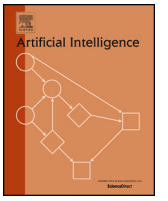
Artificial Intelligence 330 (2024) 104099

Contents lists available at ScienceDirect

Artificial Intelligence

journal homepage: www.elsevier.com/locate/artint



Datalog rewritability and data complexity of $\mathcal{ALCHOIQ}$ with closed predicates

Sanja Lukumbuzya [a,*], Magdalena Ortiz [a,b], Mantas Šimkus [a,b]

[a] Institute of Logic and Computation, TU Wien, Austria
[b] Department of Computing Science, Umeå University, Sweden

ARTICLE INFO

Keywords:
Description logics
Closed predicates
Datalog
Query rewriting

ABSTRACT

We study the relative expressiveness of ontology-mediated queries (OMQs) formulated in the expressive Description Logic $\mathcal{ALCHOIQ}$ extended with closed predicates. In particular, we present a polynomial time translation from OMQs into Datalog with negation under the stable model semantics, the formalism that underlies Answer Set Programming. This is a novel and non-trivial result: the considered OMQs are not only non-monotonic, but also feature a tricky combination of nominals, inverse roles, and counting. We start with atomic queries and then lift our approach to a large class of first-order queries where quantification is "guarded" by closed predicates. Our translation is based on a characterization of the query answering problem via integer programming, and a specially crafted program in Datalog with negation that finds solutions to dynamically generated systems of integer inequalities. 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. Thus, answering these OMQs in the presence of closed predicates is not harder than answering them in the standard setting. This is not obvious as closed predicates are known to increase data complexity for some existing ontology languages.

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. Unfortunately, accomplishing this is anything but an easy task, as the collected data is often incomplete, lacks structure, and suffers from integration problems due to semantic heterogeneity. 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 and structure back to the data, which can further be exploited for integration and completion purposes. Given these benefits, it is not surprising that ontologies have become an integral part of many modern applications, finding employment in a multitude of areas, including biomedicine and health care, governmental and financial domains, and, perhaps most widely known, the Semantic Web.

Description logics (DLs) [1] are a prominent family of formalisms specifically tailored for expressing structured knowledge. Backed up by years of active research that resulted in efficient DL reasoners along with a relatively good understanding of the computational properties of these languages, description logics are often the number one choice for building and reasoning about ontologies. For

* Corresponding author.
E-mail addresses: lukumbuzya@kr.tuwien.ac.at (S. Lukumbuzya), ortiz@kr.tuwien.ac.at (M. Ortiz), simkus@dbai.tuwien.ac.at (M. Šimkus).

https://doi.org/10.1016/j.artint.2024.104099
Received 1 February 2023; Received in revised form 29 November 2023; Accepted 19 February 2024

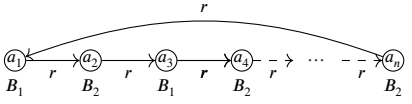
Available online 23 February 2024

instance, the Web Ontology Language (OWL) family proposed by W3C for defining ontologies on the Web [2,3] is based on description logics. DLs also play a key role in *Ontology-Based Data Access (OBDA)* [4], a data integration approach where a DL-based *ontology* in combination with a reasoner acts as a mediator between the users and possibly heterogeneous data sources. A key reasoning task in OBDA is answering *ontology-mediated queries (OMQs)*. An OMQ is a pair $(\mathcal{T}, Q)$ that couples an ontology $\mathcal{T}$ (or a *TBox*) expressed in some DL and a database query Q. Given a set $\mathcal{A}$ of facts (or an *ABox*), we compute the *certain answers* to $(\mathcal{T}, Q)$ over $\mathcal{A}$ by computing the intersection of answers to Q over all structures (essentially, databases) that contain $\mathcal{A}$ and satisfy the ontological axioms in $\mathcal{T}$. Most DLs are fragments of first-order logic (FOL) [5] which leads to the *open-world assumption* in OBDA and other applications of DLs to data management. Intuitively, the open-world assumption states that things that are not known to be true are not considered false, but rather unknown. This assumption is appropriate when data is assumed to be incomplete. However, this sometimes rules out intuitive common-sense inferences, and thus has spurred research efforts to combine the open-world assumption with the *closed-world assumption* that is made in the classical database setting, see, e.g., [6–9]. One of the basic tools to achieve this in DLs are so-called *closed predicates* [10–12], which allow us to specify which ontological predicates have extensions that are fully specified by the given data. In this paper, our focus is on the description logic $\mathcal{ALCHOIQ}$ extended with closed predicates. In this language, ontologies consist of two components: the TBox containing terminological axioms expressed in regular $\mathcal{ALCHOIQ}$ and the set Σ of predicates that are considered closed and, as such, determined solely by the given data. It is worth noting that $\mathcal{ALCHOIQ}$ itself is a close relative of the W3C-standardized OWL 2 ontology language and is among the most expressive standard DLs, extending the basic description logic $\mathcal{ALC}$ with role hierarchies, nominals, inverse roles, and qualified number restrictions. The combination of the last three constructors is known to be particularly tricky, even if the integers in the axioms are restricted to at most one. This is because their interaction can result in the creation of infinitely many *new nominals*. These domain elements are anonymous in the sense that they do not have a name, but they are nonetheless uniquely identifiable in any model of the considered ontology. 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 in $\mathcal{ALCHOIQ}$ [13]. For comparison, this problem is in ExpTime if any of the three constructors is omitted.

Adding closed predicates to $\mathcal{ALCHOIQ}$ results in a very powerful language whose nonmonotonic nature allows us to easily 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* – given a database and a unary relation A, is the number of objects in A odd? This query can be easily expressed through a Boolean OMQ $(\mathcal{T}_1, \bot)$, where the TBox $\mathcal{T}_1$ consists of the following axioms:

$$A \equiv B_1 \sqcup B_2 \quad B_1 \sqcap B_2 \sqsubseteq \bot \quad B_1 \sqsubseteq \geq 1 r.B_2 \quad B_2 \sqsubseteq \geq 1 r.B_1 \quad \top \sqsubseteq \leq 1 r.\top \quad \top \sqsubseteq \leq 1 r^-.\top$$

and A is closed. Let $a_1, \ldots, a_n$ be the elements in A, as given by some database $\mathcal{A}$. 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 $\mathcal{T}_1$ 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. Other examples of 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. As a final remark on the expressiveness of our language, we note that it has been shown that in DLs that contain $\mathcal{ALC}$, closed predicates can be simulated with the help of nominals [14]. This is rather simple and it involves hard coding the extensions of closed predicates from the ABox into the TBox. However, this approach assumes that the ABox is already given and is therefore not suitable for answering OMQs in a data-independent way, where we assume that the TBox and the query are fixed and only the data varies.

The main motivation behind this paper is to understand the *relative expressiveness* of OMQs whose ontology component is written in $\mathcal{ALCHOIQ}$ with closed predicates, especially compared to more standard database languages like various extensions of Datalog. This is usually done by means of *rewritings* (or *translations*), which are procedures that take as input an OMQ Q in the source language and produce 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 $\mathcal{A}$. In the related work section we briefly discuss various results on OMQ rewritings from the literature.

The key contribution of our paper is a polynomial-time translation that allows us to rewrite a large class of queries mediated by ontologies expressed in $\mathcal{ALCHOIQ}$ with closed predicates into Datalog$^\neg$, an extension of Datalog that allows the use of negation under the stable model semantics. This translation is done in a couple of steps:

- We first characterize the satisfiability problem in $\mathcal{ALCHOIQ}$ with closed predicates as a system of integer linear inequalities with some side conditions. This means that, given any ontology $(\mathcal{T}, \Sigma)$, where $\mathcal{T}$ is expressed in $\mathcal{ALCHOIQ}$ with closed predicates Σ and a DL ABox $\mathcal{A}$ containing some data, we can obtain a system of integer linear inequalities and implications that has a solution (over natural numbers extended with a special value that represents infinity) if and only if $(\mathcal{T}, \Sigma, \mathcal{A})$ is satisfiable.
- We then show how to construct, given a TBox $\mathcal{T}$ and a set Σ of closed predicates, 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

polynomial in the size of $(\mathcal{T}, \Sigma)$ and it consists of two components that do the following. Based on the characterization from the previous step, the first component computes the desired system of integer linear inequalities and implications for $(\mathcal{T}, \Sigma, \mathcal{A})$ and the second component solves the computed system.

- Next, 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 (IQs) mediated by $\mathcal{ALCHOIQ}$ ontologies with closed predicates. We also introduce a large class of so-called *safe-range OMQs*, which support first-order (FO) 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 [15]. 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 [16] 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 [17]. 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.

This article extends the results presented in [18], and to some extent, those presented in [19]. In particular, in [18], we used a similar integer-programming technique to establish Datalog$^{\neg}$-rewritability and data complexity of $\mathcal{ALCHOIF}$ with closed predicates, the fragment of $\mathcal{ALCHOIQ}$ in which integers in number restrictions can be at most one. The same paper also introduces safe-range ontology-mediated queries and establishes co-NP completeness of the query answering problem, when the corresponding TBox is written in $\mathcal{ALCHOIF}$. The topic of [19] deals with a different problem of determining whether extensions of some predicates are bounded in size, which is not discussed in this article. However, in that paper (and partly in its technical appendix) we introduce the integer programming encoding of the satisfiability problem for $\mathcal{ALCHOIQ}$ with closed predicates and we show that the satisfiability problem in this logic is NP-complete. In this article, we provide explanations, supporting examples, and detailed proofs of all claims made in [19] that relate to encoding the satisfiability problem in $\mathcal{ALCHOIQ}$ with closed predicates as an enriched system of integer linear inequalities. Moreover, we lift the Datalog$^{\neg}$ rewriting from [18] to this richer logic, and present the full encoding and intuitions behind the individual rules along the way.

1.1. Related work

Query rewriting Rewriting one query language into another is a popular technique for comparing expressive powers of languages and analyzing their theoretical properties. In the case of ontology-mediated queries, the motivation behind such rewritings also has an obvious practical component. Database technology is mature and efficient. Thus, rewriting OMQs into well-established query languages, like first-order logic or Datalog, opens up the possibility to reuse these highly-optimized query evaluation algorithms for answering ontological queries. There is a plethora of results in the DL literature concerning rewritings of OMQs targeting these traditional database query languages. For instance, for the members of the DL-Lite family, FO-rewritings were proposed already at the time of their introduction [20–22]. This established a very low data complexity of these lightweight DLs and has been the main reason behind their practical success – versions of DL-Lite have been employed for constructing many real-life ontologies for practical applications and are the languages of choice for the two most popular OBDA systems, Mastro [23] and Ontop [24]. The initial FO-rewritings had one major pitfall though – the target query could easily become exponential in the size of the original OMQ. This generated interest in finding succinct rewritings for DL-Lite. For example, it was shown in [25] that one can rewrite conjunctive queries (CQs) mediated by DL-Lite$_{\mathcal{R}}$ ontologies into nonrecursive Datalog programs in polynomial time, and in [26] that the crucial aspect of this translation is the assumption that each instance of the data contains a fixed number of distinct constants.

In the presence of closed predicates, only very few FO-rewritability results are available: the result in [27,12] showing that quantifier free unions of conjunctive queries in DL-Lite$_{\mathcal{R}}$ with closed predicates are FO-rewritable, and the one in [14] showing how to rewrite instance queries mediated by $\mathcal{ALC}$ ontologies with closed predicates into FO queries (when possible). This is not surprising, as it has been shown that closed predicates cause a sharp increase in the data complexity of query answering even in the simplest of settings. For example, it was shown in [10] that adding closed predicates to the core fragment of DL-Lite makes answering conjunctive queries co-NP-hard, and a more fine grained complexity analysis of answering OMQs with closed predicates was done in [12] with a focus on separating tractable from intractable cases for the logics DL-Lite$_{\mathcal{R}}$ and $\mathcal{EL}$ and $\mathcal{ALCHI}$. We remark that, under the standard complexity assumptions, FO-rewritability is ruled out for any DL whose data complexity of standard reasoning tasks is higher than AC^0.

Fortunately, many expressive DLs can be rewritten into some variant of Datalog. One of the earliest papers on this topic shows that instance queries mediated by ontologies written in the expressive DL $\mathcal{SHIQ}$ can be written as programs in disjunctive Datalog [28]. It was then shown in [29] that conjunctive queries can be rewritten into plain Datalog when mediated by ontologies that are expressed in the Horn fragment of $\mathcal{SHIQ}$, and in [30] that the same hold for a certain class of non-Horn $\mathcal{SHI}$ ontologies. Among other results, it was shown in [31] that CQs mediated by $\mathcal{ALC}$ ontologies can be rewritten into monadic disjunctive Datalog. However, all of these rewritings are exponential in size. The first succinct rewriting into Datalog for an expressive DL was proposed in [32] for IQs mediated by Horn-$\mathcal{ALCHOIQ}$ ontologies. More recently, [33] presented the first and, until now, only polynomial rewriting for an expressive DL in the presence of closed predicates. This last result shows that IQs mediated by $\mathcal{ALCHOI}$ ontologies with closed

predicates can be expressed as polynomial programs written in a non-monotonic variant of Datalog. Our translation targets the same non-monotonic variant of Datalog but uses a very different approach, as our chosen DL does not possess the convenient forest-model property that is exploited in [33]. Namely, we characterize models of $\mathcal{ALCHOIQ}$ KBs with closed predicates by means of integer programming. We then present a Datalog program that performs ontology-mediated query answering by computing solutions to a dynamically generated system of inequalities.

Integer programming techniques in DLs Our approach was inspired by works on finite model reasoning in DLs. Building on the technique from [34], Calvanese shows in [35] 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 [36] 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 [37] 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, [38] shows that both the finite and the unrestricted satisfiability problem for the two-variable fragment of FOL with counting quantifiers is decidable in NExpTime using a reduction to integer programming.

1.2. Organization

This paper is structured as follows. Section 2 reviews some basic notions in DLs and introduces the normal form that is used in the remainder of the paper. Section 3 presents our characterization of the satisfiability problem in $\mathcal{ALCHOIQ}$ with closed predicates as a system of integer linear inequalities with side conditions. For easier readability, this section is divided in four distinct parts. In Section 3.1, we introduce the notion of chromatic models and show that we can safely restrict our attention to only such models. We then present in Section 3.2 the actual characterization of the considered satisfiability problem in terms of 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 3.3 the correctness of this characterization. Finally, a connection to integer programming is made in Section 3.4. Section 4 briefly recalls relevant Datalog preliminaries and shows how to construct a Datalog program with negation under the stable model semantics for deciding this satisfiability problem. Finally, in Section 5, 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 6.

2. Description logic preliminaries

This section introduces the expressive description logic $\mathcal{ALCHOIQ}$ with closed predicates that is investigated in the remainder of this paper.

2.1. $\mathcal{ALCHOIQ}$ with closed predicates

Syntax We assume countably infinite and mutually disjoint sets $\mathsf{N_C}$, $\mathsf{N_R}$, and $\mathsf{N_I}$ of *concept names*, *role names*, and *constants*, respectively. A *role* is an expression of the form p and p^-, where $p \in \mathsf{N_R}$. 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 $p \in \mathsf{N_R}$. We say that the role r^- is the *inverse* of r. Given a set R of roles, R^- denotes the set $\{r^- : r \in R\}$. *Concepts* are defined according to the following syntax:

$$C := \top \mid \bot \mid A \mid \{a\} \mid \neg C \mid C \sqcup C \mid C \sqcap C \mid \exists r.C \mid \forall r.C \mid \leq nr.C \mid \geq nr.C \mid = nr.C,$$

where $A \in \mathsf{N_C}$, r is a role, $a \in \mathsf{N_I}$, and $n \geq 0$. Concepts of the form $\{a\}$, where $a \in \mathsf{N_I}$, are *nominals*. 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*. An expression of the form $C \sqsubseteq D$, where C and D are concepts, is a *concept inclusion*, and an expression of the form $r \sqsubseteq s$, where r and s are roles, is a *role inclusion*. A *(TBox) axiom* is a concept inclusion or a role inclusion, and a *TBox* is a finite set of axioms. An *ABox* is a finite set of *assertions* of the form $A(a)$, $\neg A(a)$, $p(a,b)$ or $\neg p(a,b)$, where $a, b \in \mathsf{N_I}$, $A \in \mathsf{N_C}$, and $p \in \mathsf{N_R}$. Given $\Sigma \subseteq \mathsf{N_C} \cup \mathsf{N_R}$, $\mathcal{A}|_\Sigma$ denotes the restriction of $\mathcal{A}$ to the concept and role names in Σ. If Σ consists of a single concept or role name q, we simply write $\mathcal{A}|_q$ to denote $\mathcal{A}|_\Sigma$. A *knowledge base (with closed predicates)* (KB) is a triple $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$, where $\mathcal{T}$ is a TBox, $\Sigma \subseteq \mathsf{N_C} \cup \mathsf{N_R}$ is a set of closed predicates in $\mathcal{K}$ and $\mathcal{A}$ is an ABox.

