\forall(\mathcal{K})|$

It is easy to verify that, by construction, for each element $e \in \Delta^\mathcal{I}$ and each $P \in \mathsf{Paths}_\forall(\mathcal{K})$ either (i) $(P, \bot) \in \pi_\forall(e)$, (ii) there is exactly one nominal $\{a\} \in \mathsf{N}_\mathsf{C}^+(\mathcal{K})$ s.t. $(P, \{a\}) \in \pi_\exists(e)$, or (iii) there is no $C \in \mathsf{N}_\mathsf{C}^+(\mathcal{K}) \cup \{\bot\}$ s.t. $(P, C) \in \pi_\exists(e)$. Thus, $|\pi_\forall| \leq |\mathsf{Paths}_\forall(\mathcal{K})|$.

TF$^+$13. If $\{a\} \in \mathsf{N}_\mathsf{C}^+(\mathcal{K})$, $\{a\} \in T$, then $(\varepsilon, \{a\}) \in \pi_\exists$

Follows immediately from the construction of $\pi_\exists(e)$

TF$^+$14. If $\exists A_1? \circ P.(\{x\} \sqcap \exists s.\{y\}) \sqcap \exists A_2? \circ R.\{y\} \sqsubseteq A \in \mathcal{T}$, $\{\{a\}, \{b\}\} \subseteq \mathsf{N}_\mathsf{C}^+(\mathcal{K})$, $\{A_1, A_2\} \subseteq T$, $\{(P, \{a\}), (R, \{b\})\} \subseteq \pi_\exists$ and $s(a, b) \in \mathcal{A}$, then $A \in \mathcal{T}$.

Let $\alpha = \exists A_1? \circ P.(\{x\} \sqcap \exists s.\{y\}) \sqcap \exists A_2? \circ R.\{y\} \sqsubseteq A$ s.t. $\alpha \in \mathcal{T}$, let $e \in \Delta^\mathcal{I}$ be an arbitrary domain element and consider the tile τ_e extracted from e. Assume that $\{A_1, A_2\} \in \mathsf{t}(e)$. By construction, this means that $e \in A_1^\mathcal{I}$ and $e \in A_2^\mathcal{I}$. Further, assume that $\{(P, \{a\}), (R, \{b\})\} \subseteq \pi_\exists(e)$. Observe once again that, by construction of $\pi_\exists(e)$, this means that $(e, a) \in P^\mathcal{I}$ and $(e, b) \in R^\mathcal{I}$. If $s(a, b) \in \mathcal{A}$, then it must be the case that $(a, b) \in s^\mathcal{I}$. In the light of our previous observations, both $(e, b) \in A_1? \circ P.(\{a\} \sqcap \exists s.\{b\})$ and $(e, b) \in A_2? \circ R.\{b\}$ must hold, and so $e \in A^\mathcal{I}$. Once again, by construction of $\mathsf{t}(e)$, we have that $A \in \mathsf{t}(e)$.

TF$^+$15. If $\exists A_1? \circ P.(\{x\} \sqcap \neg\exists s.\{y\}) \sqcap \exists A_2? \circ R.\{y\} \sqsubseteq A \in \mathcal{T}$, $\{\{a\}, \{b\}\} \subseteq \mathsf{N}_\mathsf{C}^+(\mathcal{K})$, $\{A_1, A_2\} \subseteq T$, $\{(P, \{a\}), (R, \{b\})\} \subseteq \pi_\exists$ and $s(a, b) \notin \mathcal{A}$, then $A \in \mathcal{T}$.

Let $\alpha = \exists A_1? \circ P.(\{x\} \sqcap \neg\exists s.\{y\}) \sqcap \exists A_2? \circ R.\{y\} \sqsubseteq A$ s.t. $\alpha \in \mathcal{T}$, let $e \in \Delta^\mathcal{I}$ be an arbitrary domain element and consider the tile τ_e extracted from e. Assume that $\{A_1, A_2\} \in \mathsf{t}(e)$. By construction, this means that $e \in A_1^\mathcal{I}$ and $e \in A_2^\mathcal{I}$. Further, assume that $\{(P, \{a\}), (R, \exists(b))\} \subseteq \pi_\exists(e)$. Observe once again that, by construction of $\pi_\exists(e)$, this means that $(e, a) \in P^\mathcal{I}$ and $(e, b) \in R^\mathcal{I}$. If $s(a, b) \notin \mathcal{A}$, then it must be the case that $(a, b) \notin s^\mathcal{I}$. In the light of our previous observations, both $(e, b) \in A_1? \circ P(\{x\} \sqcap \neg\exists s.\{b\})$ and $(e, b) \in A_2? \circ R.\{b\}$ must hold, and so $e \in A^\mathcal{I}$. Once again, by construction of $\mathsf{t}(e)$, we have that $A \in \mathsf{t}(e)$.

TF$^+$16. If $\{x\} \sqsubseteq \forall A? \circ P.\{x\} \in \mathcal{T}$, $\{a\} \in \mathsf{N}_\mathsf{C}^+(\mathcal{K})$, and $\{\{a\}, A\} \in T$, then either $(P, \{a\}) \in \pi_\forall$ or $(P, \bot) \in \pi_\forall$.

Let $\{\{a\}, A\} \in \mathsf{t}(e)$, for some $\{a\} \in \mathsf{N}_\mathsf{C}^+(\mathcal{K})$. This means that $e \in A^\mathcal{I}$ and $e = a$. Moreover, assume $\{x\} \sqsubseteq \forall A? \circ P.\{x\} \in \mathcal{T}$. By construction of $\Pi_\forall(\mathcal{K})$, we have that $(A? \circ P, \{a\}) \in \Pi_\forall(\mathcal{K})$, and in the first step of the construction of $\pi_\forall(a)$, we let $(P, \{a\}) \in \pi_\forall(a)$. Thus, the only way to achieve $(P, \{a\}) \notin \pi_\forall(a)$ is if we replace it by $(P, \bot)$ at some later step.

TF$^+$17. If $(\varepsilon, C) \in \pi_\forall$, then $C \in T$.

We make a case distinction:

- $C = \{a\}$, for some $\{a\} \in \mathsf{N}_\mathsf{C}^+(\mathcal{K})$. Looking at the construction of $\pi_\forall(e)$, it is easy to see that if $(\varepsilon, \{a\}) \in \pi_\forall(e)$, then there is some $(A? \circ P, \{a\}) \in \Pi_\forall(\mathcal{K})$ s.t. $\{a\} \in \mathsf{t}(a)$, i.e., $A^\mathcal{I}$, and $(a, e) \in P^\mathcal{I}$. Further, by the construction of $\Pi_\forall(\mathcal{K})$, there must be some axiom $\alpha = \{x\} \sqsubseteq \forall A? \circ P.\{x\} \in \mathcal{T}$. As $\mathcal{I} \vDash \alpha$, and $(a, e) \in P^\mathcal{I}$, it must be that $e = a$ and so $\{a\} \in \mathsf{t}(e)$.
- $C = \bot$. Since $\mathsf{t}(e)$ can never contain $\bot$, we need to show that $(\varepsilon, \bot)$ cannot possibly be in $\pi_\forall(e)$. To this end, observe that the only way to obtain $(\varepsilon, \bot)$ in $\pi_\forall(e)$ is by replacing $(\varepsilon, \{a\})$ and $(\varepsilon, \{b\})$ during the construction of $\pi_\forall(e)$, for two nominals $\{a\}, \{b\} \in \mathsf{N}_\mathsf{C}^+(\mathcal{K})$. Analogously to the previous case, this means that there are two axioms $\alpha_1 = \{x\} \sqsubseteq \forall A_1? \circ P_1.\{x\} \in \mathcal{T}$ and $\alpha_2 = \{x\} \sqsubseteq \forall A_2? \circ P_2.\{x\} \in \mathcal{T}$ s.t. $a \in A_1^\mathcal{I}$, $b \in A_2^\mathcal{I}$, $(a, e) \in P_1^\mathcal{I}$ and $(b, e) \in P_2^\mathcal{I}$. As $\mathcal{I} \vDash \alpha_1$ and $\mathcal{I} \vDash \alpha_2$, it must be that $e = a = b$, which cannot happen due to SNA. Thus, $(\varepsilon, \bot) \notin \pi_\forall(e)$.

TF$^+$18. For all $(R, T', \pi'_\exists, \pi'_\forall) \in \rho$

a) If $(r \circ A? \circ P, \{a\}) \in \Pi_\exists(\mathcal{K})$, and $r \in R$, $\{A\} \in T'$, and $(P, \{a\}) \in \pi'_\exists$, then $(r \circ A? \circ P, \{a\}) \in \pi_\exists$
Assume $(r \circ A? \circ P, \{a\}) \in \Pi_\exists(\mathcal{K})$, $(P, \{a\}) \in \pi'_\exists$, $A? \in \mathcal{T}'$ and $r \in R$. Due to Observation 2, we have that there is some domain element $e' \in \Delta^\mathcal{I}$ s.t. $e' \in A^\mathcal{I}$, $\pi'_\exists = \pi_\exists(e)$, and $(e, e') \in r^\mathcal{I}$. Thus, by construction, we have that $(r \circ A? \circ P, \{a\}) \in \pi_\exists(e)$.

b) If $(r \circ A? \circ P, \{a\}) \in \Pi_\exists(\mathcal{K})$, and $r^- \in R$, $A \in T$, and $(P, \{a\}) \in \pi_\exists$, then $(r \circ A? \circ P, \{a\}) \in \pi'_\exists$.
Assume $(r \circ A? \circ P, \{a\}) \in \Pi_\exists(\mathcal{K})$, $(P, \{a\}) \in \pi_\exists$, $A? \in \mathcal{T}$ and $r^- \in R$. Then $e \in A^\mathcal{I}$. Moreover, due to Observation 2, we have that there is some domain element $e' \in \Delta^\mathcal{I}$ s.t. $\pi'_\exists = \pi_\exists(e)$, and $(e', e) \in r^\mathcal{I}$. Thus, by construction of $\pi_\exists(e)$, we have that $(r \circ A? \circ P, \{a\}) \in \pi_\exists(e')$, i.e., $(r \circ A? \circ P, \{a\}) \in \pi'_\exists$.

c) If $(r \circ A? \circ P, C) \in \pi_\forall$, $r \in R$ and $A \in T'$, either $(P, C) \in \pi'_\forall$ or $(P, \bot) \in \pi'_\forall$.
We do a case distinction

- $C = \{a\}$, for some $\{a\} \in \mathsf{N}_\mathsf{C}^+(\mathcal{K})$. Assume that $(r \circ A? \circ P, \{a\}) \in \pi_\forall(e)$, $r \in R$, and $A \in \mathcal{T}'$. Due to Observation 2, there is some

$e' \in \Delta^{\mathcal{I}}$ s.t. $e' \in A^{\mathcal{I}}$, $(e, e') \in r^{\mathcal{I}}$, and $\pi'_\forall = \pi_\forall(e')$. By construction of $\pi_\forall(e')$, we either have $(P, \{a\}) \in \pi_\forall(e')$ or we replaced it by $(P, \bot)$ at a later stage of the construction.

- $C = \bot$. Assume that $(r \circ A? \circ P, \bot) \in \pi_\forall(e)$, $r \in R$, and $A \in \mathcal{T}'$. Due to Observation 2, there is some $e' \in \Delta^I$ s.t. $e' \in A^{\mathcal{I}}$, $(e, e') \in r^{\mathcal{I}}$, and $\pi'_\forall = \pi_\forall(e')$. Further, by Observation 1, we have $(P, \bot) \in \pi_\forall(e')$.

d) If $(r \circ A? \circ P, C) \in \pi'_\forall$, $r^- \in R$ and $A \in T$, either $(P, C) \in \pi_\forall$ or $(P, \bot) \in \pi_\forall$.

We do a case distinction

- $C = \{a\}$, for some $\{a\} \in \mathsf{N}_\mathsf{C}^+(\mathcal{K})$. Assume that $(r \circ A? \circ P, \{a\}) \in \pi'_\forall$, $r^- \in R$, and $A \in \mathcal{T}$, i.e., $e \in \mathcal{A}^{\mathcal{I}}$. Due to Observation 2, there is some $e' \in \Delta^{\mathcal{I}}$ s.t. $(e, e') \in (r^-)^{\mathcal{I}}$, i.e., $(e', e) \in r^{\mathcal{I}}$, and $\pi'_\forall = \pi_\forall(e')$. Thus, by construction of $\pi_\forall(e)$, we either have $(P, \{a\}) \in \pi_\forall(e')$ or we replaced it by $(P, \bot)$ at a later stage of the construction.
- $C = \bot$. Assume that $(r \circ A? \circ P, \bot) \in \pi'_\forall$, $r^- \in R$, and $A \in T$, i.e., $e \in A^{\mathcal{I}}$. Due to Observation 2, there is some $e' \in \Delta^{\mathcal{I}}$ s.t $(e, e') \in (r^-)^{\mathcal{I}}$, i.e., $(e', e) \in r^{\mathcal{I}}$, and $\pi'_\forall = \pi_\forall(e')$ Further, by Observation 1, we have $(P, \bot) \in \pi_\forall(e)$.

e) If $\{x\} \sqcap A \sqsubseteq \forall s.\neg\{x\} \in \mathcal{T}$, $\{a\} \in \mathsf{N}_\mathsf{C}^+(\mathcal{K})$, $\{\{a\}, A\} \subseteq \mathcal{T}$ and $\{s, s^-\} \cap R \neq \emptyset$, then $\{a\} \notin \mathcal{T}$

Assume $\{\{a\}, A\} \in \mathsf{t}(e)$, i.e., $e = a$ and $e \in A^{\mathcal{I}}$. Further assume towards a contradiction that $\{s, s^-\} \cap R \neq \emptyset$ and $\{a\} \in \mathcal{T}$. Due to Observation 2, this means that $(a, a) \in s^{\mathcal{I}}$, which is a contradiction to $\mathcal{I} \vDash \{x\} \sqcap A \sqsubseteq \forall s.\neg\{x\}$.

TF$^+$19. If $p(a, b) \in \mathcal{A}$, $p \sqsubseteq r \in \mathcal{T}$, $\mathsf{func}(r) \in \mathcal{T}$, $\{a\} \in T$, there is some $(R, T', \pi'_\exists, \pi'_\forall) \in \rho$ s.t. $r \in R$ and $\{b\} \in T'$.

Immediate by construction of ρ_a (if r is a functional role, then all r-neighbors are stored).

TF$^+$20. If $p(a, b) \in \mathcal{A}$, $p^- \sqsubseteq r \in \mathcal{T}$, $\mathsf{func}(r) \in \mathcal{T}$, $\{b\} \in T$, there is some $(R, T', \pi'_\exists, \pi'_\forall) \in \rho$ s.t. $r \in R$ and $\{a\} \in T'$.

Immediate by construction of ρ_b (if r is a functional role, then all r-neighbors are stored).

TF$^+$21. If $\{a\} \in \mathsf{N}_\mathsf{C}^+(\mathcal{K})$, $s(a, a) \in \mathcal{A}$ and $\{x\} \sqcap A \sqsubseteq \forall s.\neg\{x\} \in \mathcal{T}$, then $\{\{a\}, A\} \not\subseteq T$.

Assume $s(a, a) \in \mathcal{A}$. As $\mathcal{I} \vDash \{x\} \sqcap A \sqsubseteq \forall s.\neg\{x\}$, this means that $a \notin A$. Thus, by construction, $A \notin \mathsf{t}(a)$. Furthermore, the only tile containing $\{a\}$ in its unary type is the tile $\tau_a = (\mathsf{t}(a), \rho_a, \pi_\exists(a), \pi_\forall(a))$. Thus, $\tau = \tau_a$, and $A \notin T$.

To define a mosaic for $\mathcal{K}$, we define a function $N : \text{Tiles}(\mathcal{K}) \to \mathbb{N}^*$ as

$$N(\tau) = |\{e \in \Delta^{\mathcal{I}} : \tau_e = \tau\}|.$$

Note that if this set is infinite, we set $N(\tau) = \aleph_0$. Finally, in order to show that N is indeed a mosaic for $\mathcal{K}$, we need to show that N satisfies conditions MF$^+$1-MF$^+$6 in Definition 3.2.9. We only show the proof for MF$^+$3, MF$^+$4 and MF$^+$6, as the rest are simple adaptations of the proof from the previous section.

MF$^+$3. For all $T, T' \in \text{Types}(\mathcal{K})$, $R \subseteq \text{N}_R^+(\mathcal{K})$ with $r \in R$ and $\mathsf{func}(r^-) \in \mathcal{T}$, every $\{\pi_\exists, \pi'_\exists\} \subseteq \Pi_\exists(\mathcal{K})$, $\{\pi_\forall, \pi'_\forall\} \subseteq \Pi_\forall(\mathcal{K})$, the following holds:

$$\sum_{\substack{\tau=(T,\rho,\pi_\exists,\pi_\forall)\in\text{Tiles}(\mathcal{K}),\\ (R,T',\pi'_\exists,\pi'_\forall)\in\rho}} N(\tau) \leq \sum_{\substack{\tau'=(T',\rho',\pi'_\exists,\pi'_\forall)\in\text{Tiles}(\mathcal{K}),\\ (R^-,T,\pi_\exists,\pi_\forall)\in\rho'}} N(\tau')$$

Let T, T' be two arbitrary types for $\mathcal{K}$, let R be an arbitrary subset of $\text{N}_R^+(\mathcal{K})$ and assume that there is a role r with $\mathsf{func}(r^-) \in \mathcal{T}$ such that $r \in R$, let $\pi_\exists, \pi'_\exists$ be arbitrary subsets of $\Pi_\exists(\mathcal{K})$, and let $\pi_\forall, \pi'_\forall$, be arbitrary subsets of $\Pi_\forall(\mathcal{K})$.