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 $\mathcal{X}$, by $\mathsf{N_C^+}(\mathcal{X})$ the set of basic concepts occurring in $\mathcal{X}$, and by $\mathsf{N_R^+}(\mathcal{X})$ the set of roles occurring in $\mathcal{X}$ and their inverses.

Semantics As $\mathcal{ALCHOIQ}$ is a fragment of first-order logic with a special syntax, the semantics is given in term of first-order interpretations. 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 extension of the interpretation function to the remaining concepts and roles is defined in the standard way and summarized in Table 1.

Table 1
Semantics of $\mathcal{ALCHOIQ}$ concepts and roles.

$(r^-)^{\mathcal{I}}$	=	$\{(d_1, d_2) : (d_2, d_1) \in r^{\mathcal{I}}\}$
$\top^{\mathcal{I}}$	=	$\Delta^{\mathcal{I}}$
$\bot^{\mathcal{I}}$	=	$\emptyset$
$\{a\}^{\mathcal{I}}$	=	$a^{\mathcal{I}}$
$(\neg C)^{\mathcal{I}}$	=	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
$(C_1 \sqcup C_2)^{\mathcal{I}}$	=	$C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
$(C_1 \sqcap C_2)^{\mathcal{I}}$	=	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
$(\exists r.C)^{\mathcal{I}}$	=	$\{d_1 \in \Delta^{\mathcal{I}} :$ for some $d_2 \in \Delta^{\mathcal{I}}, (d_1, d_2) \in r^{\mathcal{I}}$ and $d_2 \in C^{\mathcal{I}}\}$
$(\forall r.C)^{\mathcal{I}}$	=	$\{d_1 \in \Delta^{\mathcal{I}} :$ for all $d_2 \in \Delta^{\mathcal{I}}, (d_1, d_2) \in r^{\mathcal{I}}$ implies $d_2 \in C^{\mathcal{I}}\}$
$(?nr.C)^{\mathcal{I}}$	=	$\{d_1 \in \Delta^{\mathcal{I}} : \lvert\{d_2 \in \Delta^{\mathcal{I}} : (d_1, d_2) \in r^{\mathcal{I}}$ and $d_2 \in C^{\mathcal{I}}\}\rvert \, ? \, n\}$, for $? \in \{\leq, \geq, =\}$.

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 $(V, E, \mathcal{L})$ where $V = \Delta^{\mathcal{I}}$, the set E of arcs is defined as $E = \{(d_1, d_2) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} : (d_1, d_2) \in r^{\mathcal{I}}$ 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:

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

Let $\mathcal{I}$ be an interpretation. We say that $\mathcal{I}$ *satisfies* a concept or a role inclusion $Q_1 \sqsubseteq Q_2$, in symbols $\mathcal{I} \models Q_1 \sqsubseteq Q_2$, if $Q_1^{\mathcal{I}} \subseteq Q_2^{\mathcal{I}}$. Furthermore, $\mathcal{I}$ satisfies an assertion $q(\mathbf{a})$, in symbols $\mathcal{I} \models q(\mathbf{a})$, if $(\mathbf{a})^{\mathcal{I}} \in q^{\mathcal{I}}$ and $\mathcal{I}$ satisfies an assertion $\neg q(\mathbf{a})$ if $\mathcal{I} \not\models q(\mathbf{a})$. We say that $\mathcal{I}$ satisfies a TBox or an ABox $\mathcal{X}$, in symbols $\mathcal{I} \models \mathcal{X}$, if it satisfies all axioms or assertions in $\mathcal{X}$. Finally, $\mathcal{I}$ satisfies a knowledge base with closed predicates $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$, in symbols $\mathcal{I} \models \mathcal{K}$, if the following holds:

(i) for each $a \in \mathsf{N}_{\mathsf{I}}$ occurring in $\mathcal{K}$, $a \in \Delta^{\mathcal{I}}$ and $a^{\mathcal{I}} = a$,
(ii) $\mathcal{I} \models \mathcal{T}$ and $\mathcal{I} \models \mathcal{A}$
(iii) $\{q(\mathbf{a}) \ : \ q \in \Sigma, (\mathbf{a})^{\mathcal{I}} \in q^{\mathcal{I}}\} = \mathcal{A}|_{\Sigma}$.

Note that we make the *Standard Name Assumption (SNA)*, which is common when dealing with closed predicates and which forces us to interpret every constant occurring in the knowledge base as itself (condition (i)). An interpretation that satisfies $\mathcal{K}$ is called a *model of* $\mathcal{K}$ and we say that $\mathcal{K}$ is *satisfiable* if it has a model. The problem of deciding whether $\mathcal{K}$ is satisfiable is called the *satisfiability problem* and it is one of the standard reasoning tasks in DLs.

2.2. Normal form

The results obtained in this paper hold for arbitrary knowledge bases, however, for ease of presentation, we restrict our attention to knowledge bases whose TBoxes are of restricted syntactical form presented below.

Definition 1. A TBox $\mathcal{T}$ is in *normal form* if every axiom in $\mathcal{T}$ is in one of the following forms:

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

(N2) $B_1 \sqsubseteq = n\, p.B_2$, **(N3)** $B_1 \sqsubseteq \forall p.B_2$, **(N4)** $r \sqsubseteq s$,

where $\{B_1, \ldots, B_m\} \subseteq \mathsf{N}_{\mathsf{C}}^+$, $n \geq 0$, $k > 1$, $m \geq k$, $p \in \mathsf{N}_{\mathsf{R}}$, and $\{r, s\} \subseteq \mathsf{N}_{\mathsf{R}}^+$.

We refer to the axioms of type **(N2)** as *counting axioms* and the roles that occur in such axioms as *counting roles*. Furthermore, we assume that TBoxes are closed under role inclusions.

Definition 2. A TBox $\mathcal{T}$ is *closed under role inclusions* if the following conditions hold:

(i) $p \sqsubseteq p \in \mathcal{T}$, for each $p \in \mathsf{N}_{\mathsf{R}}(\mathcal{T})$,
(ii) $r \sqsubseteq s \in \mathcal{T}$ implies $r^- \sqsubseteq s^- \in \mathcal{T}$, and
(iii) $r_1 \sqsubseteq r_2 \in \mathcal{T}$ and $r_2 \sqsubseteq r_3 \in \mathcal{T}$ implies $r_1 \sqsubseteq r_3 \in \mathcal{T}$.

Proposition 1. *Let $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$ be knowledge base 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 closed under role inclusions, and the following holds, for every interpretation $\mathcal{I}$:*

1. *if $\mathcal{I} \models (\mathcal{T}', \Sigma, \mathcal{A})$, then $\mathcal{I} \models (\mathcal{T}, \Sigma, \mathcal{A})$, and*

Table 2
Summary of symbols with fixed meaning, for a TBox $\mathcal{T}$.

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

2. *if $\mathcal{I} \models (\mathcal{T}, \Sigma, \mathcal{A})$, then $\mathcal{I}$ can be extended into $\mathcal{I}'$ such that $\mathcal{I}' \models (\mathcal{T}', \Sigma, \mathcal{A})$ and $q^{\mathcal{I}'} = q^{\mathcal{I}}$, for all concept and role names q occurring in $\mathcal{K}$.*

Proof (Sketch). The proof hinges on substituting fresh concept/role names for complex expressions in the original TBox. The first step involves removing all occurrences of $\leq$ and $\geq$ from $\mathcal{T}$. To this end, whenever we encounter an expression of the form $?n\,r.C$, where $? \in \{\leq, \geq\}$, r is a role, and C is a (possibly complex) concept, in some axiom of $\mathcal{T}$, we do the following. If $? = \leq$, we replace $\leq n\,r.C$ by $(= 0\,r.C \sqcup = 1\,r.C \sqcup \cdots \sqcup = n\,r.C)$. Furthermore, if $? = \geq$, we replace $\geq n\,r.C$ by $= n\,p_r.C$, where p_r is a fresh role name and we add $p_r \sqsubseteq r$ to $\mathcal{T}$. We do this recursively until there are no occurrences of $\leq$ and $\geq$ left in $\mathcal{T}$. It is easy to see that the set of models of the obtained TBox, restricted to the original signature, coincides with the set of models of $\mathcal{T}$. After this initial step, bringing the TBox into normal form is a standard procedure and its proof is therefore omitted (see, e.g., [33] for a normalization of $\mathcal{ALCHOI}$ TBoxes, expressions of the form $= n\,r.C$ are treated analogous to existential restrictions).

Finally, we assume that all concept and role names that occur in a knowledge base also occur in its TBox. We next fix some notation that remains the same for the remainder of the paper, summarized in Table 2. For a given TBox $\mathcal{T}$, we let $n_{\mathcal{T}}$ be the number of concept names in $\mathsf{N}_{\mathsf{C}}(\mathcal{T})$, $k_{\mathcal{T}}$ be the number of roles in $\mathsf{N}_{\mathsf{R}}^{+}(\mathcal{T})$, $m_{\mathcal{T}}$ be the number of counting axioms occurring in $\mathcal{T}$, and we let $c_{\mathcal{T}}$ be the maximum integer occurring in $\mathcal{T}$.

3. Characterizing KB satisfiability via integer programming

In this section, we 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 [35,36,38] that reduce the *finite satisfiability problem* to finding integer solutions to a system of linear inequalities. We reuse parts of the inequality systems in [37], 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 this paper, we modify the procedure to support ABoxes, closed predicates and generalized counting axioms instead of functionality. In the remainder of this section, 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. This characterization is the basis of our translation from queries mediated by $\mathcal{ALCHOIQ}$ TBoxes into Datalog with negation under the stable model semantics.

3.1. Chromatic models

Inspired by a technique used in [38], 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 paper.

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

Definition 3. Given a knowledge base $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$, a *role type* for $\mathcal{K}$ is any set $R \subseteq \mathsf{N}_{\mathsf{R}}^{+}(\mathcal{K})$ and a *type* for $\mathcal{K}$ is any set $T \subseteq \mathsf{N}_{\mathsf{C}}^{+}(\mathcal{K})$ such that

1. $\top \in T$ and $\bot \notin T$;
2. for all $a, b \in \mathsf{N}_{\mathsf{I}}(\mathcal{K})$, 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 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 4. Let $\mathcal{T}$ be an $\mathcal{ALCHOIQ}$ TBox, T, T' be types for $\mathcal{T}$, $R \subseteq \mathsf{N}_{\mathsf{R}}^{+}(\mathcal{T})$. 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 5. 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 2. *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 [38]) 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 4 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 there 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. Satisfiability via tiles & mosaics

We next introduce the key notions 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 satisfiability of $\mathcal{K}$ thus amounts to deciding the existence of a mosaic for $\mathcal{K}$. We next formally define these notions.

Definition 6. 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. $R \subseteq \mathsf{N}_{\mathsf{R}}^{+}(\mathcal{K})$, 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 7. 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.

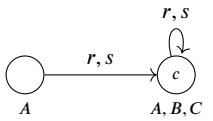
Example 1. Consider the following KB $\mathcal{K} = (\mathcal{T}, \{B\}, \{B(c)\})$, where $\mathcal{T} = \{B \sqcap C \sqsubseteq A, A \sqsubseteq\!= 1r.B, A \sqsubseteq \forall r.C, s \sqsubseteq r\}$. Let τ be the following tile for $\mathcal{K}$

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

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 6. 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 kth R-neighbor of d of type T'. This is reflected in the first condition. We also already mentioned that the tiles only encode relevant 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 T5a-T5c 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 T5a-T5c ensure that d and its neighbors respect axioms of the type $A \sqsubseteq \forall r.B$ and $r \sqsubseteq s$. Condition T5d makes use of Proposition 2 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 T5e, T5f, 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 on the following example.

Example 2. 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
- $0 \cdot \aleph_0 = \aleph_0 \cdot 0 = 0$.

Definition 8. Let $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$ be a knowledge base. A *mosaic* for $\mathcal{K}$ is a function $N : \text{Tiles}(\mathcal{K}) \to \mathbb{N}^*$ such that:

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

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

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

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

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

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

M5. For all $\{a\}, \{b\} \in \mathsf{N}_\mathsf{C}^+(\mathcal{K})$ and all $A, B \in \mathsf{N}_\mathsf{C}(\mathcal{K})$, if for some $p, r \in \mathsf{N}_\mathsf{R}^+(\mathcal{K})$ 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:

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

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

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

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. 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 $N : \text{Tiles}(\mathcal{K}) \to \mathbb{N}^*$ 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 8 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, defined in Section 3.1, 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 2. The condition T5d in Definition 6 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 posses 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 5 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 6 that requires that tiles explicitly record information about all the neighbors that are used for satisfying counting axioms.

The following result establishes the connection between the existence of mosaics and the satisfiability of $\mathcal{ALCHOIQ}$ KBs with closed predicates.

Theorem 1. *Let $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$ be an $\mathcal{ALCHOIQ}$ KB. $\mathcal{K}$ is satisfiable if and only if there exists a mosaic for $\mathcal{K}$ and 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 $r \in \Sigma \cap \mathsf{N}_{\mathsf{R}}$, $s \sqsubseteq r \in \mathcal{T}$ and $s(a, b) \in \mathcal{A}$, then $r(a, b) \in \mathcal{A}$, and*
4. *if $r \in \Sigma \cap \mathsf{N}_{\mathsf{R}}$, $s^- \sqsubseteq r \in \mathcal{T}$ and $s(a, b) \in \mathcal{A}$, then $r(b, a) \in \mathcal{A}$.*

The conditions 1-4 in Theorem 1 state that if we were to close the ABox under role inclusions in the TBox, we would 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.

3.3. Correctness of Theorem 1

Consider an arbitrary knowledge base $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$. We dedicate this subsection to showing that the result in Theorem 1 indeed holds. To this end, we first show that if $\mathcal{K}$ satisfies conditions 1-4 in Theorem 1 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 $\mathcal{K}$ satisfies the required conditions and we can easily construct a mosaic for $\mathcal{K}$.

From mosaic to model Assume that, for all $r \in \Sigma \cap \mathsf{N}_\mathsf{R}$, $\mathcal{K}$ satisfies conditions 1-4 in Theorem 1. Furthermore, 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 8, for each constant $a \in \mathsf{N}_\mathsf{I}(\mathcal{K})$, 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 $B \in \mathsf{N}_\mathsf{C}(\mathcal{K})$, 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 $R \subseteq \mathsf{N}_\mathsf{R}^+(\mathcal{T})$, if (T, R, T') is invertible, we do the following. Due to M3, we know that the following equation is satisfied:

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

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 $f_{T',R^-,T} = f_{T,R,T'}^{-1}$.
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 $\rho_{T'} = \{(R,T',k) : (R,T',k) \in \rho\}$, $\Delta_{T'}^\mathcal{I} = \{d \in \Delta^\mathcal{I} : d = (T',\rho')_j\}$, and let $\Delta_{e,T'}^\mathcal{I}$ 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 $a, b \in \mathsf{N}_{\mathsf{I}}(\mathcal{K})$, respectively. Thus, we have $\{a\} \in T$ and $\{b\} \in T'$. Moreover, we have that, for some $p \in \mathsf{N}_{\mathsf{R}}(\mathcal{K})$, 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 8, 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 T5d in Definition 6, 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 1. 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 $a, b \in \mathsf{N}_{\mathsf{I}}(\mathcal{K})$, 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 $a, b \in \mathsf{N}_{\mathsf{I}}(\mathcal{K})$, 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 T5a and T5b 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 1, this means that $e = a$ and $d = b$, for some constants $a, b \in \mathsf{N}_{\mathsf{I}}(\mathcal{K})$ 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 1 and the condition T5c 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 6, 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

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

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:

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

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 6. 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 6, $A \notin T_c$ and so $c \notin A^{\mathcal{I}}$. Hence, $\mathcal{I}$ satisfies $\neg A(c)$.

Let $p(a,b) \in \mathcal{A}$, where $p \in \mathsf{N}_{\mathsf{R}}$. 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 $p \in \mathsf{N}_{\mathsf{R}}$ and assume towards a contradiction that $(a,b) \in p^{\mathcal{I}}$. Due to our assumptions on $\mathcal{K}$, $(a,b) \in p^{\mathcal{I}}$ could not have been constructed due to conditions (i) or (ii) in Observation 1. Further, due to condition T5e in Definition 6, there is no $(R, T_b, k) \in \rho_a$ such that $p \in R$ and, due to condition T5f, there is no $(R, T_a, k) \in \rho_b$ such that $p^- \in R$. This is a contradiction to Observation 1 and $(a,b) \in p^{\mathcal{I}}$ could not have been constructed in the first place.