Since r^- is a functional role and $r^- \in R^-$, by construction, the number of tiles $(T', \rho', \pi'_\exists, \pi'_\forall)$ where $(R^-, T, \pi_\exists, \pi_\forall) \in \rho'$ is exactly the number of pairs $(e', e) \in (r^-)^{\mathcal{I}}$, where $\mathsf{t}(e') = T'$, $\mathsf{t}(e) = T$, $\pi_\exists(e) = \pi_\exists$, $\pi_\forall(e) = \pi_\forall$, $\pi_\exists(e') = \pi'_\exists$, $\pi_\forall(e') = \pi'_\forall$, and $\mathsf{rt}(e', e) = R^-$. This is the value of the sum on the right-hand side of the inequality in MF$^+$3.

Consider an element e with $\mathsf{t}(e) = T$. By Observation 2, each $(R, T', \pi'_\exists, \pi'_\forall) \in \rho_e$ corresponds to an element e' with $\mathsf{t}(e') = T'$,$\mathsf{rt}(e, e') = R$, $\pi'_\exists = \pi_\exists(e')$ and $\pi'_\forall = \pi_\forall(e')$. We compute the left-hand side of the inequality in MF$^+$3 as the number of tiles (T, ρ) with $(R, T') \in \rho$, which can be *at most* the number of pairs $(e, e') \in r^{\mathcal{I}}$ such that $\mathsf{t}(e) = T$, $\mathsf{t}(e') = T'$, $\pi_\exists(e) = \pi_\exists$, $\pi_\forall(e) = \pi_\forall$, $\pi_\exists(e') = \pi'_\exists$, $\pi_\forall(e') = \pi'_\forall$, and $\mathsf{rt}(e, e') = R$. The latter is exactly the number of pairs $(e', e) \in (r^-)^{\mathcal{I}}$, where $\mathsf{t}(e') = T'$, $\mathsf{t}(e) = T$, $\pi_\exists(e) = \pi_\exists$, $\pi_\forall(e) = \pi_\forall$, $\pi_\exists(e') = \pi'_\exists$, $\pi_\forall(e') = \pi'_\forall$, and $\mathsf{rt}(e', e) = R^-$, which is the value of the sum of the right-hand side of the inequality. This establishes that the inequality in MF$^+$3 holds.

MF$^+$4. For all $\tau = (T, \rho, \pi_\exists, \pi_\forall) \in \text{Tiles}(\mathcal{K})$ and $(R, T', \pi'_\exists, \pi'_\forall) \in \rho$ the following holds: if $N(\tau) > 0$, then there exists ρ' such that $\tau' = (T', \rho', \pi'_\exists, \pi'_\forall) \in \text{Tiles}(K)$ and $N(\tau') > 0$.

Assume there is a tile τ for $\mathcal{K}$ such that $N(\tau) > 0$. That means that there is at least one element $e \in \Delta^{\mathcal{I}}$ with $\tau = \tau_e$. Let $(R, T', \pi_\exists, \pi_\forall) \in \rho$. By Observation 2, this corresponds to some domain element e' with $\mathsf{t}(e') = T'$, $\mathsf{rt}(e, e') = R$, $\pi'_\exists = \pi_\exists(e')$ and $\pi'_\forall = \pi_\forall(e')$. We then have that $N(\tau_{e'}) \geq 1$, and so the inequality in MF$^+$4 holds.

MF$^+$6. For all $p(a,b) \in \mathcal{A}$ and $\tau = (T, \rho, \pi_\exists, \pi_\forall) \in \text{Tiles}(\mathcal{K})$ with $\{a\} \in T$, $N(\tau) > 0$ implies that there exists a tile $\tau' = (T', \rho', \pi'_\exists, \pi'_\forall)$ with $N(\tau') > 0$ s.t.

a) $\{b\} \in T'$,

b) for all $(r \circ A? \circ P, C) \in \Pi_\exists(\mathcal{K})$, $p \sqsubseteq r \in \mathcal{T}$, $(P, C) \in \pi'_\exists$, and $A \in T'$, we have $(r \circ A? \circ P, C) \in \pi_\exists$,

c) for all $(r \circ A? \circ P, C) \in \Pi_\exists(\mathcal{K})$, $p^- \sqsubseteq r \in \mathcal{T}$, $(P, C) \in \pi_\exists$, and $A \in T$, we have $(r \circ A? \circ P, C) \in \pi'_\exists$,

d) for all $(r \circ A? \circ P, C) \in \pi_\forall$, $p \sqsubseteq r \in \mathcal{T}$, and $A \in T'$, we have

$$\{(P, C), (P, \bot)\} \cap \pi'_\forall \neq \emptyset,$$

e) for all $(r \circ A? \circ P, C) \in \pi'_\forall$, $p^- \sqsubseteq r \in \mathcal{T}$, and $A \in T$, we have

$$\{(P, C), (P, \bot)\} \cap \pi_\forall \neq \emptyset.$$

Assume $p(a,b) \in \mathcal{A}$. The only tile $\tau = (T, \rho, \pi_\exists, \pi_\forall)$ with $\{a\} \in T$ is the tile obtained from the constant a itself, i.e., $\tau_a = (\mathtt{t}(a), \rho_a, \pi_\exists(a), \pi_\forall(a))$. Then, showing that MF$^+$6 holds boils down to showing that the following conditions hold for the tile $\tau_b = (\mathtt{t}(b), \rho_b, \pi_\exists(b), \pi_\forall(b))$ obtained from the constant b.

(b) for all $(r \circ A? \circ P, C) \in \Pi_\exists(\mathcal{K})$, $p \sqsubseteq r \in \mathcal{T}$, $(P, C) \in \pi_\exists(b)$, and $A \in \mathtt{t}(b)$, we have $(r \circ A? \circ P, C) \in \pi_\exists(a)$:
Assume $(r \circ A? \circ P, C) \in \Pi_\exists(\mathcal{K})$, $p \sqsubseteq r \in \mathcal{T}$, $(P, C) \in \pi_\exists(b)$, and $A \in \mathtt{t}(b)$. As $\mathcal{I} \vDash p \sqsubseteq r$ and $(a,b) \in p^\mathcal{I}$, due to $p(a,b) \in \mathcal{A}$, we have that $(a,b) \in r^\mathcal{I}$. Moreover, we have $b \in A^\mathcal{I}$ and $(P, C) \in \pi_\exists(b)$. Thus, by construction of $\pi_\exists(a)$, we must have $(r \circ A? \circ P, C) \in \pi_\exists(a)$.

(c) for all $(r \circ A? \circ P, C) \in \Pi_\exists(\mathcal{K})$, $p^- \sqsubseteq r \in \mathcal{T}$, $(P, C) \in \pi_\exists(a)$, and $A \in \mathtt{t}(a)$, we have $(r \circ A? \circ P, C) \in \pi_\exists(b)$:
Assume $(r \circ A? \circ P, C) \in \Pi_\exists(\mathcal{K})$, $p^- \sqsubseteq r \in \mathcal{T}$, $(P, C) \in \pi_\exists(a)$, and $A \in \mathtt{t}(a)$. As $\mathcal{I} \vDash p^- \sqsubseteq r$ and $(a,b) \in p^\mathcal{I}$, due to $p(a,b) \in \mathcal{A}$, we have that $(b,a) \in r^\mathcal{I}$. Moreover, we have $a \in A^\mathcal{I}$ and $(P, C) \in \pi_\exists(a)$. Thus, by construction of $\pi_\exists(b)$, we must have $(r \circ A? \circ P, C) \in \pi_\exists(b)$.