Satisfaction of the closed predicates. Let $A \in \mathsf{N}_{\mathsf{C}} \cap \Sigma$ 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 $c \in \mathsf{N}_{\mathsf{I}}$ 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 $r \in \mathsf{N}_{\mathsf{R}} \cap \Sigma$ 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 $a, b \in \mathsf{N}_{\mathsf{I}}(\mathcal{K})$ 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 for all $r \in \Sigma \cap \mathsf{N}_{\mathsf{R}}$, 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 1, 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 6, we have that there exists $c \in \mathsf{N}_{\mathsf{I}}(\mathcal{K})$ 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 $d \in \mathsf{N}_{\mathsf{I}}(\mathcal{K})$ 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, $\mathcal{K}$ satisfies the conditions 1-4 in Theorem 1. 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 $e^{\alpha}_1, \ldots, e^{\alpha}_n \in \Delta^{\mathcal{I}}$ such that $(e, e^{\alpha}_i) \in r^{\mathcal{I}}$ and $e^{\alpha}_i \in B^{\mathcal{I}}$, for $i = 1, \ldots, n$. We let

$$E(e) = \{e^{\alpha}_i \in \Delta^{\mathcal{I}} : \alpha = A \sqsubseteq\!= nr.B \in \mathcal{T}, 1 \leq i \leq n, A \in \mathsf{t}(e)\}$$

$$\rho_e = \{(R, T', i) : T' \in \text{Types}(\mathcal{T}), R \subseteq \mathsf{N}^{+}_{\mathsf{R}}(\mathcal{T}), 1 \leq i \leq |\{e' \in E(e) : \mathsf{rt}(e, e') = R, t(e') = T'\}|\}.$$

It is easy to verify that τ_e is indeed a tile for $\mathcal{K}$. We note that condition T5d is satisfied due to chromaticity of $\mathcal{I}$.

We next define a function $N : \text{Tiles}(\mathcal{T}) \to \mathbb{N}^*$ 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 8.

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 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 2. For each $d \in \Delta^{\mathcal{I}}$ with $\mathsf{t}(d) = T$, 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 $\mathsf{t}(d') = T'$ and $\mathsf{rt}(d, d') = R$.

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

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

Observation 4. For each $d' \in \Delta^{\mathcal{I}}$ with $\mathsf{t}(d') = T'$, 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 $\mathsf{t}(d) = T$ and $\mathsf{rt}(d', d) = R^-$.

Observation 5. For each $d' \in \Delta^{\mathcal{I}}$ with $\mathsf{t}(d') = T'$, $|\{d \in \Delta^{\mathcal{I}} : \mathsf{t}(d) = T, \mathsf{rt}(d', d) = R^-\}| \leq 1$.

We are now ready to show that M3 holds:

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

$$|\{d \in \Delta^{\mathcal{I}} : T_d = T \text{ and there exists some } k \text{ s.t. } (R, T', k) \in \rho_d\}| \overset{Obs.\ 2}{=}$$

$$|\{d \in \Delta^{\mathcal{I}} : T_d = T \text{ and there exists } d' \in \Delta^{\mathcal{I}} \text{ s.t. } \mathsf{rt}(d, d') = R, T_{d'} = T'\}| \overset{Obs.\ 3}{=}$$

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

$$|\{(d', d) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} : T_d = T, T_{d'} = T', \mathsf{rt}(d', d) = R^-\}| \overset{Obs.\ 5}{=}$$

$$|\{d' \in \Delta^{\mathcal{I}} : T_{d'} = T' \text{ and there exists } d \in \Delta^{\mathcal{I}} \text{ s.t. } \mathsf{rt}(d', d) = R^-, T_d = T\}| \overset{Obs.\ 4}{=}$$

$$|\{d' \in \Delta^{\mathcal{I}} : T_{d'} = T' \text{ and there exists some } l \text{ s.t. } (R^-, T, l) \in \rho_{d'}\}| =$$

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

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:

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

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 $N(\tau) \leq 0$.

3.4. Existence of mosaics via integer programming

So far, we have shown that we can reduce knowledge base satisfiability to deciding 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 9. An *enriched system (of 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

$$a_1 \cdot x_1 + \cdots + a_n \cdot x_n + c \leq b_1 \cdot y_1 + \cdots + b_m \cdot y_m + d, \tag{1}$$

  where $a_1, \ldots, a_n, b_1, \ldots, b_m, c, d$ are non-negative integers, and $x_1, \ldots, x_n, y_1, \ldots, y_m \in V$, and
- I is the set of implications of the form

$$\alpha \Rightarrow \beta, \tag{2}$$

  where α and β are inequalities of the form (1).

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

Definition 10. A *solution* S over $\mathbb{N}^*$ to an enriched system $(V, \mathcal{E}, I)$ is a function $S : V \to \mathbb{N}^*$ such that all inequalities and implications are satisfied.

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 $\mathcal{K}$. To this end, we first associate a variable x_τ to every tile $\tau \in \mathrm{Tiles}(\mathcal{K})$ and we let $V = \{x_\tau : \tau \in \mathrm{Tiles}(\mathcal{K})\}$. We obtain the set of inequalities $\mathcal{E}$ from the conditions M1-M3 and the set of implications I from the conditions M4-M5 in Definition 8 by simply replacing every occurrence of $N(\tau)$ by x_τ, for every $\tau \in \mathrm{Tiles}(\mathcal{K})$.

Proposition 3. *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.*

4. The translation

The goal of this section is to provide a translation, or a rewriting, of $\mathcal{ALCHOIQ}$ into Datalog with negation under the stable model semantics (Datalog$^\neg$) in the following sense. For a given $\mathcal{ALCHOIQ}$ TBox $\mathcal{T}$ and a set of closed predicates Σ, we obtain in polynomial time a Datalog$^\neg$ program $\mathcal{P}_{sat}^{\mathcal{T},\Sigma}$ that takes as input arbitrary ABoxes over the signature of $\mathcal{T}$ and has the following property: $\mathcal{P}_{sat}^{\mathcal{T},\Sigma}$ is satisfiable over an input ABox $\mathcal{A}$ (i.e., $\mathcal{P}_{sat}^{\mathcal{T},\Sigma}$ together with the facts extracted from $\mathcal{A}$ has a stable model) if and only if the knowledge base $(\mathcal{T}, \Sigma, \mathcal{A})$ is satisfiable.

4.1. Datalog$^\neg$ preliminaries

We next briefly introduce Datalog with negation under the stable model semantics [39]. We assume countably infinite and disjoint sets $\mathsf{N_V}$ and $\mathsf{N_P}$ of *variables* and *predicate symbols*, respectively, where each predicate symbol in $\mathsf{N_P}$ has an arity associated to it. We further assume that $\mathsf{N_C} \cup \mathsf{N_R} \subseteq \mathsf{N_P}$, where concept names from $\mathsf{N_C}$ are unary and role names from $\mathsf{N_R}$ are binary predicate symbols. An *atom* is an expression of the form $p(t_1, \ldots, t_n)$, where $p \in \mathsf{N_P}$ is an n-ary predicate symbol and $\{t_1, \ldots, t_n\} \subseteq \mathsf{N_V} \cup \mathsf{N_I}$. A *negated atom*

is an expression of the form *not* α, where α is an atom. We say that a (negated) atom $(not\ )p(t_1,\dots,t_n)$ is *ground* if $\{t_1,\dots,t_n\} \in \mathsf{N_I}$. A *rule* r is an expression of the form

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

where $n,m \geq 0$, h, $b_1,\dots,b_m$ are atoms, and all variables in r occur in some $b_1,\dots,b_n$. We let $head(r) = h$ be the *head of* r, $body^+(r) = \{b_1,\dots,b_n\}$ be the *positive body of* r, and $body^-(r) = \{b_{n+1},\dots,b_m\}$ be the *negative body of* r. If r contains no variables, r is called *ground*. If $body^-(r) = \emptyset$, r is *positive*. *Facts* are rules of the form $h \leftarrow$, abbreviated simply by the atom h. A *Datalog*$^\neg$ *program*, or simply a *program*, is a finite set of rules. A predicate p is called an *EDB (extensional database) predicate* if it only occurs in the body of the program's rules, and *IDB (intensional database) predicate* if it occurs in the head of some rule of the program. We say that a program is *ground* (resp. positive) if all its rules are ground (resp. positive). A *constraint* in a program $\mathcal{P}$ 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 $\mathcal{P}$.

A *Herbrand interpretation* I is any finite set of ground atoms. We assume the availability of a built-in predicate $=^i$ that compares two tuples of constants of length i in the obvious way, for each $i > 0$. In any Herbrand interpretation I, $(a_1,\dots,a_i) =^i (b_1,\dots,b_i) \in I$ iff (i) $1 \leq i \leq n$, where n is the maximum arity of any predicate symbol occurring in I, (ii) $a_1 = b_1 \dots a_i = b_i$, and (iii) all $a_1, b_1, \dots, a_i, b_i$ occur in some non-$=$ atoms in I. Note that these predicates can easily be axiomatized using (Datalog$^\neg$) rules. When i is clear from the context, we write $=$ instead of $=^i$. We also use $(a_1,\dots,a_i) \neq^i (b_1,\dots,b_i)$ instead of $not\ (a_1,\dots,a_i) = (b_1,\dots,b_i)$.

A *grounding of a program* $\mathcal{P}$, denoted by $\mathsf{ground}(\mathcal{P})$, is obtained from $\mathcal{P}$ in the following way. For each rule $r \in \mathcal{P}$, let $\mathsf{ground}(r,\mathcal{P})$ be the set of ground rules obtained from r by applying all possible uniform substitutions of constants occurring in $\mathcal{P}$ for the variables in r. We let $\mathsf{ground}(\mathcal{P}) = \bigcup_{r\in\mathcal{P}} \mathsf{ground}(r,\mathcal{P})$. Notice that $\mathsf{ground}(\mathcal{P})$ is a ground program. A Herbrand interpretation I is a *model* of a positive program $\mathcal{P}$, if for every rule $h \leftarrow b_1,\dots,b_n \in \mathsf{ground}(\mathcal{P})$, $b_1,\dots,b_n \in I$ implies $h \in I$. Further, I is a *minimal model of* $\mathcal{P}$ if there is no $J \subsetneq I$ such that J is a model of $\mathcal{P}$.

The *reduct* of a program $\mathcal{P}$ w.r.t. to a Herbrand interpretation I is a positive program $\mathcal{P}^I$ defined as

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

We say that I is a *stable model* (or an *answer set*) of $\mathcal{P}$ if I is a minimal model of $\mathcal{P}^I$.

The following generalization of Lemma 5.1 in [40] allows us to define programs in a modular way.

Proposition 4. *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two Datalog$^\neg$ 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 $\{I : I$ is an answer set of $P_2 \cup J$, for some answer set J of $P_1\}$.*

Finally, a *Datalog*$^\neg$ *query* is a pair $(\mathcal{P}, p)$, where $\mathcal{P}$ is a program and p is a distinguished predicate symbol occurring in $\mathcal{P}$. A tuple of constants $\mathbf{a}$ is a *certain answer* to the query $(\mathcal{P}, p)$ over a set of atoms I if $p(\mathbf{a}) \in J$, for each stable model J of $\mathcal{P} \cup I$.

4.2. $\mathcal{ALCHOIQ}$ satisfiability via Datalog$^\neg$

We now present the main idea behind our translation. Recall that, for a given TBox $\mathcal{T}$ and a set of predicates Σ occurring in $\mathcal{T}$, we want to construct a Datalog$^\neg$ program $\mathcal{P}^{\mathcal{T},\Sigma}_{sat}$ whose size depends solely on $\mathcal{T}$ and Σ, that can be used to determine whether $(\mathcal{T},\Sigma,\mathcal{A})$ is satisfiable, for any ABox $\mathcal{A}$ over the signature of $\mathcal{T}$. We note that we consider the ABox $\mathcal{A}$ an input to the program $\mathcal{P}^{\mathcal{T},\Sigma}_{sat}$ in the sense that the assertions in $\mathcal{A}$ are treated as additional facts that need to be taken into account when using $\mathcal{P}^{\mathcal{T},\Sigma}_{sat}$ to determine the satisfiability of $(\mathcal{T},\Sigma,\mathcal{A})$. Since the considered variant of Datalog supports only positive facts, we resort to a common technical trick to accommodate negative assertions in $\mathcal{A}$, i.e., assertions of the form $\neg q(\mathbf{a})$. 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 $\neg q(\mathbf{a})$ by $\overline{q}(\mathbf{a})$. Note that if $\mathcal{A}$ contains no negative assertions, $\mathcal{A}$ and $\hat{\mathcal{A}}$ coincide. Therefore, our goal for this section is to we show how to obtain in polynomial time the program $\mathcal{P}^{\mathcal{T},\Sigma}_{sat}$ from $\mathcal{T}$ and Σ such that, for all ABoxes $\mathcal{A}$ over the signature of $\mathcal{T}$, $\mathcal{P}^{\mathcal{T},\Sigma}_{sat} \cup \hat{\mathcal{A}}$ has a stable model if and only if $(\mathcal{T},\Sigma,\mathcal{A})$ is satisfiable.

Our translation is based on the characterization of the satisfiability problem for $\mathcal{ALCHOIQ}$ knowledge bases with closed predicates via existence of mosaics, as described in the previous section. Relying on Theorem 1, we construct the program $\mathcal{P}^{\mathcal{T},\Sigma}_{sat}$ consisting of two components, $\mathcal{P}^{\mathcal{T},\Sigma}_{sys}$ and $\mathcal{P}^{\mathcal{T},\Sigma}_{sol}$. 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 (2), and every inequality of the form (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

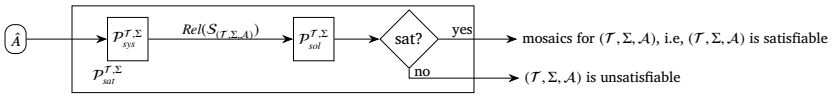


Fig. 1. $\mathcal{P}^{\Sigma,\mathcal{T}}_{sat}$ and its components.

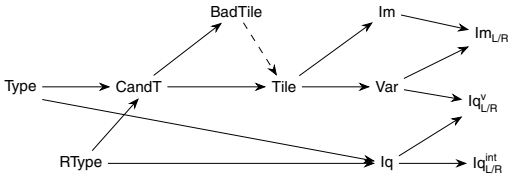


Fig. 2. Partial dependency graph of $\mathcal{P}^{\mathcal{T},\Sigma}_{sat}$ (negation represented via dashed arcs).

- Relation Iq^* storing IDs of inequalities in the enriched system that must be satisfied
- Relations $\mathsf{Iq}^{\mathsf{v}}_{\mathsf{L}}$ and $\mathsf{Iq}^{\mathsf{v}}_{\mathsf{R}}$ storing a pair $(\mathbf{q},\mathbf{v})$, for each inequality ID $\mathbf{q}$ and a variable ID $\mathbf{v}$ for which the variable identified by $\mathbf{v}$ occurs on the left-hand side (LHS) (resp. right-hand side (RHS)) of the inequality identified by $\mathbf{q}$.
- Relations $\mathsf{Iq}^{\mathsf{int}}_{\mathsf{L}}$ and $\mathsf{Iq}^{\mathsf{int}}_{\mathsf{R}}$ storing a pair $(\mathbf{q},\mathbf{n})$, for each inequality ID $\mathbf{q}$ and an integer n (in binary encoding) for which n occurs on the LHS (resp. RHS) of the inequality identified by $\mathbf{q}$.
- Relations Im_{L} and Im_{R} storing a pair $(\mathbf{m},\mathbf{q})$, for each implication ID $\mathbf{m}$ and an inequality ID $\mathbf{q}$ for which the inequality identified by $\mathbf{q}$ occurs on the LHS (resp. RHS) of the implication identified by $\mathbf{m}$.

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 the paper. 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 $\mathcal{A}$, $\mathcal{P}^{\mathcal{T},\Sigma}_{sys} \cup \hat{\mathcal{A}}$ checks whether $(\mathcal{T},\Sigma,\mathcal{A})$ satisfies conditions 1-4 of Theorem 1, and it computes $Rel(\mathcal{S}_{(\mathcal{T},\Sigma,\mathcal{A})})$, for the enriched system $\mathcal{S}_{(\mathcal{T},\Sigma,\mathcal{A})}$ defined in Section 3.4. 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. 1. It is worth noting that our constructed 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 4 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 $\mathcal{P}^{\mathcal{T},\Sigma}_{sysl}$. 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.