(d) for all $(r \circ A? \circ P, C) \in \pi_\forall(a)$ s.t. $p \sqsubseteq r \in \mathcal{T}$ and $A \in \mathtt{t}(b)$, we have that $\{(P, C), (P, \bot)\} \cap \pi_\forall(b) \neq \emptyset$:
As $\mathcal{I} \vDash p \sqsubseteq r$ and $(a,b) \in p^\mathcal{I}$, due to $p(a,b) \in \mathcal{A}$, we have that $(a,b) \in r^\mathcal{I}$. Since $A \in \mathtt{t}(b)$, i.e., $b \in A^\mathcal{I}$, we have by construction of $\pi_\forall(b)$, that (P, C) or $(P, \bot)$ is in $\pi_\forall(b)$ (same argument as in the proof of TF$^+$18 (b))

(e) for all $(s \circ B? \circ S, D) \in \pi_\forall(b)$ s.t. $p^- \sqsubseteq s \in \mathcal{T}$ and $B \in \mathtt{t}(a)$, we have that $\{(S, D), (S, \bot)\} \cap \pi_\forall(a) \neq \emptyset$:
As $\mathcal{I} \vDash p^- \sqsubseteq r$ and $(a,b) \in p^\mathcal{I}$, due to $p(a,b) \in \mathcal{A}$, we have that $(b,a) \in r^\mathcal{I}$. Since $B \in \mathtt{t}(a)$, i.e., $a \in B^\mathcal{I}$, we have by construction of $\pi_\forall(a)$, that (S, D) or $(S, \bot)$ is in $\pi_\forall(a)$ (same argument as in the proof of TF$^+$18 (c))

Thus, the condition MF^+6 is satisfied by N.

We have thus shown that N is indeed a mosaic for $\mathcal{K}$, which completes this direction of the proof.

($\Leftarrow$) : $\mathcal{K}$ respects role inclusions and let N be a mosaic for $\mathcal{K}$. We show how to construct a model $\mathcal{I}$ of $\mathcal{K}$.

Intuitively, each tile can be seen as a description of a domain element and a description of (the relevant part of) its neighborhood. A mosaic for $\mathcal{K}$ tells us for each tile τ, how many domain elements that fit the description provided by τ we need in order to build a model of $\mathcal{K}$. Thus, we set our domain

$$\Delta^{\mathcal{I}} = \{\tau_i : \tau \in \mathsf{Tiles}(\mathcal{K}) \text{ and } 0 < i \leq N(\tau)\}.$$

Intuitively, the constant τ_i corresponds to the i-th domain element described by the tile τ. Further, for every $B \in \mathsf{N}_{\mathsf{C}}^{+}(\mathcal{K}) \setminus \{\top, \bot\}$, we set

$$B^{\mathcal{I}} = \{\tau_i \in \Delta^{\mathcal{I}} : \tau = (T, \rho, \pi_\exists, \pi_\forall), B \in T, 1 \leq i \leq N(\tau)\}.$$

The extensions of roles are constructed in multiple steps.

1. We first make sure that that $\mathcal{I}$ satisfies the ABox $\mathcal{A}$. Due to MF^+1, for each nominal $\{a\} \in \mathsf{N}_{\mathsf{C}}^{+}(\mathcal{K})$, we know that there is exactly one domain element $e_a = (T_a, \rho_a, \pi_{\exists a}, \pi_{\forall a})_1$ s.t. $\{a\} \in T_a$. Hence, there is exactly one domain element participating in $\{a\}^{\mathcal{I}}$ and that is e_a. This element "represents" the constant a in $\mathcal{I}$ (and can be renamed to a to comply with SNA). Now, for each assertion $r(a, b) \in \mathcal{A}$ and each role s with $r \sqsubseteq s \in \mathcal{T}$, we set $(e_a, e_b) \in s^{\mathcal{I}}$, if s is a role name and $(e_b, e_a) \in s^{\mathcal{I}}$, otherwise.

We need to take special care when we construct the extensions of roles that are involved in functionality assertions.

2. For each role $r \in \mathrm{N}_R^+(\mathcal{K})$ with $\mathsf{func}(r) \in \mathcal{T}$ and $\mathsf{func}(r^-) \in \mathcal{T}$ we do the following. Using the inequalities from MF^+3 in two directions, we have that for every $T, T' \in \mathrm{Types}(\mathcal{K})$, $\pi_\exists, \pi'_\exists \in \Pi_\exists(\mathcal{K})$, $\pi_\forall, \pi'_\forall \in \Pi_\forall(\mathcal{K})$, and $R \subseteq \mathrm{N}_R^+(\mathcal{K})$ with $r \in R$, the following is satisfied:

$$\sum_{\substack{\tau=(T,\rho,\pi_\exists,\pi_\forall)\in \mathrm{Tiles}(\mathcal{K}),\\ (R,T',\pi'_\exists,\pi'_\forall)\in\rho}} N(\tau) = \sum_{\substack{\tau'=(T',\rho',\pi'_\exists,\pi'_\forall)\in \mathrm{Tiles}(\mathcal{K}),\\ (R^-,T,\pi_\exists,\pi_\forall)\in\rho'}} N(\tau')$$

Thus, the sets

$$X_{T,T',\pi_\exists,\pi'_\exists,\pi_\forall,\pi'_\forall,R} = \{(T, \rho, \pi_\exists, \pi_\forall)_i \in \Delta^{\mathcal{I}} : (R, T', \pi'_\exists, \pi'_\forall) \in \rho, \\ 1 \leq i \leq N((T, \rho, \pi_\exists, \pi_\forall))\}$$

$$Y_{T,T',\pi_\exists,\pi'_\exists,\pi_\forall,\pi'_\forall,R} = \{(T',\rho',\pi'_\exists,\pi'_\forall)_i \in \Delta^{\mathcal{I}} : (R^-,T,\pi_\exists,\pi_\forall) \in \rho, \\ 1 \leq i \leq N((T',\rho',\pi'_\exists,\pi'_\forall))\}$$

have the same cardinality and so there exists a bijection $f_{T,T',\pi_\exists,\pi'_\exists,\pi_\forall,\pi'_\forall,R}$ between them with them.

Furthermore, the bijection $f_{T,T',\pi_\exists,\pi'_\exists,\pi_\forall,\pi'_\forall,R}$ has the following property for all elements $e \in X_{T,T',\pi_\exists,\pi'_\exists,\pi_\forall,\pi'_\forall,R}$: If r is a role name (resp. an inverse role) and $(e,e') \in r^{\mathcal{I}}$ (resp. $(e',e) \in (r^-)^{\mathcal{I}}$) was constructed in step 1, then $f_{T,T',\pi_\exists,\pi'_\exists,\pi_\forall,\pi'_\forall,R}(e) = e'$.

To see this, assume r is a role name and $(e,e') \in r^{\mathcal{I}}$ or was constructed in the first step (similarly for r an inverse role and $(e',e) \in (r^-)^{\mathcal{I}}$). Then, e and e' correspond to some constants a and b and we must have one of the following scenarios:

a) $p(a,b) \in \mathcal{A}$, $p \sqsubseteq r \in \mathcal{T}$, or
b) $p(b,a) \in \mathcal{A}$, $p \sqsubseteq r^- \in \mathcal{T}$.

This means that $\{a\} \in T$. Due to the conditions TF^+8 (a) and TF^+8 (b), we can conclude that $\{b\} \in T'$. In the light of this, due to MF^+1, e is the only element in $X_{T,T',\pi_\exists,\pi'_\exists,\pi_\forall,\pi'_\forall,R}$ and e' is the only element in $Y_{T,T',\pi_\exists,\pi'_\exists,\pi_\forall,\pi'_\forall,R}$. Thus, $f_{T,T',\pi_\exists,\pi'_\exists,\pi_\forall,\pi'_\forall,R}(e) = e'$.

We now do the following. For every $e \in X_{T,T',\pi_\exists,\pi'_\exists,\pi_\forall,\pi'_\forall,R}$ and every $s \in R$, we set $(e, f_{T,T',\pi_\exists,\pi'_\exists,\pi_\forall,\pi'_\forall,R}(e)) \in s^{\mathcal{I}}$ if s is a role name and $(f_{T,T',\pi_\exists,\pi'_\exists,\pi_\forall,\pi'_\forall,R}(e), e) \in (s^-)^{\mathcal{I}}$ otherwise. Intuitively, this connects via an r-arc, every domain element of type T that requires an r-neighbor of type T' to one domain element of type T' that requires an r^--neighbor of type T.

3. Next, for every role $r \in \mathrm{N}_R^+(\mathcal{K})$ such that $\mathsf{func}(r^-) \in \mathcal{T}$ but $\mathsf{func}(r) \notin \mathcal{T}$ we do the following. For every pair $T,T' \in \mathrm{Types}(\mathcal{K})$, $\pi_\exists,\pi'_\exists \in \Pi_\exists(\mathcal{K})$, $\pi_\forall,\pi'_\forall \in \Pi_\forall(\mathcal{K})$, and every $R \subseteq \mathrm{N}_R^+(\mathcal{K})$ with $r \in R$, we get by employing MF^+3 that the following is satisfied:

$$\sum_{\substack{\tau=(T,\rho,\pi_\exists,\pi_\forall)\in \mathrm{Tiles}(\mathcal{K}),\\ (R,T',\pi'_\exists,\pi'_\forall)\in\rho}} N(\tau) \leq \sum_{\substack{\tau'=(T',\rho',\pi'_\exists,\pi'_\forall)\in \mathrm{Tiles}(\mathcal{K}),\\ (R^-,T,\pi_\exists,\pi_\forall)\in\rho'}} N(\tau')$$

We next define the sets $X_{T,T',\pi_\exists,\pi'_\exists,\pi_\forall,\pi'_\forall,R}$ and $Y_{T,T',\pi_\exists,\pi'_\exists,\pi_\forall,\pi'_\forall,R}$ as before, we get that $|X_{T,T',\pi_\exists,\pi'_\exists,\pi_\forall,\pi'_\forall,R}| \leq |Y_{T,T',\pi_\exists,\pi'_\exists,\pi_\forall,\pi'_\forall,R}|$.Thus, there exists an injection $g_{T,T',\pi_\exists,\pi'_\exists,\pi_\forall,\pi'_\forall,R}$ from $X_{T,T',\pi_\exists,\pi'_\exists,\pi_\forall,\pi'_\forall,R}$ onto $Y_{T,T',\pi_\exists,\pi'_\exists,\pi_\forall,\pi'_\forall,R}$. This injection $g_{T,T',\pi_\exists,\pi'_\exists,\pi_\forall,\pi'_\forall,R}$ has the same property as the bijection $f_{T,T',\pi_\exists,\pi'_\exists,\pi_\forall,\pi'_\forall,R}$ in the previous case. Once again, for every $e \in X_{T,T',\pi_\exists,\pi'_\exists,\pi_\forall,\pi'_\forall,R}$ and every $s \in R$, we set $(e, g_{T,T',\pi_\exists,\pi'_\exists,\pi_\forall,\pi'_\forall,R}(e)) \in s^{\mathcal{I}}$ if s is a role name and $(g_{T,T',\pi_\exists,\pi'_\exists,\pi_\forall,\pi'_\forall,R}(e), e) \in (s^-)^{\mathcal{I}}$ otherwise. We note that, if there is a role s s.t. $(\mathrm{func}\ s) \in \mathcal{T}$ and $(\mathrm{func}\ s^-) \in \mathcal{T}$ and $s \in R$, we let the injection $g_{T,T',\pi_\exists,\pi'_\exists,\pi_\forall,\pi'_\forall,R}$ be exactly the bijection $f_{T,T',R}$ from the previous case.

4. Finally, we do the following for each $e = (T, \rho, \pi_\exists, \pi_\forall)_i \in \Delta^\mathcal{I}$ with $(R, T', \pi'_\exists, \pi'_\forall) \in \rho$ for which there is no $d \in \Delta^\mathcal{I}$ such that $d = (T', \rho', \pi'_\exists, \pi'_\forall)$ and $(e, d) \in r^\mathcal{I}$, for all $r \in R$. Due to MF$^+$4, there exists some ρ' such that $\tau' = (T', \rho', \pi'_\exists, \pi'_\forall) \in \mathsf{Tiles}(\mathcal{K})$ and $N(\tau')) > 0$. Let e' denote the domain element $\tau'_1 \in \Delta^\mathcal{I}$. For each $r \in R$, if $r \in \mathrm{N}_R$ we set $(e, e') \in r^\mathcal{I}$, otherwise we set $(e', e) \in (r^-)^\mathcal{I}$. This concludes the construction of $\mathcal{I}$.

We note that the construction of the model $\mathcal{I}$ is virtually the same as in the previous section, the satisfaction of the ABox and standard $\mathcal{ALCHOIF}$ axioms in $\mathcal{I}$ under closed predicates is immediate and therefore omitted. We only show that $\mathcal{I}$ satisfies axioms of the form (A4)-(A7).

- α is of the shape $\mathsf{func}(r)$: Let $e = (T, \rho)_i$, $e_1 = (T_1, \rho_1)_j$, and $e_2 = (T_2, \rho_2)_k$ be elements in $\Delta^\mathcal{I}$ and assume that $(e, e_1) \in r^\mathcal{I}$ and $(e, e_2) \in r^\mathcal{I}$. Note that, steps 1-3 can together construct at most one r-arc from e to some other element, thus these steps alone cannot be responsible for the situation. The same holds for step 4. We conclude that the violation arises as follows. W.lo.g., assume that steps 1-3 construct $(e, e_1) \in r^\mathcal{I}$ and step 4 constructs $(e, e_2) \in r^\mathcal{I}$ due to a pair $(R_2, T_2) \in \rho$ with $r \in R_2$. Now, if $(e, e_1) \in r^\mathcal{I}$ was constructed in steps 2-3, then there must be also be a pair $(R_1, T_1) \in \rho$ with $r \in R_1$. Due to condition TF$^+$5, (R_1, T_1) and (R_2, T_2) must be the same pair. Thus, there already exists a witness for (R_2, T_2) and step 4. is never actually executed. On the other hand, if $(e, e_1) \in r^\mathcal{I}$ was constructed in step 1, then due to conditions TF$^+$8 (a) and TF$^+$8 (b) and MF$^+$1, we can conclude that $T_1 = T_2$. As e_1 represents a constant and is thus the only element with the type T_2, we can conclude that $e_1 = e_2$ and that there is no violation.

- α is of the shape $\exists A? \circ P.(\{x\} \sqcap \exists s.\{y\}) \sqcap \exists B? \circ R.\{x\} \sqsubseteq C$, where $A, B \in \mathsf{N}_\mathsf{C}^+$, $C \in \mathrm{N}_C$, $s \in \Sigma$ and P, R are complex roles starting with a role an consisting of only tests and functional roles:

  Let $e = (T_e, \rho_e, \pi_\exists, \pi_\forall) \in \Delta^\mathcal{I}$ and $\{a\}, \{b\}$ be nominals in $\mathsf{N}_\mathsf{C}^+(\mathcal{K})$ s.t. $e \in (\exists A \circ P.(\{a\} \sqcap \exists s.\{b\}) \sqcap \exists B \circ R.\{b\})^\mathcal{I}$. This means that $e \in A^\mathcal{I}, e \in B^\mathcal{I}$, $(e, a) \in P^\mathcal{I}$, $s(a, b) \in \mathcal{A}$, and $(e, b) \in R^\mathcal{I}$. Assume towards a contradiction that $e \notin C^\mathcal{I}$, i.e., $C \notin T$.

  We next show that $(e, a) \in P^\mathcal{I}$ implies that $(P, \{a\}) \in \pi_\exists$ by induction on the length of P.

  Base case: $\mathsf{len}(P) = 0$, i.e., $P = \varepsilon$.

  The only way $(e, a) \in \varepsilon^\mathcal{I}$, is if $e = a = (T_a, \rho_a, \pi_{\exists a}, \pi_{\forall a})_1$, where $\{a\} \in T_a$. By TF$^+$13, we have that $(\varepsilon, \{a\}) \in \pi_{\exists a}$.

  Induction step: The statement holds for all P of length n. We show that it also holds if $\mathsf{len}(P) = n + 1$, i.e, $P = r_{n+1} \circ A_{n+1}? \circ S$ where $\mathsf{len}(S) = n$.

By assumption $(e,a) \in P^{\mathcal{I}}$. This means that there is some $e' = (T', \rho', \pi'_\exists, \pi'_\forall)_j \in \Delta^{\mathcal{I}}$ s.t. $(e,e') \in (r_{n+1})^{\mathcal{I}}$, $e' \in (A_{n+1})^{\mathcal{I}}$ and $(e',a) \in S^{\mathcal{I}}$. We have that $A_{n+1} \in T'$. Moreover, by induction hypothesis, $(S, \{a\}) \in \pi'_\exists$.

We now do a case distinction based on how $(e,e') \in (r_{n+1})^{\mathcal{I}}$ was created.