4.2.1. Generating linear inequalities

Consider an $\mathcal{ALCHOIQ}$ TBox $\mathcal{T}$ and a set of predicates Σ occurring in $\mathcal{T}$. We next show how to obtain in polynomial time the program $\mathcal{P}^{\mathcal{T},\Sigma}_{sys}$ that has the following property: for every ABox $\mathcal{A}$ over the signature of $\mathcal{T}$, $\mathcal{P}^{\mathcal{T},\Sigma}_{sys} \cup \hat{\mathcal{A}}$ has a stable model if and only if $(\mathcal{T},\Sigma,\mathcal{A})$ fulfills conditions 1-4 in Theorem 1. More importantly, the stable models of this program 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 $\mathcal{P}^{\mathcal{T},\Sigma}_{sys}$. This program is modular and consists of four 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 6 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 $(\mathcal{T},\Sigma,\mathcal{A})$ respect conditions 1-4 in Theorem 1.

We note that all constants used by $\mathcal{P}^{\mathcal{T},\Sigma}_{sys}$ (and also $\mathcal{P}^{\mathcal{T},\Sigma}_{sol}$) 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 4, the answer sets of $\mathcal{P}^{\mathcal{T},\Sigma}_{sys}$ 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 $\mathcal{P}^{\mathcal{T},\Sigma}_{sys}$ are depicted in Fig. 2 and the complete overview of the signature is listed in Table 3. 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 $\mathbf{0}$ to refer to the vector consisting of only zeroes. We note that length of $\mathbf{0}$ might vary from rule to rule. We are now ready to begin with our construction.

Table 3
Overview of the signature used for in $\mathcal{P}_{sys}$.

Relation	Arity	Meaning
A, for each $A \in \mathsf{N_C}(\mathcal{T})$	1	concept name $A \in \mathsf{N_C}(\mathcal{T})$
$\overline{A}$, for each $A \in \mathsf{N_C}(\mathcal{T})$	1	$\neg A$, for a concept name $A \in \mathsf{N_C}(\mathcal{T})$
r, for each $r \in \mathsf{N_R^+}(\mathcal{T})$	2	role $r \in \mathsf{N_R^+}(\mathcal{T})$
$\overline{r}$, for each $r \in \mathsf{N_R^+}(\mathcal{T})$	2	$\neg r$, for a role $r \in \mathsf{N_R^+}(\mathcal{T})$
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 $i = 1, \ldots, l_{tile}$	$2i$	linear order over strings of length i over constants from Cst
Tile	l_{tile}	tiles for $\mathcal{K}$
CandT	l_{tile}	candidate tiles for $\mathcal{K}$, i.e., tuples consisting of a type for $\mathcal{K}$ and n triples for $\mathcal{K}$ in the correct order w.r.t. LEQ
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 Definition 6
In$_{B_i}$, for $i = 1, \ldots, n_{\mathcal{T}}$	$n_{\mathcal{T}} + 1$	types for $\mathcal{K}$ that contain concept name B_i
In$_{r_i}$, for $i = 1, \ldots, k_{\mathcal{T}}$	$k_{\mathcal{T}}$	role types for $\mathcal{K}$ that contain role r_i
Succ$_{\text{int}}$	$2l_{int}$	successor relation over integers w.r.t. LEQ
$\overline{\text{Succ}_{\text{int}}}$	$2l_{int}$	complement of Succ$_{\text{int}}$
OK^1_i, for $i = 1, \ldots, n$	l_{tile}	candidate tiles whose i-th triple (R, T', k) satisfies T1 in Definition 6
OK^2_i, for $i = 1, \ldots, n$	l_{tile}	candidate tiles whose i-th triple (R, T', k) satisfies T2 in Definition 6
Upto$_{i, A \sqsubseteq = nr.B \in \mathcal{T}}$, for $i = 1, \ldots, n$	$l_{tile} + l_{int}$	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^-)
$\overline{\text{RType}^-}$	$2k_{\mathcal{T}}$	complement of RType$^-$
Upto$_i$, for $i = 1, \ldots, n$	$l_{tile} + n_{\mathcal{T}} + 1 + l_{int}$	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 $i = 1, \ldots, n$ 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 $S_{\mathcal{K}}$
Iq	l_{id}	inequalities in $S_{\mathcal{K}}$ (stand-alone or within implications)
Iq*	l_{id}	stand-alone inequalities in $S_{\mathcal{K}}$
Im	l_{id}	implications in $S_{\mathcal{K}}$
Iq$^{\text{int}}_{\text{L}}$/Iq$^{\text{int}}_{\text{R}}$	$l_{id} + l_{int}$	pairs of (q, z) s.t. integer z occurs in inequality q on the LHS/RHS
Iq$^{\text{v}}_{\text{L}}$/Iq$^{\text{v}}_{\text{R}}$	$l_{id} + l_{tile}$	pairs of (q, v) s.t. variable v occurs in inequality q on the LHS/RHS
Im$_{\text{L}}$/Im$_{\text{R}}$	$2l_{id}$	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 $l_{\text{id}} = 2(n_{\mathcal{T}} + 2) + k_{\mathcal{T}} + \max(n, 1)(n_{\mathcal{T}} + 1 + k_{\mathcal{T}} + l_{int})$.

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 $S_{\mathcal{K}}$, 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:

$$\text{Bin}(0), \qquad \text{Bin}(1).$$

Building on Bin, the relation Int stores all relevant integers for computing the system $\mathcal{S}_{\mathcal{K}}$. From T1 and T2 in Definition 6, 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 8, 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 $\mathcal{P}_{sys}^{\mathcal{T},\Sigma}$ in the size of $\mathcal{T}$. To gain access to the constants that occur in the ABox A while keeping the size of $\mathcal{P}_{sys}^{\mathcal{T},\Sigma}$ 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 Definition 6. 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 2). 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 $r_1, \ldots, r_{k_{\mathcal{T}}}$ 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 $11100\ldots0$ 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:

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

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 $B_1, \ldots, B_{n_{\mathcal{T}}}$ of the concept names in $\mathsf{N_C}(\mathcal{T})$. 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 $11100\ldots0a$ represents the type $\{B_1, B_2, B_3, \{a\}\}$, whereas the string $11100\ldots0*$ represents the type $\{B_1, B_2, B_3\}$. These strings are stored in the relation Type and are computed using the rule:

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

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:

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

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 6 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 $\mathcal{P}^{\mathcal{T},\Sigma}_{sys}$:

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

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:

$$\mathsf{LEQ}(0, x) \leftarrow \mathsf{Cst}(x),$$

$$\mathsf{LEQ}(x, x) \leftarrow \mathsf{Cst}(x),$$

$$\mathsf{LEQ}(x, y) \leftarrow \mathsf{Cst}(x), \mathsf{Cst}(y), \mathit{not}\ \mathsf{LEQ}(y, x),$$

$$\mathsf{LEQ}(x, z) \leftarrow \mathsf{LEQ}(x, y), \mathsf{LEQ}(y, z).$$

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

(i) $(\mathbf{R}_i, \mathbf{T}_i, \mathbf{k}_i, \mathbf{R}_j, \mathbf{T}_j, \mathbf{k}_j)$ 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 $(\mathbf{R}_i, \mathbf{T}_i, \mathbf{k}_i) = (\mathbf{R}_j, \mathbf{T}_j, \mathbf{k}_j)$, then $\mathbf{R}_i = (0, \ldots, 0)$, $\mathbf{T}_i = (0, \ldots, 0, *)$ and $\mathbf{k} = (0, \ldots, 0, 1)$, 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 InvCandT that is computed as follows:

$$\begin{aligned}\mathsf{InvCandT}(\mathbf{T}, \mathbf{R}_1, \mathbf{T}_1, \mathbf{k}_1, \ldots, \mathbf{R}_n, \mathbf{T}_n, \mathbf{k}_n) \leftarrow\ & \mathsf{Type}(\mathbf{T}), \mathsf{Triple}(\mathbf{R}_1, \mathbf{T}_1, \mathbf{k}_1), \ldots, \mathsf{Triple}(\mathbf{R}_n, \mathbf{T}_n, \mathbf{k}_n),\\ & \mathit{not}\ \mathsf{LEQ}^{n_{\mathcal{T}}+1+k_{\mathcal{T}}+\log(\max(n,1))}(\mathbf{R}_i, \mathbf{T}_i, \mathbf{k}_i, \mathbf{R}_j, \mathbf{T}_j, \mathbf{k}_j),\end{aligned}$$

$$\begin{aligned}\mathsf{InvCandT}(\mathbf{T}, \mathbf{R}_1, \mathbf{T}_1, \mathbf{k}_1, \ldots, \mathbf{R}_n, \mathbf{T}_n, \mathbf{k}_n) \leftarrow\ & \mathsf{Type}(\mathbf{T}), \mathsf{Triple}(\mathbf{R}_1, \mathbf{T}_1, \mathbf{k}_1), \ldots, \mathsf{Triple}(\mathbf{R}_n, \mathbf{T}_n, \mathbf{k}_n),\\ & (\mathbf{R}_i, \mathbf{T}_i, \mathbf{k}_i) = (\mathbf{R}_j, \mathbf{T}_j, \mathbf{k}_j), \mathbf{R}_i \neq (0, \ldots, 0),\end{aligned}$$

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

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

for all $1 \leq i, j \leq n$. Note that the first of the rules above stores in InvCandT the potential encodings for a tile that violate 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:

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

$$\mathit{not}\ \mathsf{InvCandT}(\mathbf{T}, \mathbf{R}_1, \mathbf{T}_1, \mathbf{k}_1, \dots, \mathbf{R}_n, \mathbf{T}_n, \mathbf{k}_n).$$

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 BadTile storing bad candidate tiles, i.e., those that violate one of the conditions in Definition 6. From now on, unless stated otherwise, we assume that $\mathbf{T}, \mathbf{T}'$ and $\mathbf{T}_i$, are variable vectors of length $n_\tau + 1$, $\mathbf{R}$ and $\mathbf{R}_i$ are variable vectors of length $k_\mathcal{T}$, $\mathbf{k}$ and $\mathbf{k}_i$ are variable vectors of length $\log(\max(n, 1))$, and $\mathbf{t}, \mathbf{t}'$, and $\mathbf{t}_i$ 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 $B_1, \dots, B_{n_\mathcal{T}}$ concept names in $\mathsf{N_C}(\mathcal{T})$ and $r_1, \dots, r_{k_\mathcal{T}}$ 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:

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

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

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

We now go through the conditions in Definition 6 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 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:

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

The first rule stores all pairs $(\mathbf{x}, \mathbf{y})$ in $\overline{\mathsf{Succ}_{\mathsf{int}}}$ for which the integer encoded by $\mathbf{y}$ is *not* the successor of the integer encoded by $\mathbf{x}$, i.e., if there is some integer encoded by $\mathbf{z}$, different from the ones encoded by $\mathbf{x}$ and $\mathbf{y}$, which is greater than $\mathbf{x}$ and smaller than $\mathbf{y}$. The second rule then stores pairs $(\mathbf{x}, \mathbf{y})$ in $\mathsf{Succ}_{\mathsf{int}}$, if $\mathbf{y}$ encodes an integer that is greater than the one encoded by $\mathbf{x}$ 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^{T1}_i, for all $1 \leq i \leq n$, that stores candidate tiles (T, ρ) whose i-th triple in ρ satisfies T1 in Definition 6. 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:

$$\mathsf{OK}^{T1}_i(\mathbf{T}, \mathbf{R}_1, \mathbf{T}_1, \mathbf{k}_1, \dots, \mathbf{R}_n, \mathbf{T}_n, \mathbf{k}_n) \leftarrow \mathsf{CandT}(\mathbf{T}, \mathbf{R}_1, \mathbf{T}_1, \mathbf{k}_1, \dots, \mathbf{R}_n, \mathbf{T}_n, \mathbf{k}_n), \mathbf{k}_i = (0, \dots, 0, 1),$$
$$\mathsf{OK}^{T1}_i(\mathbf{T}, \mathbf{R}_1, \mathbf{T}_1, \mathbf{k}_1, \dots, \mathbf{R}_n, \mathbf{T}_n, \mathbf{k}_n) \leftarrow \mathsf{CandT}(\mathbf{T}, \mathbf{R}_1, \mathbf{T}_1, \mathbf{k}_1, \dots, \mathbf{R}_n, \mathbf{T}_n, \mathbf{k}_n),$$
$$\mathbf{R}_j = \mathbf{R}_i, \mathbf{T}_j = \mathbf{T}_i, \mathsf{Succ}_{\mathsf{int}}(\mathbf{k}_j, \mathbf{k}_i), \text{ for } 1 \leq j \leq n, j \neq i.$$

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 $1 \leq i \leq n$:

$$\mathsf{BadTile}(\mathbf{t}) \leftarrow \mathsf{CandT}(\mathbf{t}), \mathit{not}\ \mathsf{OK}^{T1}_i(\mathbf{t}).$$

T2. For all $1 \leq i \leq n$, we define an auxiliary relation OK^{T2}_i that collects all candidate tiles (T, ρ) whose i-th triple in ρ fulfills T2 in Definition 6. The relation OK^{T2}_i is then computed using the following rules:

$$\mathsf{OK}^{T2}_i(\mathbf{T}, \mathbf{R}_1, \mathbf{T}_1, \mathbf{k}_1, \dots, \mathbf{R}_n, \mathbf{T}_n, \mathbf{k}_n) \leftarrow \mathsf{CandT}(\mathbf{T}, \mathbf{R}_1, \mathbf{T}_1, \mathbf{k}_1, \dots, \mathbf{R}_n, \mathbf{T}_n, \mathbf{k}_n),$$
$$\mathbf{R}_i = (0, \dots, 0), \mathbf{T}_i = (0, \dots, 0, *),$$
$$\mathsf{OK}^{T2}_i(\mathbf{T}, \mathbf{R}_1, \mathbf{T}_1, \mathbf{k}_1, \dots, \mathbf{R}_n, \mathbf{T}_n, \mathbf{k}_n) \leftarrow \mathsf{CandT}(\mathbf{T}, \mathbf{R}_1, \mathbf{T}_1, \mathbf{k}_1, \dots, \mathbf{R}_n, \mathbf{T}_n, \mathbf{k}_n),$$
$$\mathsf{In}_{B_{i_1}}(\mathbf{T}), \mathsf{In}_{B_{i_2}}(\mathbf{T}_i), \mathsf{In}_{r_h}(\mathbf{R}_i),$$

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:

$$\mathsf{BadTile}(\mathbf{t}) \leftarrow \mathsf{CandT}(\mathbf{t}), \mathit{not}\ \mathsf{OK}_i^{T2}(\mathbf{t}), \text{ for } 1 \le i \le n.$$

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:

$$\mathsf{BadTile}(\mathbf{T}, \mathbf{r}) \leftarrow \mathsf{CandT}(\mathbf{T}, \mathbf{r}), \mathsf{In}_{B_{i_1}}(\mathbf{T}), \ldots, \mathsf{In}_{B_{i_{m-1}}}(\mathbf{T}), \mathit{not}\ \mathsf{In}_{B_{i_m}}(\mathbf{T}), \ldots, \mathit{not}\ \mathsf{In}_{B_{i_t}}(\mathbf{T}).$$

T4. Eliminating the candidate tiles that violate T4 is done in multiple 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 $\mathbf{t} \in \mathsf{CandT}$ represents the candidate tile (T, ρ) and let $\mathbf{k} \in \mathsf{Int}$ 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 $(\mathbf{t}, \mathbf{k})$ 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 $j = 1, \ldots, n$:

$$\mathsf{Upto}_{0,\alpha}(\mathbf{T}, \mathbf{r}, \mathbf{0}) \leftarrow \mathsf{CandT}(\mathbf{T}, \mathbf{r}), \mathsf{In}_{B_{i_1}}(\mathbf{T}), \mathsf{Int}(\mathbf{0}),$$

$$\mathsf{Upto}_{j,\alpha}(\mathbf{T}, \mathbf{r}, \mathbf{k}) \leftarrow \mathsf{Upto}_{j-1,\alpha}(\mathbf{T}, \mathbf{r}, \mathbf{k}'), \mathsf{Succ}_{\mathsf{int}}(\mathbf{k}', \mathbf{k}), \mathsf{In}_{B_{i_2}}(\mathbf{T}_j), \mathsf{In}_{r_h}(\mathbf{R}_j),$$

$$\mathsf{Upto}_{j,\alpha}(\mathbf{T}, \mathbf{r}, \mathbf{k}) \leftarrow \mathsf{Upto}_{j-1,\alpha}(\mathbf{T}, \mathbf{r}, \mathbf{k}), \mathit{not}\ \mathsf{In}_{B_{i_2}}(\mathbf{T}_j),$$

$$\mathsf{Upto}_{j,\alpha}(\mathbf{T}, \mathbf{r}, \mathbf{k}) \leftarrow \mathsf{Upto}_{j-1,\alpha}(\mathbf{T}, \mathbf{r}, \mathbf{k}), \mathit{not}\ \mathsf{In}_{r_h}(\mathbf{R}_j),$$

where $\mathbf{r} = \mathbf{R}_1, \mathbf{T}_1, \mathbf{k}_1, \ldots, \mathbf{R}_n, \mathbf{T}_n, \mathbf{k}_n$. 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

$$\mathsf{BadTile}(\mathbf{t}) \leftarrow \mathsf{CandT}(\mathbf{t}), \mathit{not}\ \mathsf{Upto}_{n,B_{i_1} \sqsubseteq= m\ r_h.B_{i_2}}(\mathbf{t}, \mathbf{b}),$$

where $\mathbf{b}$ 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:

$$\mathsf{BadTile}(\mathbf{T}, \mathbf{R}_1, \mathbf{T}_1, \mathbf{k}_1, \ldots, \mathbf{R}_n, \mathbf{T}_n, \mathbf{k}_n) \leftarrow \mathsf{CandT}(\mathbf{T}, \mathbf{R}_1, \mathbf{T}_1, \mathbf{k}_1, \ldots, \mathbf{R}_n, \mathbf{T}_n, \mathbf{k}_n), \mathsf{In}_{B_{i_1}}(\mathbf{T}), \mathsf{In}_{r_h}(\mathbf{R}_j), \mathit{not}\ \mathsf{In}_{B_{i_2}}(\mathbf{T}_j),$$

$$\mathsf{BadTile}(\mathbf{T}, \mathbf{R}_1, \mathbf{T}_1, \mathbf{k}_1, \ldots, \mathbf{R}_n, \mathbf{T}_n, \mathbf{k}_n) \leftarrow \mathsf{CandT}(\mathbf{T}, \mathbf{R}_1, \mathbf{T}_1, \mathbf{k}_1, \ldots, \mathbf{R}_n, \mathbf{T}_n, \mathbf{k}_n), \mathsf{In}_{B_{i_1}}(\mathbf{T}_j), \mathsf{In}_{r_{h'}}(\mathbf{R}_j), \mathit{not}\ \mathsf{In}_{B_{i_2}}(\mathbf{T}),$$

for all $1 \le j \le n$. This makes sure to eliminate the tiles that do not satisfy T5a and T5b.