- In Step 1, due to $p(e,e') \in \mathcal{A}$, $p \sqsubseteq r_{n+1} \in \mathcal{T}$. In that case, $e = c = (T_c, \rho_c, \pi_{\exists c}, \pi_{\forall c})_1$, where $\{c\} \in T_c$, and $e' = d = (T_d, \rho_d, \pi_{\exists d}, \pi_{\forall d})_1$, where $\{d\} \in T_d$, for some $\{c\}, \{d\} \in \mathsf{N}_\mathsf{C}^+(\mathcal{K})$. Recall that P consists only of functional roles and so r_{n+1} is a functional role. Thus, by TF^+19 (and by MF^+1, stating that there is only one realized tile with d in its unary type), there is some $(R, T_d, \pi_{\exists d}, \pi_{\forall d}) \in \rho$ s.t. $r_{n+1} \in R$ and $\{d\} \in T_d$. This means that $(R, T', \pi'_\exists, \pi'_\forall) \in \rho$, where $r_{n+1} \in R$, $A_{n+1} \in T'$, and $(S, \{a\}) \in \pi'_\exists$. Finally, by TF^+18 (a), we have $(r_{n+1} \circ A_{n+1}? \circ S, \{a\}) \in \pi_\exists$.
- In Step 1, due to $p(e',e) \in \mathcal{A}$, $p^- \sqsubseteq (r_{n+1}) \in \mathcal{T}$. Analogous to the previous case, but using the condition TF^+20 instead of TF^+19.

  In that case, $e = c = (T_c, \rho_c, \pi_{\exists c}, \pi_{\forall c})_1$, where $\{c\} \in T_c$, and $e' = d = (T_d, \rho_d, \pi_{\exists d}, \pi_{\forall d})_1$, where $\{d\} \in T_d$, for some $\{c\}, \{d\} \in \mathsf{N}_\mathsf{C}^+(\mathcal{K})$. Recall that P consists only of functional roles and so r_{n+1} is a functional role. Thus, by TF^+20 (and by MF^+1, stating that there is only one realized tile with d in its unary type), there is some $(R, T_d, \pi_{\exists d}, \pi_{\forall d}) \in \rho$ s.t. $r_{n+1} \in R$ and $\{d\} \in T_d$. This means that $(R, T', \pi'_\exists, \pi'_\forall) \in \rho$, where $r_{n+1} \in R$, $A_{n+1} \in T'$, and $(S, \{a\}) \in \pi'_\exists$. Finally, by TF^+18 (a), we have $(r_{n+1} \circ A_{n+1}? \circ S, \{a\}) \in \pi_\exists$.
- In Step 2-4, due to some $(R, T', \pi'_\exists, \pi'_\forall) \in \rho$ with $r_{n+1} \in R$. As $A \in T'$, we have by TF^+18 (a) that $(r_{n+1} \circ A_{n+1}? \circ S, \{a\}) \in \pi_\exists$.
- In Step 2-4, due to some $(R', T, \pi_\exists, \pi_\forall) \in \rho'$, with $(r_{n+1})^- \in R$. Since $A \in T'$, by TF^+18 (b), we have that $(r_{n+1} \circ A_{n+1}? \circ S, \{a\}) \in \pi_\exists$.

Thus, we have shown that $(P, \{a\}) \in \pi_\exists$, for all P of length $n+1$, which concludes our proof by induction.

The same argument also applies to show that $(S, \{b\}) \in \pi_\exists$. Finally, as $e \in A^{\mathcal{I}}$, i.e., $A \in T$, $e \in B^{\mathcal{I}}$, i.e., $B \in T$, $(P, \{a\}) \in \pi_\exists$, $(S, \{b\}) \in \pi_\exists$, and $s(a,b) \in \mathcal{A}$, we have, by TF^+14 that $C \in T$, which is a contradiction.

- α is of the shape $\exists A? \circ P.(\{x\} \sqcap \neg \exists s.\{y\}) \sqcap \exists B? \circ R.\{x\} \sqsubseteq C$, where $A, B \in \mathsf{N}_\mathsf{C}^+$, $C \in \mathrm{N}_C$, $s \in \Sigma$ and P, R are complex roles starting with a role an consisting of only tests and functional roles:

  Let $e = (T_e, \rho_e, \pi_\exists, \pi_\forall) \in \Delta^{\mathcal{I}}$ and $\{a\}, \{b\}$ be nominals in $\mathsf{N}_\mathsf{C}^+(\mathcal{K})$ s.t. $e \in (\exists A \circ P.(\{a\} \sqcap \neg \exists s.\{b\}) \sqcap \exists B \circ R.\{b\})^{\mathcal{I}}$. This means that $e \in A^{\mathcal{I}}, e \in B^{\mathcal{I}}$, $(e,a) \in P^{\mathcal{I}}$, $s(a,b) \notin \mathcal{A}$, and $(e,b) \in R^{\mathcal{I}}$. Assume towards a contradiction that $e \notin C^{\mathcal{I}}$, i.e., $C \notin T$.

Using the same argument from the case before, we have that $\{(P,\{a\}),(R,\{b\})\} \subseteq \pi_\exists(e) = \pi_\exists$. Finally, as $e \in A^\mathcal{I}$, i.e., $A \in T$, $e \in B^\mathcal{I}$, i.e., $B \in T$, $(P,\{a\}) \in \pi_\exists$, $(S,\{b\}) \in \pi_\exists$, and $s(a,b) \notin \mathcal{A}$, we have, by TF$^+$15 that $C \in T$, which is a contradiction.

- α is of the shape $\{x\} \sqsubseteq \forall P.\{x\}$, where P is a complex role

  Assume towards a contradiction that α is not satisfied in $\mathcal{I}$. This means that there is some nominal $\{a\} \in \mathsf{N}_\mathsf{C}^+(\mathcal{K})$ and some domain elements $a = (T_a, \rho_a, \pi_{\exists a}, \pi_{\forall a})_1$, $\{a\} \in T_a$, $e = (T, \rho, \pi_\exists, \pi_\forall)_i$, s.t., $a \in A^\mathcal{I}$, $(a,e) \in P^\mathcal{I}$ and $e \neq a$.

  As $a \in A^\mathcal{I}$, we have by construction that $A \in T_a$.

  We next prove that $(a,e) \in P^\mathcal{I}$ implies $(\varepsilon,\{a\}) \in \pi_\exists$.

  Assume $(a,e) \in P^\mathcal{I}$, where $P = r_1 \circ A_1? \circ \ldots r_n \circ A_n?$. That means that there exist $e_0 = a, e_1, \ldots, e_{n-1}, e_n = e \in \Delta^\mathcal{I}$ s.t. $e_i = (T_i, \rho_i, \pi_{\exists i}, \pi_{\forall i},$ and $(e_i, e_{i+1}) \in r_{i+1}$, $e_j \in (A_j)^\mathcal{I}$, for $0 \leq i \leq n-1$, $1 \leq j \leq n$.

  We next prove that for each e_i, $0 \leq i \leq n-1$, either (*) $(r_{i+1} \circ A_{i+1}? \ldots r_n \circ A_n? \circ \varepsilon, \{a\})$ or $(r_{i+1} \circ A_{i+1}? \ldots r_n \circ A_n? \circ \varepsilon, \bot)$ occurs in $\pi_{\forall i}$. Proof is given by induction on i:

  Base case: i=0: We know that $e_0 = a$, thus $T_0 = T_a$ and so $\{\{a\}, A\} \subseteq T_0$. By TF$^+$16, we have that either $(P,\{a\}) \in \pi_{\forall 0}$ or $(P,\bot) \in \pi_{\forall 0}$.

  Induction step Assume the statement (*) holds for all $1 \leq i \leq n-2$. We show that it holds for $i+1$.

  By assumption, (*) holds for e_i, and so $(r_{i+1} \circ A_{i+1}? \ldots r_n \circ A_n? \circ \varepsilon, \{a\})$ or $(r_{i+1} \circ A_{i+1}? \ldots r_n \circ A_n? \circ \varepsilon, \bot)$ occurs in $\pi_{\forall i}$. By assumption, we have that $(e_i, e_{i+1}) \in (r_{i+1})^\mathcal{I}$. We do a case distinction on when $(e_i, e_{i+1}) \in (r_{i+1})^\mathcal{I}$ was created.

  - In Step 1, due to $p(e_i, e_{i+1}) \in \mathcal{A}$, $p \sqsubseteq r_{i+1} \in \mathcal{T}$. In that case, $e_i = c = (T_c, \rho_c, \pi_{\exists c}, \pi_{\forall c})_1$, where $\{c\} \in T_c$, and $e_{i+1} = d = (T_d, \rho_d, \pi_{\exists d}, \pi_{\forall d})_1$, where $\{d\} \in T_d$, for some $\{c\}, \{d\} \in \mathsf{N}_\mathsf{C}^+(\mathcal{K})$. Recall also, that we assumed that $e_{i+1} \in (A_{i+1})^\mathcal{I}$, i.e., $A_{i+1} \in T_{i+1}$ and so $A_{i+1} \in T_d$ Now, due to MF$^+$6 c) and MF$^+$1, we can conclude that, due to $(r_{i+1} \circ A_{i+1}? \ldots r_n \circ A_n? \circ \varepsilon, C) \in \pi_{\forall c}$, where $C \in \{\{a\}, \bot\}$, that either $(r_{i+2} \circ A_{i+2}? \ldots r_n \circ A_n? \circ \varepsilon, C) \in \pi_{\forall c}$ or $(r_{i+2} \circ A_{i+2}? \ldots r_n \circ A_n? \circ \varepsilon, \bot) \in \pi_{\forall c}$.
  - In Step 1, due to $p(e_{i+1}, e_i) \in \mathcal{A}$, $p^- \sqsubseteq (r_{i+1}) \in \mathcal{T}$. Analogous to the previous case, but using the condition MF$^+$6 (d)
  - In Step 2-4, due to some $(R, T_{i+1}, \pi_{\exists i+1}, \pi_{\forall i+1}) \in \rho_i$ with $r_{i+1} \in R$. As $A_{i+1} \in T_{i+1}$, we have by TF$^+$18 (c) that $(r_{i+2} \circ A_{i+2}? \ldots r_n \circ A_n? \circ \varepsilon, C) \in \pi_{\forall i+1}$, where $C \in \{\{a\}, \bot\}$.