To satisfy the T5c we do the following. For $1 \le j \le n$ and each $r_h \sqsubseteq r_g \in \mathcal{T}$, we add the following rule:

$$\mathsf{BadTile}(\mathbf{T}, \mathbf{R}_1, \mathbf{T}_1, \mathbf{k}_1, \ldots, \mathbf{R}_n, \mathbf{T}_n, \mathbf{k}_n) \leftarrow \mathsf{CandT}(\mathbf{T}, \mathbf{R}_1, \mathbf{T}_1, \mathbf{k}_1, \ldots, \mathbf{R}_n, \mathbf{T}_n, \mathbf{k}_n), \mathsf{In}_{r_h}(\mathbf{R}_j), \mathit{not}\ \mathsf{In}_{r_g}(\mathbf{R}_j).$$

The rules for T5a-T5c are rather simple – they are direct translations of the conditions in Definition 6.

To deal with T5d, we first need a way to represent invertible triples (see Definition 4). 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:

$$\mathsf{Invrt}(\mathbf{T}_1, \mathbf{R}, \mathbf{T}_2) \leftarrow \mathsf{Type}(\mathbf{T}_1), \mathsf{RType}(\mathbf{R}), \mathsf{Type}(\mathbf{T}_2), \mathsf{In}_{B_{i_1}}(\mathbf{T}_1), \mathsf{In}_{B_{j_2}}(\mathbf{T}_1),$$
$$\mathsf{In}_{B_{i_2}}(\mathbf{T}_2), \mathsf{In}_{B_{j_1}}(\mathbf{T}_2), \mathsf{In}_{r_h}(\mathbf{R}), \mathsf{In}_{r_{g'}}(\mathbf{R}),$$

where $r_{g'}$ denotes the role r_g^-.

Then, adding the rules that "invalidate" candidate tiles that violate T5d is once again straightforward:

$$\mathsf{BadTile}(\mathbf{T}, \mathbf{R}_1, \mathbf{T}_1, \mathbf{k}_1, \ldots, \mathbf{R}_n, \mathbf{T}_n, \mathbf{k}_n) \leftarrow \mathsf{CandT}(\mathbf{T}, \mathbf{R}_1, \mathbf{T}_1, \mathbf{k}_1, \ldots, \mathbf{R}_n, \mathbf{T}_n, \mathbf{k}_n),$$
$$\mathsf{Invrt}(\mathbf{T}, \mathbf{R}_i, \mathbf{T}_i), \mathbf{T} = \mathbf{T}_i,$$

$$\mathsf{BadTile}(\mathbf{T}, \mathbf{R}_1, \mathbf{T}_1, \mathbf{k}_1, \ldots, \mathbf{R}_n, \mathbf{T}_n, \mathbf{k}_n) \leftarrow \mathsf{CandT}(\mathbf{T}, \mathbf{R}_1, \mathbf{T}_1, \mathbf{k}_1, \ldots, \mathbf{R}_n, \mathbf{T}_n, \mathbf{k}_n),$$
$$\mathsf{Invrt}(\mathbf{T}, \mathbf{R}_i, \mathbf{T}_i), \mathsf{Invrt}(\mathbf{T}, \mathbf{R}_j, \mathbf{T}_j), \mathbf{T}_i = \mathbf{T}_j,$$

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:

$$\mathsf{BadTile}(\mathbf{T}, \mathbf{R}_1, \mathbf{T}_1, \mathbf{k}_1, \ldots, \mathbf{R}_n, \mathbf{T}_n, \mathbf{k}_n) \leftarrow \mathsf{CandT}(\mathbf{T}, \mathbf{R}_1, \mathbf{T}_1, \mathbf{k}_1, \ldots, \mathbf{R}_n, \mathbf{T}_n, \mathbf{k}_n),$$
$$\overline{r_h}(x, x'), \mathsf{In}_{r_h}(\mathbf{y}_i), w_i = x',$$

$$\mathsf{BadTile}(\mathbf{T}, \mathbf{R}_1, \mathbf{T}_1, \mathbf{k}_1, \ldots, \mathbf{R}_n, \mathbf{T}_n, \mathbf{k}_n) \leftarrow \mathsf{CandT}(\mathbf{T}, \mathbf{R}_1, \mathbf{T}_1, \mathbf{k}_1, \ldots, \mathbf{R}_n, \mathbf{T}_n, \mathbf{k}_n),$$
$$\overline{r_h}(x', x), \mathsf{In}_{r_{h'}}(\mathbf{R}_i), w_i = x',$$

where $r_{h'} = r_h^-$, $\mathbf{T} = x_1, \ldots, x_{n_\mathcal{T}}, w$, and $\mathbf{T}_i = x_{i_1}, \ldots, x_{i_{n_\mathcal{T}}}, w_i$.

The rules that invalidate the tiles that violate one of T6-T9 are once again straightforward. In what follows, we assume $\mathbf{T} = x_1, \ldots, x_{n_\mathcal{T}}, w$ and $\mathbf{T}_i = x_{i_1}, \ldots, x_{i_{n_\mathcal{T}}}, w_i$, for all $1 \leq i \leq n$.

T6. For each $B_i \in \mathsf{N_C}(\mathcal{T})$ we add:

$$\mathsf{BadTile}(\mathbf{T}, \mathbf{r}) \leftarrow \mathsf{CandT}(\mathbf{T}, \mathbf{r}), B_i(w), \mathit{not}\ \mathsf{In}_{B_i}(\mathbf{T}).$$

T7. For each $B_i \in \mathsf{N_C}(\mathcal{T})$ we add:

$$\mathsf{BadTile}(\mathbf{T}, \mathbf{r}) \leftarrow \mathsf{CandT}(\mathbf{T}, \mathbf{r}), \overline{B_i}(w), \mathsf{In}_{B_i}(\mathbf{T}).$$

T8. For each $B_i \in \Sigma \cap \mathsf{N_C}(\mathcal{T})$,

$$\mathsf{BadTile}(\mathbf{T}, \mathbf{r}) \leftarrow \mathsf{CandT}(\mathbf{T}, \mathbf{r}), \mathsf{In}_{B_i}(\mathbf{T}), \mathit{not}\ B_i(w).$$

T9. For each $r_h \in \Sigma \cap \mathsf{N_R}(\mathcal{T})$ and each $i = 1, \ldots, n$, we add:

$$\mathsf{BadTile}(\mathbf{T}, \mathbf{R}_1, \mathbf{T}_1, \mathbf{k}_1, \ldots, \mathbf{R}_n, \mathbf{T}_n, \mathbf{k}_n) \leftarrow \mathsf{CandT}(\mathbf{T}, \mathbf{R}_1, \mathbf{T}_1, \mathbf{k}_1, \ldots, \mathbf{R}_n, \mathbf{T}_n, \mathbf{k}_n),$$
$$\mathsf{In}_{r_h}(\mathbf{R}_i), \mathit{not}\ r_h(w, w_i).$$

At this point, the relation 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 $\mathcal{K}$:

$$\mathsf{Tile}(\mathbf{t}) \leftarrow \mathsf{CandT}(\mathbf{t}), \mathit{not}\ \mathsf{BadTile}(\mathbf{t}).$$

Recall that the answer sets of the first component enriched with the facts from the ABox $\mathcal{A}$ contain precisely one tuple $\mathbf{t}$ in CandT, for each candidate tile τ encoded by $\mathbf{t}$. 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 $\tau \in \mathsf{Tiles}(\mathcal{K})$.

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

$$\mathsf{Var}(\mathbf{t}) \leftarrow \mathsf{Tile}(\mathbf{t}).$$

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, that assigns to each implication of the form (2), and inequality of the form (1) occurring in $\mathcal{S}_\mathcal{K}$ a string of length l_{id} over the available constants that uniquely identifies it.

Table 4
Identifiers of inequalities and implications of $S_{\mathcal{K}}$.

Condition	Inequality/Implication	ID
M1	$\sum_{(T,\rho)\in \mathrm{Tiles}(\mathcal{K}),\{c\}\in T} x_{(T,\rho)} \leq 1$, for $\{c\} \in \mathsf{N}_{\mathsf{C}}^{+}(\mathcal{K})$	$(0, c, \mathbf{0})$
M1	$1 \leq \sum_{(T,\rho)\in \mathrm{Tiles}(\mathcal{K}),\{c\}\in T} x_{(T,\rho)}$, for $\{c\} \in \mathsf{N}_{\mathsf{C}}^{+}(\mathcal{K})$	$(1, c, \mathbf{0})$
M2	$1 \leq \sum_{\tau\in \mathrm{Tiles}(\mathcal{K})} x_{\tau}$	$(\mathbf{0})$
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 $T, T' \in \mathrm{Types}(\mathcal{K})$ and $R \subseteq \mathsf{N}_{\mathsf{R}}^{+}(\mathcal{K})$ s.t. (T, R, T') invertible	$(0, \mathbf{T}, \mathbf{T}', \mathbf{R}, \mathbf{0})$, where $\mathbf{T}$ and $\mathbf{T}'$ encode T and T', respectively, and $\mathbf{R}$ 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	$(1, \mathbf{T}, \mathbf{T}', \mathbf{R}, \mathbf{0})$, where $\mathbf{T}$ and $\mathbf{T}'$ encode T and T', respectively, and $\mathbf{R}$ encodes R
M4/M6	$1 \leq x_{\tau}$, for $\tau \in \mathrm{Tiles}(\mathcal{K})$	$(\mathbf{t}, \mathbf{0})$, where $\mathbf{t}$ 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 $T' \in \mathrm{Types}(\mathcal{K})$	$(\mathbf{t}, \mathbf{T}', \mathbf{0})$, where $\mathbf{t}$ encodes τ and $\mathbf{T}'$ 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 $T' \in \mathrm{Types}(\mathcal{K})$,	$(\mathbf{t}, \mathbf{T}', \mathbf{0})$, where $\mathbf{t}$ encodes τ and $\mathbf{T}'$ encodes T'
M5	$1 \leq \sum_{\substack{(T,\rho)\in \mathrm{Tiles}(\mathcal{K}),\\ \{c\},B\in T}} x_{(T,\rho)}$, for $\{c\} \in \mathsf{N}_{\mathsf{C}}^{+}(\mathcal{K})$, $B \in \mathsf{N}_{\mathsf{C}}(\mathcal{K})$	$(c, \mathbf{b}, \mathbf{0})$, where $\mathbf{b}$ 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 $B, B' \in \mathsf{N}_{\mathsf{C}}(\mathcal{K})$	$(c_1, c_2, \mathbf{b}, \mathbf{b}', \mathbf{0})$, where $\mathbf{b}$ (resp. $\mathbf{b}'$) 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 $r \in \mathsf{N}_{\mathsf{R}}^{+}(\mathcal{K})$	$(c, \mathbf{T}', \mathbf{b}, \mathbf{r}, \mathbf{0})$, where $\mathbf{T}'$ encodes T', $\mathbf{b}$ encodes the type containing only B, and $\mathbf{r}$ 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 $r \in \mathsf{N}_{\mathsf{R}}^{+}(\mathcal{K})$	$(c, \mathbf{t}', \mathbf{b}, \mathbf{r}, \mathbf{0})$, where $\mathbf{t}'$ encodes τ', $\mathbf{b}$ encodes the type containing only B, and $\mathbf{r}$ 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 $\mathbf{0}$ is always a all-zero vector of the appropriate length.

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:

$$\mathsf{Iq}(\mathbf{x}) \leftarrow \mathsf{Iq}^*(\mathbf{x}).$$

Once we agree on how IDs are assigned to the implications and inequalities, we go through the conditions in Definition 8 and add the rules that compute the relevant relations storing implications and inequalities.

M1. As Table 4 suggests, for each knowledge base constant c, M1 gives rise to two inequalities identified by $(0, c, \mathbf{0})$ and $(1, c, \mathbf{0})$. We add the rules that store these IDs in Iq^*:

$$\mathsf{Iq}^*(0, x, \mathbf{0}) \leftarrow \mathsf{Adom}(x), \qquad \mathsf{Iq}^*(1, x, \mathbf{0}) \leftarrow \mathsf{Adom}(x).$$

We next need to relate these inequalities with the variables and integers that occur in them using relations $\mathsf{Iq}_{\mathsf{L}}^{\mathsf{v}}$, $\mathsf{Iq}_{\mathsf{R}}^{\mathsf{v}}$, $\mathsf{Iq}_{\mathsf{L}}^{\mathsf{int}}$, and $\mathsf{Iq}_{\mathsf{R}}^{\mathsf{int}}$.
On the LHS of the inequality identified by $(0, c, \mathbf{0})$ as well as and the RHS of the inequality identified by $(1, c, \mathbf{0})$, 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:

$$\mathsf{Iq}_{\mathsf{L}}^{\mathsf{v}}(0, x, \mathbf{0}, \mathbf{T}, \mathbf{r}) \leftarrow \mathsf{Iq}(0, x, \mathbf{0}), \mathsf{Adom}(x), \mathsf{Var}(\mathbf{T}, \mathbf{r}), w = x,$$

$$\mathsf{Iq}_{\mathsf{R}}^{\mathsf{v}}(1, x, \mathbf{0}, \mathbf{T}, \mathbf{r}) \leftarrow \mathsf{Iq}(1, x, \mathbf{0}), \mathsf{Adom}(x), \mathsf{Var}(\mathbf{T}, \mathbf{r}), w = x,$$

where $\mathbf{T} = x_1, \ldots, x_{n_{\mathcal{T}}}, w$.

On the RHS of the inequality identified by $(0, c, \mathbf{0})$ as well as the LHS of the inequality identified by $(1, c, \mathbf{0})$ is equal to 1. We encode this using the following:

$$\mathsf{Iq}^{\mathsf{int}}_{\mathsf{R}}(0, x, \mathbf{0}, 1) \leftarrow \mathsf{Adom}(x),$$

$$\mathsf{Iq}^{\mathsf{int}}_{\mathsf{L}}(1, x, \mathbf{0}, 1) \leftarrow \mathsf{Adom}(x).$$

M2. The inequality in M2 is simply identified by $\mathbf{0}$ of length l_{id}. Thus, we add the fact $\mathsf{Iq}^*(\mathbf{0})$. On the RHS of this inequality we have all variables and on the LHS we have the constant 1. Thus, we add:

$$\mathsf{Iq}^{\mathsf{v}}_{\mathsf{R}}(\mathbf{0}, \mathbf{t}) \leftarrow \mathsf{Var}(\mathbf{t}) \qquad \text{and} \qquad \mathsf{Iq}^{\mathsf{int}}_{\mathsf{L}}(\mathbf{0}, 1).$$

M3. For all types $T, T' \in \mathsf{Types}(\mathcal{K})$ and a role type $R \subseteq \mathsf{N}^+_{\mathsf{R}}(\mathcal{K})$ such that (T, R, T') is invertible, M3 introduces two inequalities identified by $(0, \mathbf{T}, \mathbf{T}', \mathbf{R}, \mathbf{0})$ and $(1, \mathbf{T}, \mathbf{T}', \mathbf{R}, \mathbf{0})$, where $\mathbf{T}$ and $\mathbf{T}'$ identify T and T', respectively, and $\mathbf{R}$ identifies R. We thus add the following rules to store these IDs in Iq^* using the following rules:

$$\mathsf{Iq}^*(0, \mathbf{T}, \mathbf{T}', \mathbf{R}, \mathbf{0}) \leftarrow \mathsf{Type}(\mathbf{T}), \mathsf{Type}(\mathbf{T}'), \mathsf{RType}(\mathbf{R}), \mathsf{Invrt}(\mathbf{T}, \mathbf{R}, \mathbf{T}'),$$

$$\mathsf{Iq}^*(1, \mathbf{T}, \mathbf{T}', \mathbf{R}, \mathbf{0}) \leftarrow \mathsf{Type}(\mathbf{T}), \mathsf{Type}(\mathbf{T}'), \mathsf{RType}(\mathbf{R}), \mathsf{Invrt}(\mathbf{T}, \mathbf{R}, \mathbf{T}').$$

On the LHS of the inequality identified by $(0, \mathbf{T}, \mathbf{T}', \mathbf{R}, \mathbf{0})$ and the RHS of the inequality identified by $(1, \mathbf{T}, \mathbf{T}', \mathbf{R}, \mathbf{0})$, we have all tiles (T, ρ) with $(R, T', k) \in \rho$, for some k. Thus, we add:

$$\mathsf{Iq}^{\mathsf{v}}_{\mathsf{L}}(0, \mathbf{T}, \mathbf{T}', \mathbf{R}, \mathbf{0}, \mathbf{t}) \leftarrow \mathsf{Iq}^*(0, \mathbf{T}, \mathbf{T}', \mathbf{R}, \mathbf{0}), \mathsf{Var}(\mathbf{t}), \mathbf{R}_i = \mathbf{R}, \mathbf{T}_i = \mathbf{T}',$$

$$\mathsf{Iq}^{\mathsf{v}}_{\mathsf{R}}(1, \mathbf{T}, \mathbf{T}', \mathbf{R}, \mathbf{0}, \mathbf{t}) \leftarrow \mathsf{Iq}^*(1, \mathbf{T}, \mathbf{T}', \mathbf{R}, \mathbf{0}), \mathsf{Var}(\mathbf{t}), \mathbf{R}_i = \mathbf{R}, \mathbf{T}_i = \mathbf{T}',$$

for all $1 \leq i \leq n$, where $\mathbf{t} = \mathbf{T}, \mathbf{R}_1, \mathbf{T}_1, \mathbf{k}_1, \ldots, \mathbf{R}_n, \mathbf{T}_n, \mathbf{k}_n$.
Further, on the RHS of the inequality identified by $(0, \mathbf{T}, \mathbf{T}', \mathbf{R}, \mathbf{0})$ and the LHS of the inequality identified by $(1, \mathbf{T}, \mathbf{T}', \mathbf{R}, \mathbf{0})$, we have all tiles (T', ρ') with $(R^-, T, k) \in \rho'$, for some k. We therefore add:

$$\mathsf{Iq}^{\mathsf{v}}_{\mathsf{R}}(0, \mathbf{T}, \mathbf{T}', \mathbf{R}, \mathbf{0}, \mathbf{t}) \leftarrow \mathsf{Iq}^*(0, \mathbf{T}, \mathbf{T}', \mathbf{R}, \mathbf{0}), \mathsf{Var}(\mathbf{t}), \mathsf{RType}^-(\mathbf{R}, \mathbf{R}'), \mathbf{R}_i = \mathbf{R}', \mathbf{T}_i = \mathbf{T},$$

$$\mathsf{Iq}^{\mathsf{v}}_{\mathsf{L}}(1, \mathbf{T}, \mathbf{T}', \mathbf{R}, \mathbf{0}, \mathbf{t}) \leftarrow \mathsf{Iq}^*(1, \mathbf{T}, \mathbf{T}', \mathbf{R}, \mathbf{0}), \mathsf{Var}(\mathbf{t}), \mathsf{RType}^-(\mathbf{R}, \mathbf{R}'), \mathbf{R}_i = \mathbf{R}', \mathbf{T}_i = \mathbf{T},$$

for all $1 \leq i \leq n$, where $\mathbf{t} = \mathbf{T}', \mathbf{R}_1, \mathbf{T}_1, \mathbf{k}_1, \ldots, \mathbf{R}_n, \mathbf{T}_n, \mathbf{k}_n$.
The rules above make use of an auxiliary relation RType^-, that stores $(\mathbf{R}, \mathbf{R}')$ if $\mathbf{R}$ encodes a role type R and $\mathbf{R}'$ 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:

$$\overline{\mathsf{RType}^-}(\mathbf{R}, \mathbf{R}') \leftarrow \mathsf{RType}(\mathbf{R}), \mathsf{RType}(\mathbf{R}'), \mathsf{In}_{r_g}(\mathbf{R}), \mathit{not}\ \mathsf{In}_{r_{g'}}(\mathbf{R}'),$$

$$\overline{\mathsf{RType}^-}(\mathbf{R}, \mathbf{R}') \leftarrow \mathsf{RType}(\mathbf{R}), \mathsf{RType}(\mathbf{R}'), \mathsf{In}_{r_g}(\mathbf{R}'), \mathit{not}\ \mathsf{In}_{r_{g'}}(\mathbf{R}),$$

$$\mathsf{RType}^-(\mathbf{R}, \mathbf{R}') \leftarrow \mathsf{RType}(\mathbf{R}), \mathsf{RType}(\mathbf{R}'), \mathit{not}\ \overline{\mathsf{RType}^-}(\mathbf{R}, \mathbf{R}').$$

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 $(\mathbf{R}, \mathbf{R}') \in \mathsf{RType}^-$.

M4. For each tile $\tau = (T, \rho) \in \mathsf{Tiles}(\mathcal{K})$ and each type $T' \in \mathsf{Types}(\mathcal{K})$, the condition M4 introduces an implication with the ID $(\mathbf{t}, \mathbf{T}', \mathbf{0})$, which in turn consists of two inequalities identified by $(\mathbf{t}, \mathbf{0})$ and $(\mathbf{t}, \mathbf{T}', \mathbf{0})$, where $\mathbf{t}$ encodes τ and $\mathbf{T}'$ encodes T'. We define rules that store these IDs in Im and Iq:

$$\mathsf{Im}(\mathbf{t}, \mathbf{T}', \mathbf{0}) \leftarrow \mathsf{Tile}(\mathbf{t}), \mathsf{Type}(\mathbf{T}'),$$

$$\mathsf{Iq}(\mathbf{t}, \mathbf{0}) \leftarrow \mathsf{Tile}(\mathbf{t}),$$

$$\mathsf{Iq}(\mathbf{t}, \mathbf{T}', \mathbf{0}) \leftarrow \mathsf{Tile}(\mathbf{t}), \mathsf{Type}(\mathbf{T}').$$

We also add the rules that relate the implications with the inequalities they consist of:

$$\mathsf{Im}_{\mathsf{L}}(\mathbf{t}, \mathbf{T}', \mathbf{0}_1, \mathbf{t}, \mathbf{0}_2) \leftarrow \mathsf{Im}(\mathbf{t}, \mathbf{T}', \mathbf{0}_1), \mathsf{Iq}(\mathbf{t}, \mathbf{0}_2), \mathsf{Tile}(\mathbf{t}), \mathsf{Type}(\mathbf{T}'),$$

$$\mathsf{Im}_{\mathsf{R}}(\mathbf{t}, \mathbf{T}', \mathbf{0}_1, \mathbf{t}, \mathbf{T}', \mathbf{0}_2) \leftarrow \mathsf{Im}(\mathbf{t}, \mathbf{T}', \mathbf{0}_1), \mathsf{Iq}(\mathbf{t}, \mathbf{T}', \mathbf{0}_2), \mathsf{Tile}(\mathbf{t}), \mathsf{Type}(\mathbf{T}').$$

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 $(\mathbf{t}, \mathbf{0})$ is equal to 1 and the RHS simply consists of the tile encoded by $\mathbf{t}$. Thus, we have:

$$\mathsf{Iq}^{\mathsf{v}}_{\mathsf{R}}(\mathbf{t}, \mathbf{0}, \mathbf{t}) \leftarrow \mathsf{Iq}(\mathbf{t}, \mathbf{0}), \mathsf{Tile}(\mathbf{t}),$$

$$\mathsf{Iq}^{\mathsf{int}}_{\mathsf{L}}(\mathbf{t}, \mathbf{0}, 1) \leftarrow \mathsf{Iq}(\mathbf{t}, \mathbf{0}), \mathsf{Tile}(\mathbf{t}).$$

On the LHS of the inequality with the ID $(\mathbf{t}, \mathbf{T}', \mathbf{0})$ we have all tiles $\tau' = (T', \rho')$ such that T' is encoded by $\mathbf{T}'$, while on the RHS we have the number of triples in ρ that contains the type encoded by $\mathbf{T}'$. We add the following rules:

$$\mathsf{Iq}^{\mathsf{int}}_{\mathsf{L}}(\mathbf{t}, \mathbf{T}', \mathbf{0}, \mathbf{k}) \leftarrow \mathsf{Iq}(\mathbf{t}, \mathbf{T}', \mathbf{0}), \mathsf{Tile}(\mathbf{t}), \mathsf{Type}(\mathbf{T}'), \mathsf{Upto}_n(\mathbf{t}, \mathbf{T}', \mathbf{k}),$$

$$\mathsf{Iq}^{\mathsf{v}}_{\mathsf{R}}(\mathbf{t}, \mathbf{T}', \mathbf{0}, \mathbf{t}') \leftarrow \mathsf{Iq}(\mathbf{t}, \mathbf{T}', \mathbf{0}), \mathsf{Tile}(\mathbf{t}), \mathsf{Type}(\mathbf{T}'), \mathsf{Tile}(\mathbf{t}'),$$

where $\mathbf{t} = \mathbf{T}, \mathbf{R}_1, \mathbf{T}_1, \mathbf{k}_1, \dots, \mathbf{R}_n, \mathbf{T}_n, \mathbf{k}_n$ and $\mathbf{t}' = \mathbf{T}', \mathbf{R}'_1, \mathbf{T}'_1, \mathbf{k}'_1, \dots, \mathbf{R}'_n, \mathbf{T}'_n, \mathbf{k}'_n$.
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 $\mathbf{t}$, T' be a type identified by $\mathbf{T}'$ and k be an integer whose binary representation $\mathbf{k}$ is stored in Int. The relation Upto_i stores the tuple $(\mathbf{t}, \mathbf{Tt}', \mathbf{k})$ 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:

$$\mathsf{Upto}_0(\mathbf{t}, \mathbf{T}', \mathbf{0}) \leftarrow \mathsf{Tile}(\mathbf{t}), \mathsf{Type}(\mathbf{T}'),$$

$$\mathsf{Upto}_i(\mathbf{t}, \mathbf{T}', \mathbf{k}) \leftarrow \mathsf{Upto}_{i-1}(\mathbf{t}, \mathbf{T}', \mathbf{k}'), \mathsf{Succ}_{\mathsf{int}}(\mathbf{k}', \mathbf{k}), \mathbf{T}_i = \mathbf{T}',$$

$$\mathsf{Upto}_i(\mathbf{t}, \mathbf{T}', \mathbf{k}) \leftarrow \mathsf{Upto}_{i-1}(\mathbf{t}, \mathbf{T}', \mathbf{k}), \mathit{not}\ \mathbf{T}_i \neq \mathbf{T}',$$

for each $1 \leq i \leq n$, where $\mathbf{t} = \mathbf{T}, \mathbf{R}_1, \mathbf{T}_1, \mathbf{k}_1, \dots, \mathbf{R}_n, \mathbf{T}_n, \mathbf{k}_n$.

For the next two conditions, let $\mathbf{b}_i$ 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 $\mathbf{r}_j$ 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 8), M5 introduces an implication with the ID $(c_1, c_2, \mathbf{b}_i, \mathbf{b}_j, \mathbf{0})$. This implication in turn consists of two inequalities with the IDs $(c_1, \mathbf{b}_i, \mathbf{0})$ and $(c_2, \mathbf{b}_j, \mathbf{0})$.
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:

$$\mathsf{Im}(x, y, \mathbf{b}_{i_1}, \mathbf{b}_{i_2}, \mathbf{0}) \leftarrow r_g(x, y),$$
$$\mathsf{Iq}(x, \mathbf{b}_{i_1}, \mathbf{0}) \leftarrow r_g(x, y),$$
$$\mathsf{Iq}(y, \mathbf{b}_{i_2}, \mathbf{0}) \leftarrow r_g(x, y).$$

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:

$$\mathsf{Im}(x, y, \mathbf{b}_{i_1}, \mathbf{b}_{i_2}, \mathbf{0}) \leftarrow r_g(y, x),$$
$$\mathsf{Iq}(y, \mathbf{b}_{i_1}, \mathbf{0}) \leftarrow r_g(x, y),$$
$$\mathsf{Iq}(x, \mathbf{b}_{i_2}, \mathbf{0}) \leftarrow r_g(x, y).$$

To relate the implications with the corresponding inequalities, we do the following:

$$\mathsf{Im}_{\mathsf{L}}(x, y, \mathbf{T}, \mathbf{T}', \mathbf{0}, x, \mathbf{T}, \mathbf{0}) \leftarrow \mathsf{Im}(x, y, \mathbf{T}, \mathbf{T}', \mathbf{0}), \mathsf{Iq}(x, \mathbf{T}, \mathbf{0}), \mathsf{Adom}^*(x), \mathsf{Adom}^*(y),$$

$$\mathsf{Im}_{\mathsf{R}}(x, y, \mathbf{T}, \mathbf{T}', \mathbf{0}, y, \mathbf{T}', \mathbf{0}) \leftarrow \mathsf{Im}(x, y, \mathbf{T}, \mathbf{T}', \mathbf{0}), \mathsf{Iq}(y, \mathbf{T}', \mathbf{0}), \mathsf{Adom}^*(x), \mathsf{Adom}^*(y).$$

The last step is to compute the LHS and RHS of the introduced inequalities. The LHS of the inequality with the ID $(c, \mathbf{b}_i, \mathbf{0})$ 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:

$$\mathsf{Iq}^{\mathsf{int}}_{\mathsf{L}}(x, \mathbf{b}_i, \mathbf{0}, 1) \leftarrow \mathsf{Iq}(x, \mathbf{b}_i, \mathbf{0}), \mathsf{Adom}^*(x),$$

$$\mathsf{Iq}^{\mathsf{v}}_{\mathsf{R}}(x, \mathbf{b}_i, \mathbf{0}, \mathbf{t}) \leftarrow \mathsf{Iq}(x, \mathbf{b}_i, \mathbf{0}), \mathsf{Adom}^*(x), \mathsf{Tile}(\mathbf{t}), \mathsf{In}_{B_i}(\mathbf{T}), w = x,$$

where $\mathbf{t} = \mathbf{T}, \mathbf{r}$ and $\mathbf{T} = x_1, \dots, x_{n_{\mathcal{T}}}, w$.

M6. Finally, for every tile $\tau' = (T', \rho') \in \mathsf{Tiles}(\mathcal{K})$, 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 8), M6 gives rise to an implication with the ID $(c, \mathbf{t}', \mathbf{b}_i, \mathbf{r}_j, \mathbf{0})$, which in turn consists of two inequalities with the IDs $(c, \mathbf{T}', \mathbf{b}_i, \mathbf{r}_j, \mathbf{0})$ and $(\mathbf{t}', 0)$, where $\mathbf{t}'$ encodes τ' and $\mathbf{T}'$ encodes T'.
Note that the inequalities with the ID of the form $(\mathbf{t}, 0)$, where $\mathbf{t}$ 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:

$$\mathsf{Im}(x, \mathbf{t}, \mathbf{b}_{i_1}, \mathbf{r}_h, \mathbf{0}) \leftarrow \mathsf{Tile}(\mathbf{t}), p(x, y), \mathsf{In}_{B_{i_2}}(\mathbf{T}), w = y,$$

$$\mathsf{Iq}(x, \mathbf{T}, \mathbf{b}_{i_1}, \mathbf{r}_h, \mathbf{0}) \leftarrow \mathsf{Tile}(\mathbf{t}), p(x, y), \mathsf{In}_{B_{i_2}}(\mathbf{T}), w = y,$$

where $\mathbf{t} = \mathbf{T}, \mathbf{r}$ and $\mathbf{T} = x_1, \dots, x_{n_{\mathcal{T}}}, w$.