– In Step 2-4, due to some $(R', T_i, \pi_{\exists i}, \pi_{\forall i}) \in \rho_{i+1}$, with $(r_{i+1})^- \in R$. Since $A_{i+1} \in T_{i+1}$, by TF$^+$18 (d), we have that $(r_{i+2} \circ A_{i+2}? \dots r_n \circ A_n? \circ \varepsilon, C) \in \pi_{\forall i+1}$, where $C \in \{\{a\}, \bot\}$.

Thus, we have shown that for all $1 \leq i \leq n-1$, $(r_{i+1} \circ A_{i+1}? \dots r_n \circ A_n? \circ \varepsilon, \{a\})$ or $(r_{i+1} \circ A_{i+1}? \dots r_n \circ A_n? \circ \varepsilon, \bot)$ occurs in $\pi_{\forall i}$.

In particular, we have that $(r_n \circ A_n? \circ \varepsilon, \{a\})$ or $(r_n \circ A_n? \circ \varepsilon, C)$. As $A_n \in T_n$ and $(e_{n-1}, e_n) \in (r_n)^{\mathcal{I}}$, we have that $(\varepsilon, \{a\})$ or $(\varepsilon, \bot)$ in $\pi_{\forall n}$ (once again proven by case distinction on when $(e_{n-1}, e_n) \in (r_n)^{\mathcal{I}}$ was created. Now, due to TF$^+$17, if $(\varepsilon, C) \in \pi_{\forall n}$, then $C \in T_n$. As $\bot \notin T_n$ always holds, it must be that $C = \{a\}$ and $\{a\} \in T_n$, making $e = a$. This is a contradiction, and thus $\mathcal{I} \vDash \alpha$.

- α is of the shape $\{x\} \sqcap A \sqsubseteq \forall s.\neg\{x\}$, where $A \in \mathrm{N}_C$, $s \in \Sigma$

  Assume to the contrary, that there is some nominal $\{a\} \in \mathsf{N}_\mathsf{C}^+(\mathcal{K})$ s.t. $a \in A^{\mathcal{I}}$ and $(a, a) \in s^{\mathcal{I}}$. We have already shown that $\mathcal{I}$ respects closed predicates and thus $s(a, a) \in \mathcal{A}$. Now, recall that a nominal a is actually a domain element e_a obtained from the unique tile $\tau_a = (T_a, \rho_a, \pi_{\exists a}, \pi_{\forall a})$ s.t. $\{a\} \in T_a$. Due to the condition TF$^+$21, this means that $A \notin T_a$ and so $a \notin A^{\mathcal{I}}$, which is a contradiction.

As we have shown that $\mathcal{I} \vDash_\Sigma \mathcal{A}$ and $\mathcal{I} \vDash \mathcal{T}$, we have that $\mathcal{I} \vDash \mathcal{K}$ and thus $\mathcal{K}$ is satisfiable.

Reasoning with Safe RLPs

The correctness of both of Algorithm 7.1 and Algorithm 7.3 relies on the following results:

Lemma 1. *Let $\mathcal{P}$ be a Datalog$^\neg$ and let $\mathcal{B}, \Sigma \subseteq \mathsf{N_P}$. Further, assume that $\mathcal{P}$ fulfills the following condition:*

$$\text{for each } \rho \in \mathcal{P} \text{ every variable in } \rho \text{ occurs in some } p(\vec{u}) \in body^+(\rho),\ p \in \mathcal{B}. \qquad (*)$$

Given an Herbrand interpretation I, let J be a minimal model of $\mathcal{P}^{I,\Sigma}$. For each $c \in \mathsf{N_I}$ that occurs in J, either $c \in adom(\mathcal{P})$ or c occurs in some $q(\vec{c}) \in I$, where $q \in \mathcal{B} \cap \Sigma$. Additionally, if for some $1 \leq i \leq art(q_1)$, c occurs in position $q_1[i]$ in some atom $q_1(\vec{c}_1) \in J$ s.t. $q_1[i] \notin \mathsf{ap}(\mathcal{P}, \mathcal{B} \cap \Sigma)$, then $c \in adom(\mathcal{P})$.

Proof. By definition $\mathcal{P}^{I,\Sigma}$ is a ground positive program and so each atom in J must have a derivation from the facts of $\mathcal{P}^{I,\Sigma}$ using its rules. Let c be the constant in the i-th position of some $q_1(\vec{c})_1 \in J$. Both statements of the lemma can be shown using induction on the length of the derivation of $q_1(\vec{c})$. We only show the second. To this end, assume that $q_1[i] \notin \mathsf{ap}(\mathcal{P}, \mathcal{B} \cap \Sigma)$.

Base case: $q_1(\vec{c}_1)$ is a fact in $\mathcal{P}^{I,\Sigma}$. Then, this atom must have been obtained from some rule $\rho \in \mathit{ground}(\mathcal{P}, \mathsf{N_I})$ of the following form:

$$\rho : q_1(\vec{c}_1) \leftarrow q_2(\vec{c}_2), \ldots, q_m(\vec{c}_m), \mathit{not}\ q_{m+1}(\vec{c}_{m+1}), \ldots, \mathit{not}\ q_l(\vec{c}_l),$$

s.t. $\{q_2(\vec{c}_2), \ldots, q_m(\vec{c}_m)\} \subseteq I$, $\{q_{m+1}(\vec{c}_{m+1}), \ldots, q_l(\vec{c}_l)\} \cap I = \emptyset$, and $\{q_2, \ldots, q_m\} \subseteq \Sigma$. In turn, ρ was obtained from some $\rho' \in \mathcal{P}$ of the form:

$$\rho' : q_1(\vec{t}_1) \leftarrow q_2(\vec{t}_2), \ldots, q_m(\vec{t}_m), \mathit{not}\ q_{m+1}(\vec{t}_{m+1}), \ldots, \mathit{not}\ q_l(\vec{t}_l),$$

where each t in position $q_j[k]$ in $q_j(\vec{t}_j)$ is either a variable or the constant in position $q_j[k]$ in $q_j(\vec{c})$, for $1 \leq j \leq l$, $1 \leq k \leq \mathit{art}(q_j)$.

If c is in position $q_1[i]$ in $q_1(\vec{t}_1)$, then obviously $c \in \mathit{adom}(\mathcal{P})$. Otherwise, there must be a variable in position $q_1[i]$ in $q_1(\vec{t}_1)$. Due to condition $(*)$, this variable would have to occur in some $\vec{t}_j$, $1 \leq j \leq m$ s.t. $q_j \in \mathcal{B}$. However, as $q_j \in \Sigma$, for all $2 \leq j \leq m$, this would make the position $q[i]$ affected in $\mathcal{P}$ w.r.t. Σ, which contradicts our assumption. Hence, the only option is that $c \in \mathit{adom}(\mathcal{P})$.

Step: Assume that every constant that occurs in a position unaffected in $\mathcal{P}$ w.r.t. Σ in some atom in J with a derivation of length $\leq n$ also occurs in $\mathit{adom}(\mathcal{P})$. If $q_1(\vec{c}_1)$ has a derivation of length n then it was obtained using some rule $\rho \in \mathcal{P}^{I,\Sigma}$ of the form

$$\rho : q_1(\vec{c}_1) \leftarrow q_2(\vec{c}_2), \ldots, q_m(\vec{c}_m),$$

where each $q_j(\vec{c}_j)$, $2 \leq j \leq m$, has a derivation of length less than n. Then, ρ must have been obtained from some $\rho' \in \mathit{ground}(\mathcal{P}, \mathsf{N_I})$:

$$\rho' : q_1(\vec{c}_1) \leftarrow q_2(\vec{c}_2), \ldots, q_m(\vec{c}_m), q_{m+1}(\vec{c}_{m+1}), \ldots, q_l(\vec{c}_l), \mathit{not}\ q_{l+1}(\vec{c}_{l+1}), \ldots, \mathit{not}\ q_h(\vec{c}_h),$$

where $\{q_{m+1}(\vec{c}_{m+1}), \ldots, q_l(\vec{c}_l)\} \subseteq I$, $\{q_{l+1}(\vec{c}_{l+1}), \ldots, q_h(\vec{c}_h)\} \cap I = \emptyset$, and $\{q_{m+1}, \ldots, q_l\} \in \Sigma$.