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

$$\mathsf{Im}(x, \mathbf{t}, \mathbf{b}_{i_1}, \mathbf{r}_h, \mathbf{0}) \leftarrow \mathsf{Tile}(\mathbf{t}), p(y, x), \mathsf{In}_{B_{i_2}}(\mathbf{T}), w = y,$$

where $\mathbf{t} = \mathbf{T}, \mathbf{r}$ and $\mathbf{T} = x_1, \ldots, x_{n_\mathcal{T}}, w$.
We then relate inequalities to the corresponding implications:

$$\mathsf{Im}_\mathsf{L}(x, \mathbf{t}, \mathbf{b}_{i_1}, \mathbf{r}_h, \mathbf{0}, \mathbf{t}, \mathbf{0}) \leftarrow \mathsf{Im}(x, \mathbf{t}, \mathbf{b}_{i_1}, \mathbf{r}_h, \mathbf{0}), \mathsf{Iq}(\mathbf{t}, \mathbf{0}), \mathsf{Adom}^*(x), \mathsf{Tile}(\mathbf{t}),$$

$$\mathsf{Im}_\mathsf{R}(x, \mathbf{t}, \mathbf{b}_{i_1}, \mathbf{r}_h, \mathbf{0}, x, \mathbf{T}, \mathbf{b}_{i_1}, \mathbf{r}_h, \mathbf{0}) \leftarrow \mathsf{Im}(x, \mathbf{t}, \mathbf{b}_{i_1}, \mathbf{r}_h, \mathbf{0}), \mathsf{Iq}(x, \mathbf{T}, \mathbf{b}_{i_1}, \mathbf{r}_h, \mathbf{0}), \mathsf{Adom}^*(x), \mathsf{Tile}(\mathbf{t}),$$

where $\mathbf{t} = \mathbf{T}, \mathbf{r}$ and $\mathbf{T} = x_1, \ldots, x_{n_\mathcal{T}}, w$.
Finally, we compute the LHS and RHS of the newly-introduced inequalities. The RHS of the inequality with the ID $(c, \mathbf{T}', \mathbf{b}_i, \mathbf{r}_h, \mathbf{0})$ 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 $B_i \in \mathsf{N}_\mathsf{C}(\mathcal{K})$ and $r_h \in \mathsf{N}_\mathsf{R}^+(\mathcal{K})$, we add:

$$\mathsf{Iq}^{\mathsf{int}}_\mathsf{R}(x, \mathbf{T}', \mathbf{b}_i, \mathbf{r}_h, \mathbf{0}, \mathbf{k}) \leftarrow \mathsf{Iq}(x, \mathbf{T}', \mathbf{b}_i, \mathbf{r}_h, \mathbf{0}), \mathsf{Adom}^*(x), \mathsf{Int}(\mathbf{k}), \mathbf{k} = (0, \ldots, 0),$$

$$\mathsf{Iq}^{\mathsf{v}}_\mathsf{L}(x, \mathbf{T}', \mathbf{b}_i, \mathbf{r}_h, \mathbf{0}, \mathbf{t}) \leftarrow \mathsf{Iq}(x, \mathbf{T}', \mathbf{b}_i, \mathbf{r}_h, \mathbf{0}), \mathsf{Tile}(\mathbf{t}), \mathsf{Adom}^*(x),$$
$$w = x, \mathsf{In}_{B_{i_1}}(\mathbf{T}), \mathsf{Upto}_{n, r_h}(\mathbf{t}, \mathbf{T}, 0, \ldots, 0),$$

for all $r_h \in \mathsf{N}_\mathsf{R}^+(\mathcal{K})$, where $\mathbf{t} = \mathbf{T}, \mathbf{r}$ and $\mathbf{T} = x_1, \ldots, x_{n_\mathcal{T}}, w$.
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 $\mathbf{t}$, T' be a type identified by $\mathbf{T}'$, r_h be a role in $\mathsf{N}_\mathsf{R}^+(\mathcal{K})$, and k be an integer whose binary representation $\mathbf{k}$ is stored in Int. The relation Upto_{i, r_h} stores the tuple $(\mathbf{t}, \mathbf{T}', \mathbf{k})$ 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:

$$\mathsf{Upto}_{0, r_h}(\mathbf{t}, \mathbf{T}', \mathbf{0}) \leftarrow \mathsf{Tile}(\mathbf{t}), \mathsf{Type}(\mathbf{T}')$$

$$\mathsf{Upto}_{i, r_h}(\mathbf{t}, \mathbf{T}', \mathbf{k}) \leftarrow \mathsf{Upto}_{i-1, r_h}(\mathbf{t}, \mathbf{T}', \mathbf{k}'), \mathsf{Succ}_{\mathsf{int}}(\mathbf{k}', \mathbf{k}), \mathbf{T}_i = \mathbf{T}', \mathsf{In}_{r_h}(\mathbf{R}_i)$$

$$\mathsf{Upto}_{i, r_h}(\mathbf{t}, \mathbf{T}', \mathbf{k}) \leftarrow \mathsf{Upto}_{i-1, r_h}(\mathbf{t}, \mathbf{T}', \mathbf{k}), \mathit{not}\ \mathbf{T}_i \neq \mathbf{T}'$$

$$\mathsf{Upto}_{i, r_h}(\mathbf{t}, \mathbf{T}', \mathbf{k}) \leftarrow \mathsf{Upto}_{i-1, r_h}(\mathbf{t}, \mathbf{T}', \mathbf{k}), \mathit{not}\ \mathsf{In}_{r_h}(\mathbf{R}_i),$$

for each $1 \leq i \leq n$ and counting role $r_h \in \mathsf{N}_\mathsf{R}^+(\mathcal{K})$, where $\mathbf{t} = \mathbf{T}, \mathbf{R}_1, \mathbf{T}_1, \mathbf{k}_1, \ldots, \mathbf{R}_n, \mathbf{T}_n, \mathbf{k}_n$, and $\mathbf{T} = x_1, \ldots, x_{n_\mathcal{T}}, w$.

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 closed predicates & functionality assertions To complete our construction of $\mathcal{P}^{\mathcal{T}, \Sigma}_{sys}$, we add the rules that ensure that $\mathcal{K}$ satisfies conditions 1-4 in Theorem 1. 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 $r \in \Sigma \cap \mathsf{N}_\mathsf{R}$ and all $s \sqsubseteq r \in \mathcal{T}$, we add
$$\leftarrow s(x, y), \mathit{not}\ r(x, y).$$
4. For all $r \in \Sigma \cap \mathsf{N}_\mathsf{R}$ 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 $\mathcal{P}^{\mathcal{T}, \Sigma}_{sys}$. First, the arities of the predicates occurring in $\mathcal{P}^{\mathcal{T}, \Sigma}_{sat}$ are polynomial in the size of $\mathcal{T}$ and Σ and do not depend on $\mathcal{A}$ (see Table 3). Moreover, looking at the encoding above, we can easily verify that the number of non-ground rules in $\mathcal{P}^{\mathcal{T}, \Sigma}_{sat}$ 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 $Rel(S_{\mathcal{K}})$. 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 Theorem 1. This leads us to the following statement.

Proposition 5. *Given a TBox $\mathcal{T}$ and a set Σ of closed predicates, $\mathcal{P}_{sat}^{\mathcal{T},\Sigma}$ is polynomial in the size of $\mathcal{T}$ and Σ, and does not depend on the size of the input ABox. Moreover, for any ABox $\mathcal{A}$ over the signature of $\mathcal{T}$, if $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$ does not satisfy one of the conditions 1-4 in Theorem 1, then $\mathcal{P}_{sat}^{\mathcal{T},\Sigma} \cup \hat{\mathcal{A}}$ has no answer sets. Otherwise, each answer set of $\mathcal{P}_{sat}^{\mathcal{T},\Sigma} \cup \hat{\mathcal{A}}$ corresponds to $Rel(S_{\mathcal{K}})$, up to the renaming of IDs of the variables, implications and inequalities.*

4.2.2. 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 $\mathcal{P}_{sys}^{\mathcal{T},\Sigma} \cup \hat{\mathcal{A}}$ 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.

We begin by recalling a well-known result in integer programming 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 [42]. This result can easily be generalized to solutions over $\mathbb{N}^*$ (cf. Lemma 18 in [38]) and it also holds for enriched systems of linear inequalities.

Theorem 2. *Let $(V, \mathcal{E}, I)$ be a finite enriched system of linear inequalities in which all constants and coefficients are in $\{0, \pm 1, \dots, \pm a\}$. 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 $(|V| + |I| + |\mathcal{E}|) \cdot ((|\mathcal{E}| + |I|) \cdot a)^{2(|\mathcal{E}|+|I|)+1}$.*

Proof *(Proof sketch).* To decide whether an enriched system of linear inequalities $(V, \mathcal{E}, I)$ has a solution over $\mathbb{N}^*$, we can construct a set of ordinary linear inequality systems and check whether at least one of them has a solution over $\mathbb{N}^*$. Each system in this set is of the form $(V, \mathcal{E} \cup \hat{\mathcal{E}})$, where $\hat{\mathcal{E}}$ is obtained by adding to $\mathcal{E}$ either α or β, for each implication $\alpha \implies \beta$ in I. Thus, to solve these systems it is enough to consider the solutions whose finite values are bounded by $(|V| + |\hat{\mathcal{E}} \cup \mathcal{E}|) \cdot ((|\mathcal{E} \cup \hat{\mathcal{E}}|) \cdot a)^{2(|\mathcal{E} \cup \hat{\mathcal{E}}|)+1}$ [38]. The result of the theorem follows from $|\hat{\mathcal{E}}| = |I|$.

Theorem 2 essentially says that there is a finite search space that we have to explore in order to determine whether an enriched system has a solution. The program $\mathcal{P}_{sol}^{\mathcal{T},\Sigma}$ does this non-deterministically in a 'guess-and-check' manner, i.e., it guesses a potential solution and checks that the guess is valid. However, the solutions that $\mathcal{P}_{sol}^{\mathcal{T},\Sigma}$ has to consider are very large - exponential in the size of the given system. Moreover, for a given KB $\mathcal{K} = (\mathcal{T}, \Sigma, \mathcal{A})$, the solutions of $S_{\mathcal{K}}$ are doubly exponential in the size of $\mathcal{T}$. As we would like to keep $\mathcal{P}_{sol}^{\mathcal{T},\Sigma}$ polynomial in the size of $\mathcal{T}$, 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 is 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 Theorem 2, 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 $\mathcal{S}$ is to associate to each variable whose value in $\mathcal{S}$ is finite, a binary string of length d^{3l} that encodes this value. However this makes the translation both exponential as well as and 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: $(\mathbf{x}, \mathbf{z}, b)$ in the relation Val denotes that in the value of the variable encoded by $\mathbf{x}$ the bit at the position $\mathbf{z}$ has the value b, where b is either 0 or 1. This is depicted in Fig. 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 $\mathsf{Cst}^i(x_1, \dots, x_i)$ to abbreviate $\mathsf{Cst}(x_1), \dots \mathsf{Cst}(x_i)$, for $i \geq 1$. We begin by adding the rules that guess which variables in a potential solution are set to infinity:

$$\mathsf{Fin}(\mathbf{x}) \leftarrow \mathsf{Var}(\mathbf{x}), \mathit{not}\ \mathsf{Inf}(\mathbf{x}), \qquad \mathsf{Inf}(\mathbf{x}) \leftarrow \mathsf{Var}(\mathbf{x}), \mathit{not}\ \mathsf{Fin}(\mathbf{x}).$$

If a variable is not set to infinity, we separately guess each bit of its value by using the following rules:

$$\mathsf{Val}(\mathbf{x}, \mathbf{z}', \mathbf{z}, 0) \leftarrow \mathsf{Fin}(\mathbf{x}), \mathsf{Cst}^l(\mathbf{z}'), \mathsf{Cst}^{2l}(\mathbf{z}), \mathbf{z}' \neq (0, \dots, 0),$$

$$\mathsf{Val}(\mathbf{x}, \mathbf{z}, 0) \leftarrow \mathsf{Fin}(\mathbf{x}), \mathsf{Cst}^{3l}(\mathbf{z}), \mathit{not}\ \mathsf{Val}(\mathbf{x}, \mathbf{z}, 1),$$

$$\mathsf{Val}(\mathbf{x}, \mathbf{z}, 1) \leftarrow \mathsf{Fin}(\mathbf{x}), \mathsf{Cst}^{3l}(\mathbf{z}), \mathit{not}\ \mathsf{Val}(\mathbf{x}, \mathbf{z}, 0).$$

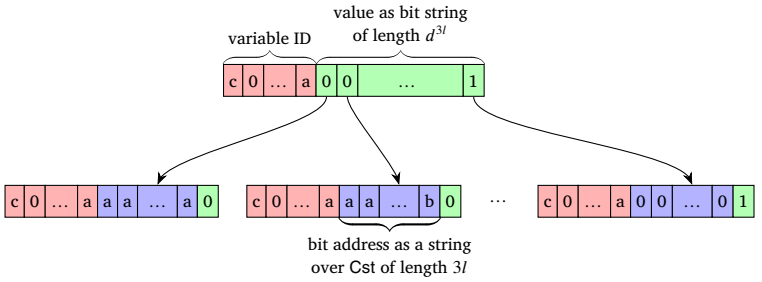


Fig. 3. Naive (top) vs. our encoding (bottom) of solutions in $\mathcal{P}_{sol}^{\mathcal{T},\Sigma}$.

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 9), checking whether inequalities with occurrences of $\aleph_0$ (i.e., infinity) are satisfied is easy – such an inequality is satisfied if and only if infinity occurs on its RHS. We thus have the following rule:

$$\mathsf{Sat}(\mathbf{y}) \leftarrow \mathsf{Var}(\mathbf{x}), \mathsf{Iq}(\mathbf{y}), \mathsf{Iq}_{\mathsf{R}}(\mathbf{y}, \mathbf{x}), \mathsf{Inf}(\mathbf{x}).$$

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:

$$\mathsf{InfIq}(\mathbf{y}) \leftarrow \mathsf{Iq}(\mathbf{y}), \mathsf{Var}(\mathbf{x}), \mathsf{Inf}(\mathbf{x}), \mathsf{Iq}_{\mathsf{L}}(\mathbf{y}, \mathbf{x}),$$

$$\mathsf{InfIq}(\mathbf{y}) \leftarrow \mathsf{Iq}(\mathbf{y}), \mathsf{Var}(\mathbf{x}), \mathsf{Inf}(\mathbf{x}), \mathsf{Iq}_{\mathsf{R}}(\mathbf{y}, \mathbf{x}),$$

$$\mathsf{FinIq}(\mathbf{y}) \leftarrow \mathsf{Iq}(\mathbf{y}), \mathit{not}\ \mathsf{InfIq}(\mathbf{y}).$$

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 $\mathsf{Upto}_{\mathsf{L}}^{\mathsf{v}}$, $\mathsf{Upto}_{\mathsf{R}}^{\mathsf{v}}$, $\mathsf{Upto}_{\mathsf{L}}^{\mathsf{int}}$ and $\mathsf{Upto}_{\mathsf{R}}^{\mathsf{int}}$. Intuitively, $\mathsf{Upto}_{\mathsf{L}}^{\mathsf{v}}$ (resp. $\mathsf{Upto}_{\mathsf{R}}^{\mathsf{v}}$) stores a tuple $(\mathbf{q}, \mathbf{v}, \mathbf{p}, b)$ if and only if the bit with the address $\mathbf{p}$ in the sum of all variables up to and including the variable $\mathbf{v}$ that occur on the LHS (resp. on the RHS) of the inequality $\mathbf{q}$ 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 $\mathsf{Upto}_{\mathsf{L}}^{\mathsf{int}}$ stores a tuple $(\mathbf{q}, \mathbf{k}, \mathbf{p}, b)$, if the bit with the address $\mathbf{p}$ in the sum of all variables and all integers occurring on the LHS of the inequality $\mathbf{q}$ up to $\mathbf{k}$ (including $\mathbf{k}$) has the value b.