This rule was in turn obtained from some $\rho'' \in \mathcal{P}$ s.t.

$$\rho'' : p(\vec{t}_1) \leftarrow q_2(\vec{t}_2), \ldots, q_m(\vec{t}_m), q_{m+1}(\vec{t}_{m+1}), \ldots, q_l(\vec{t}_l), \mathit{not}\ q_{l+1}(\vec{t}_{l+1}), \ldots, \mathit{not}\ q_h(\vec{t}_h),$$

where each t in position $q_j[k]$ in $q_j(\vec{t}_j)$ is either a variable or the constant in position $q_j[k]$ in $q_j(\vec{c})$, for $1 \leq j \leq h$, $1 \leq k \leq \mathit{art}(q_j)$.

As before, if c is in position $q_1[i]$ in $q_1(\vec{t}_1)$, then obviously $c \in \mathit{adom}(\mathcal{P})$. Otherwise, there must be a variable x in position $q_1[i]$ in $q_1(\vec{t}_1)$. Due to condition $(*)$, x would have to occur in some $\vec{t}_j$ s.t. $q_j \in \mathcal{B}$, $2 \leq j \leq l$. If $m+1 \leq j \leq l$, then $q_j \in \Sigma$ and so this would make the position $q[i]$ affected in $\mathcal{P}$ w.r.t. Σ, which contradicts our assumption. Similarly, if $2 \leq j \leq m$ and the position where x occurs in $q_j(\vec{t}_j)$ is affected in $\mathcal{P}$ w.r.t. Σ, then $q_1[i]$ would be affected as well, which once again contradicts our assumption. Hence, it must be that the position where x occurs in q_j is unaffected, and so the constant c' occurring in this position in $\vec{c}_j$ is in $\mathit{adom}(\mathcal{P})$. As, $c' = c$, this means that $c \in \mathit{adom}(\mathcal{P})$. □

As a corollary of the lemma above we get the following two results:

Proposition 1. *Let* $\Pi = (\mathcal{P}, \mathcal{T}, \Sigma_{\mathsf{out}}, \Sigma_{\mathsf{owa}}, \Sigma_{\mathsf{re}})$ *be an RLP and let* I *be an answer set of* Π*. For every* $c \in \mathsf{N_I}$ *occurring in* I*,* $c \in adom(\mathcal{P})$ *holds.*

Proposition 2. *Assume a* $\mathsf{Datalog}^{\neg}$ *program* $\mathcal{P}$ *s.t.* $(*)$ *from Lemma 1 holds for some sets* $\mathcal{B}$ *and* Σ *of predicate symbols, and interpretations* I, J*. Then* J *is a minimal model of* $\mathcal{P}^{I,\Sigma}$ *iff* J *is a minimal model of* $ground(\mathcal{P}, C)^{I,\Sigma}$*, where* C *is the set of constants from* $adom(\mathcal{P})$ *and those that occur in some* $q(\vec{c}) \in I$ *with* $q \in \mathcal{B} \cap \Sigma$*.*

Lemma 2. *Assume a* $\mathsf{Datalog}^{\neg}$ *program* $\mathcal{P}$ *that fulfills condition* $(*)$ *from Lemma 1 for two sets* $\mathcal{B}$ *and* Σ *of predicate symbols. Let* I *be an interpretation and let* $\Delta = adom(\mathcal{P}) \cup \{c : c \text{ occurs in some } q(\vec{c}) \in I, q \in \mathcal{B} \cap \Sigma\}$*. Then,* J *is a minimal model of* $P^{I,\Sigma}$ *iff* J *is a minimal model of* $P^{I',\Sigma}$*, where* $I' = \{p(\vec{c}) \in I : \vec{c} \in \Delta^{art(p)}\}$*.*

Proof. Consider a rule $\rho \in \mathcal{P}^{I,\Sigma}$ s.t. $\rho \notin \mathcal{P}^{I',\Sigma}$. Then, there must be some atom $p(\vec{c}) \in body^{+}(\rho)$ s.t. $p(\vec{c}) \in I$ and $p(\vec{c}) \notin I'$. Hence, $\vec{c}$ must contain a constant c that is not in Δ. Then c was then obtained from a variable by grounding $\mathcal{P}$, and hence, due to $(*)$, there must be a positive atom $q(\vec{d})$ containing c s.t. either (i) $q \in \mathcal{B} \cap \Sigma$ and $q(\vec{d}) \in I$ or (ii) $q \in \mathcal{B} \setminus \Sigma$ and $q(\vec{d}) \in body^{+}(\rho)$. In the first case, c would occur in Δ which is a contradiction. In the second case, we have that due to Lemma 1, this atom cannot be contained in a minimal model of $P^{I,\Sigma}$ and so this rule does not participate.

Similarly, consider a rule $\rho' \in \mathcal{P}^{I',\Sigma}$ s.t. $\rho' \notin \mathcal{P}^{I,\Sigma}$. Then this rule cannot participate in the computation of the minimal model of $\mathcal{P}^{I',\Sigma}$.

Hence, the rules that are relevant for determining the minimal model of $P^{I,\Sigma}$ and $P^{I',\Sigma}$ are the same for both programs. It follows that they have the same minimal model. □

From Lemma 2 we get the following result:

Proposition 3. *Let* $\Pi = (\mathcal{P}, \mathcal{T}, \Sigma_{\mathsf{out}}, \Sigma_{\mathsf{owa}}, \Sigma_{\mathsf{re}})$ *be an RLP ,* I *and* J *be interpretations over* Σ *and* $\mathsf{sig}(\mathcal{T})$*, respectively. Let* $J' = \{p(\vec{c}) \in J : p \in \Sigma_{\mathsf{owa}} \cap \mathsf{sig}(\mathcal{P}), \vec{c} \in \Delta^{art(p)}\}$*, where* Δ *is the set of constants that occur in* $\mathcal{P}$ *or in some* $q(\vec{a}) \in J$ *with* $q \in \boldsymbol{B}_{cn}(\mathcal{T}, \Sigma_{\mathsf{out}})$*. Then* J *and* J' *have the same responses w.r.t.* I *and* Π*.*

Based on the results above, we can now give a formal proof of Theorem 7.3.4.

Proof sketch. Due to Proposition 1, the maximum number of different constants occurring in some answer set of Π is bounded by $|adom(\mathcal{P})|$. In view of Proposition 7.3.2, we can thus compute an integer b that limits the amount of different constants that can be in the extension of any concept name in $\mathbf{B}_{\mathrm{cn}}(\mathcal{T}, \Sigma_{\mathsf{out}})$ in any model of $\mathcal{T}$ that agrees with some answer set of Π on Σ_{out}, referred to as a *relevant model* of $\mathcal{T}$. We construct a set Δ of size b consisting of $adom(\mathcal{P})$, constants occurring in $\mathcal{T}$ and additionally of fresh constants giving a name to each anonymous element that could hypothetically occur in

the extension of some $A \in \mathbf{B}_{\mathrm{cn}}(\mathcal{T}, \Sigma_{\mathsf{out}})$ in some relevant model of $\mathcal{T}$. Checking whether there are models of $\mathcal{T}$ that agree with I on Σ_{out} is a well-known problem in the literature and it boils down to checking the consistency of a description logic knowledge base $(\mathcal{T}, I)$ in the presence of closed predicates from Σ_{out}, where I is seen as an ABox. In view of Proposition 3, to find a model of $\mathcal{T}$ with no response we only guess the extensions of open predicates that occur in $\mathcal{P}$ using the constants from Δ. We next check whether our partial guess J actually corresponds to some model of $\mathcal{T}$ and if so, we check whether there is a response H to J. Due to our safety condition, for this guess it suffices to consider only the constants from J and $adom(\mathcal{P})$. Note that to verify whether H is a response to J, we need not fully compute the possibly infinite reduct $\mathcal{P}^{H,\Sigma_{\mathsf{owa}}}$. Instead, we use the reduct $ground(\mathcal{P}, \Delta)^{H,\Sigma_{\mathsf{owa}}}$ which is guaranteed to be finite and has the same minimal model as the original reduct. □