For iterating through the variables and integers, we use the linear in 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 $(\mathbf{k}, \mathbf{p}, b)$ if and only if for the integer $\mathbf{k}$, the bit at the position $\mathbf{p}$ is set to b. To compute this relation, we rely on the auxiliary relations $\mathsf{Val}_i^{\mathsf{int}}$, for each $1 \leq i \leq l_a$, and $\mathsf{Val}_{>l_a}^{\mathsf{int}}$ that are computed as follows:

$$\mathsf{Val}_1^{\mathsf{int}}(\mathbf{x}, \mathbf{z}, x_1) \leftarrow \mathsf{Int}(\mathbf{x}), \mathsf{First}^{3l}(\mathbf{z}),$$

$$\mathsf{Val}_i^{\mathsf{int}}(\mathbf{x}, \mathbf{z}, x_i) \leftarrow \mathsf{Val}_{i-1}^{\mathsf{int}}(\mathbf{x}, \mathbf{z}', x'), \mathsf{Succ}^{3l}(\mathbf{z}', \mathbf{z}),$$

$$\mathsf{Val}_{>l_a}^{\mathsf{int}}(\mathbf{x}, \mathbf{z}, 0) \leftarrow \mathsf{Val}_{l_a}^{\mathsf{int}}(\mathbf{x}, \mathbf{z}', \mathbf{x}'), \mathsf{LEQ}^{3l}(\mathbf{z}', \mathbf{z}),$$

for all $1 < i \leq l_a$, where $\mathbf{x} = x_{l_a}, \ldots, x_1$. Intuitively, the relation $\mathsf{Val}_i^{\mathsf{int}}$, $1 \leq i \leq l_a$, 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 $\mathsf{Val}_{>l_a}^{\mathsf{int}}$. We then simply copy the information stored in the auxiliary relations into the relation $\mathsf{Val}^{\mathsf{int}}$ using the following rules, for all $1 \leq i \leq l_a$:

$$\mathsf{Val}^{\mathsf{int}}(\mathbf{x}, \mathbf{z}, x) \leftarrow \mathsf{Val}_i^{\mathsf{int}}(\mathbf{x}, \mathbf{z}, x),$$

$$\mathsf{Val}^{\mathsf{int}}(\mathbf{x},\mathbf{z},x) \leftarrow \mathsf{Val}^{\mathsf{int}}_{>l_a}(\mathbf{x},\mathbf{z},x).$$

Finally, we need to add the facts and rules that define bitwise 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 $(\mathbf{q},\mathbf{x},\mathbf{p},c)$ is in $\mathsf{Carry}^{\mathsf{v}}_{\mathsf{L}}$ (resp. $\mathsf{Carry}^{\mathsf{int}}_{\mathsf{L}}$) if when adding the bit at the position $\mathbf{p}$ of the variable (resp. integer) $\mathbf{x}$ to the result we have obtained so far for the LHS of the inequality $\mathbf{q}$, 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 the 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:

$$\mathsf{Upto}^{\mathsf{v}}_{\mathsf{L}}(\mathbf{y},\mathbf{0},\mathbf{x},\mathbf{z},0) \leftarrow \mathsf{FinIq}(\mathbf{y}), \mathsf{First}_{\mathsf{V}}(\mathbf{x}), \mathit{not}\ \mathsf{Iq}^{\mathsf{v}}_{\mathsf{L}}(\mathbf{y},\mathbf{x}), \mathsf{Cst}^{3l}(\mathbf{z}),$$

$$\mathsf{Upto}^{\mathsf{v}}_{\mathsf{L}}(\mathbf{y},\mathbf{0},\mathbf{x},\mathbf{z},x) \leftarrow \mathsf{FinIq}(\mathbf{y}), \mathsf{First}_{\mathsf{V}}(\mathbf{x}), \mathsf{Val}(\mathbf{x},\mathbf{z},x), \mathsf{Iq}^{\mathsf{v}}_{\mathsf{L}}(\mathbf{y},\mathbf{x}).$$

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:

$$\mathsf{Carry}^{\mathsf{v}}_{\mathsf{L}}(\mathbf{y},\mathbf{0},\mathbf{x},\mathbf{z},0) \leftarrow \mathsf{FinIq}(\mathbf{y}), \mathsf{Var}(\mathbf{x}), \mathsf{First}^{3l}(\mathbf{z}).$$

We then add the rules that perform the bitwise addition of the sum we have so far and the next variable with respect to $\mathsf{Succ}_{\mathsf{V}}$:

$$\begin{aligned}
\mathsf{Upto}^{\mathsf{v}}_{\mathsf{L}}(\mathbf{y},\mathbf{0},\mathbf{x},\mathbf{z},x') &\leftarrow \mathsf{Upto}^{\mathsf{v}}_{\mathsf{L}}(\mathbf{y},\mathbf{0},\mathbf{x}',\mathbf{z},x'), \mathsf{Succ}_{\mathsf{V}}(\mathbf{x}',\mathbf{x}), \mathit{not}\ \mathsf{Iq}^{\mathsf{v}}_{\mathsf{L}}(\mathbf{y},\mathbf{x}),\\
\mathsf{Upto}^{\mathsf{v}}_{\mathsf{L}}(\mathbf{y},\mathbf{0},\mathbf{x},\mathbf{z},x) &\leftarrow \mathsf{Upto}^{\mathsf{v}}_{\mathsf{L}}(\mathbf{y},\mathbf{0},\mathbf{x}',\mathbf{z},x'), \mathsf{Succ}_{\mathsf{V}}(\mathbf{x}',\mathbf{x}), \mathsf{Iq}^{\mathsf{v}}_{\mathsf{L}}(\mathbf{y},\mathbf{x}), \mathsf{Val}(\mathbf{x},\mathbf{z},x''),\\
&\qquad \mathsf{Carry}^{\mathsf{v}}_{\mathsf{L}}(\mathbf{y},\mathbf{0},\mathbf{x},\mathbf{z},z), \mathsf{Add}(x',x'',z,y,x),\\
\mathsf{Carry}^{\mathsf{v}}_{\mathsf{L}}(\mathbf{y},\mathbf{0},\mathbf{x},\mathbf{z}',y) &\leftarrow \mathsf{Upto}^{\mathsf{v}}_{\mathsf{L}}(\mathbf{y},\mathbf{0},\mathbf{x}',\mathbf{z},x'), \mathsf{Succ}_{\mathsf{V}}(\mathbf{x}',\mathbf{x}), \mathsf{Iq}^{\mathsf{v}}_{\mathsf{L}}(\mathbf{y},\mathbf{x}), \mathsf{Val}(\mathbf{x},\mathbf{z},x''), \mathsf{Succ}^{3l}(\mathbf{z},\mathbf{z}'),\\
&\qquad \mathsf{Carry}^{\mathsf{v}}_{\mathsf{L}}(\mathbf{y},\mathbf{0},\mathbf{x},\mathbf{z},z), \mathsf{Add}(x',x'',z,y,x),
\end{aligned}$$

where $\mathbf{0}$ is the 0-vector of length $\max(l_v,l_a) - l_v$.

Similarly, the relation $\mathsf{Upto}^{\mathsf{int}}_{\mathsf{L}}$ is computed as follows:

$$\begin{aligned}
\mathsf{Upto}^{\mathsf{int}}_{\mathsf{L}}(\mathbf{y},\mathbf{0}_1,\mathbf{x},\mathbf{z},x) &\leftarrow \mathsf{FinIq}(\mathbf{y}), \mathsf{First}_{\mathsf{int}}(\mathbf{x}), \mathsf{Upto}^{\mathsf{v}}_{\mathsf{L}}(\mathbf{y},\mathbf{0}_2,\mathbf{x}',\mathbf{z},x), \mathsf{Last}_{\mathsf{V}}(\mathbf{x}'), \mathit{not}\ \mathsf{Iq}^{\mathsf{int}}_{\mathsf{L}}(\mathbf{y},\mathbf{x}),\\
\mathsf{Carry}^{\mathsf{int}}_{\mathsf{L}}(\mathbf{y},\mathbf{0},\mathbf{x},\mathbf{z},0) &\leftarrow \mathsf{FinIq}(\mathbf{y}), \mathsf{Int}(\mathbf{x}), \mathsf{First}^{3l}(\mathbf{z}),\\
\mathsf{Upto}^{\mathsf{int}}_{\mathsf{L}}(\mathbf{y},\mathbf{0}_1,\mathbf{x},\mathbf{z},x) &\leftarrow \mathsf{FinIq}(\mathbf{y}), \mathsf{First}_{\mathsf{int}}(\mathbf{x}), \mathsf{Iq}^{\mathsf{int}}_{\mathsf{L}}(\mathbf{y},\mathbf{x}), \mathsf{Upto}^{\mathsf{v}}_{\mathsf{L}}(\mathbf{y},\mathbf{0}_2,\mathbf{x}',\mathbf{z},x'), \mathsf{Last}_{\mathsf{V}}(\mathbf{x}'), \mathsf{Val}^{\mathsf{int}}(\mathbf{x},\mathbf{z},x''),\\
&\qquad \mathsf{Carry}^{\mathsf{int}}_{\mathsf{L}}(\mathbf{y},\mathbf{0}_1,\mathbf{x},\mathbf{z},z), \mathsf{Add}(x',x'',z,y,x),\\
\mathsf{Carry}^{\mathsf{int}}_{\mathsf{L}}(\mathbf{y},\mathbf{0}_1,\mathbf{x},\mathbf{z}',y) &\leftarrow \mathsf{FinIq}(\mathbf{y}), \mathsf{First}_{\mathsf{int}}(\mathbf{x}), \mathsf{Iq}^{\mathsf{int}}_{\mathsf{L}}(\mathbf{y},\mathbf{x}), \mathsf{Upto}^{\mathsf{v}}_{\mathsf{L}}(\mathbf{y},\mathbf{0}_2,\mathbf{x}',\mathbf{z},x'), \mathsf{Last}_{\mathsf{V}}(\mathbf{x}'), \mathsf{Val}^{\mathsf{int}}(\mathbf{x},\mathbf{z},x''),\\
&\qquad \mathsf{Succ}^{3l}(\mathbf{z},\mathbf{z}'), \mathsf{Carry}^{\mathsf{int}}_{\mathsf{L}}(\mathbf{y},\mathbf{0}_1,\mathbf{x},\mathbf{z},z), \mathsf{Add}(x',x'',z,y,x),\\
\mathsf{Upto}^{\mathsf{int}}_{\mathsf{L}}(\mathbf{y},\mathbf{0}_1,\mathbf{x},\mathbf{z},x') &\leftarrow \mathsf{Upto}^{\mathsf{int}}_{\mathsf{L}}(\mathbf{y},\mathbf{0}_1,\mathbf{x}',\mathbf{z},x'), \mathsf{Succ}_{\mathsf{int}}(\mathbf{x}',\mathbf{x}), \mathit{not}\ \mathsf{Iq}^{\mathsf{int}}_{\mathsf{L}}(\mathbf{y},\mathbf{x}),\\
\mathsf{Upto}^{\mathsf{int}}_{\mathsf{L}}(\mathbf{y},\mathbf{0}_1,\mathbf{x},\mathbf{z},x) &\leftarrow \mathsf{Upto}^{\mathsf{int}}_{\mathsf{L}}(\mathbf{y},\mathbf{0}_1,\mathbf{x}',\mathbf{z},x'), \mathsf{Succ}_{\mathsf{int}}(\mathbf{x}',\mathbf{x}), \mathsf{Iq}^{\mathsf{int}}_{\mathsf{L}}(\mathbf{y},\mathbf{x}), \mathsf{Val}(\mathbf{x},\mathbf{z},x''),\\
&\qquad \mathsf{Carry}^{\mathsf{int}}_{\mathsf{L}}(\mathbf{y},\mathbf{0}_1,\mathbf{x},\mathbf{z},z), \mathsf{Add}(x',x'',z,y,x),\\
\mathsf{Carry}^{\mathsf{int}}_{\mathsf{L}}(\mathbf{y},\mathbf{0}_1,\mathbf{x},\mathbf{z}',y) &\leftarrow \mathsf{Upto}^{\mathsf{int}}_{\mathsf{L}}(\mathbf{y},\mathbf{0}_1,\mathbf{x}',\mathbf{z},x'), \mathsf{Succ}_{\mathsf{int}}(\mathbf{x}',\mathbf{x}), \mathsf{Iq}^{\mathsf{int}}_{\mathsf{L}}(\mathbf{y},\mathbf{x}), \mathsf{Val}(\mathbf{x},\mathbf{z},x''), \mathsf{Succ}^{3l}(\mathbf{z},\mathbf{z}'),\\
&\qquad \mathsf{Carry}^{\mathsf{int}}_{\mathsf{L}}(\mathbf{y},\mathbf{0}_1,\mathbf{x},\mathbf{z},z), \mathsf{Add}(x',x'',z,y,x),
\end{aligned}$$

where $\mathbf{0}_1$ is the 0-vector of the length $l_a - \max(l_v,l_a)$ and $\mathbf{0}_2$ is the 0-vector of the length $l_v - \max(l_v,l_a)$. The relations $\mathsf{Upto}^{\mathsf{v}}_{\mathsf{R}}$ and $\mathsf{Upto}^{\mathsf{int}}_{\mathsf{R}}$ are computed analogously.

We next store the final result of our computation using the relations LHS and RHS as follows:

$$\mathsf{LHS}(\mathbf{y},\mathbf{z},b) \leftarrow \mathsf{Upto}^{\mathsf{int}}_{\mathsf{L}}(\mathbf{y},\mathbf{0},\mathbf{x},\mathbf{z},b), \mathsf{Last}_{\mathsf{int}}(\mathbf{x}),$$

$$\mathsf{RHS}(\mathbf{y},\mathbf{z},b) \leftarrow \mathsf{Upto}^{\mathsf{int}}_{\mathsf{R}}(\mathbf{y},\mathbf{0},\mathbf{x},\mathbf{z},b), \mathsf{Last}_{\mathsf{int}}(\mathbf{x}),$$

where $\mathbf{0}$ 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 lq, and those inequalities that must be satisfied, stored in the relation lq^*. We next add the rules that check whether all the inequalities in lq^* are indeed satisfied by comparing the LHS and RHS of inequalities, bit by bit:

$$\mathsf{Sat}(\mathbf{y}) \leftarrow \mathsf{LHS}(\mathbf{y},\mathbf{z},c), \mathsf{RHS}(\mathbf{y},\mathbf{z},c'), \mathsf{Last}^{3l}(\mathbf{z}), \mathsf{LEQ}(c,c'), c \neq c',$$

$$\mathsf{Sat}'(\mathbf{y},\mathbf{z}) \leftarrow \mathsf{LHS}(\mathbf{y},\mathbf{z},c), \mathsf{RHS}(\mathbf{y},\mathbf{z},c'), \mathsf{Last}^{3l}(\mathbf{z}), c = c',$$

$$\mathsf{Sat}(\mathbf{y}) \leftarrow \mathsf{LHS}(\mathbf{y},\mathbf{z}',c), \mathsf{RHS}(\mathbf{y},\mathbf{z}',c'), \mathsf{Sat}'(\mathbf{y},\mathbf{z}), \mathsf{Succ}^{3l}(\mathbf{z}',\mathbf{z}), \mathsf{LEQ}(c,c'), c \neq c',$$

$$\mathsf{Sat}'(\mathbf{y},\mathbf{z}) \leftarrow \mathsf{Sat}'(\mathbf{y},\mathbf{z}'), \mathsf{Succ}^{3l}(\mathbf{z},\mathbf{z}'), \mathsf{LHS}(\mathbf{y},\mathbf{z},c), \mathsf{RHS}(\mathbf{y},\mathbf{z},c'), c = c',$$

$$\mathsf{Sat}(\mathbf{y}) \leftarrow \mathsf{Sat}'(\mathbf{y},\mathbf{z}), \mathsf{First}^{3l}(\mathbf{z}),$$

$$\leftarrow \mathsf{lq}^*(\mathbf{y}), \mathit{not}\ \mathsf{Sat}(\mathbf{y}).$$

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 $\mathbf{y}$ cannot be satisfied and indeed, due to the stable model semantics, we are not able to derive $\mathsf{Sat}(\mathbf{y})$.

Finally, we finish the construction of $\mathcal{P}^{\mathcal{T},\Sigma}_{sol}$ by adding the constraints that ensure that all the implications are satisfied:

$$\leftarrow \mathsf{Im}_{\mathsf{L}}(\mathbf{x},\mathbf{y}), \mathsf{Im}_{\mathsf{R}}(\mathbf{x},\mathbf{z}), \mathsf{lq}(\mathbf{y}), \mathsf{lq}(\mathbf{z}), \mathsf{Sat}(\mathbf{y}), \mathit{not}\ \mathsf{Sat}(\mathbf{z}).$$

The construction above makes sure that $\mathcal{P}^{\mathcal{T},\Sigma}_{sol}$ has the following properties. Firstly, just like before, all predicates occurring in $\mathcal{P}^{\mathcal{T},\Sigma}_{sol}$ 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 $\mathcal{P}^{\mathcal{T},\Sigma}_{sol}$ – 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 $\mathcal{P}^{\mathcal{T},\Sigma}_{sol}$ 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 $\mathcal{S}$ over $\mathbb{N}^*$ to $\mathcal{S}$ such that all finite values are bounded by $2^{d^{2l}}$, then there is an answer set of $\mathcal{P}^{\mathcal{T},\Sigma}_{sol} \cup \mathit{Rel}(\mathcal{S})$ that corresponds to this solution. To see this last point, observe that for every variable, $\mathcal{P}^{\mathcal{T},\Sigma}_{sol}$ freely guesses any value between 0 and $2^{d^{2l}}$. The observations above in conjunction with Theorem 2 give rise to the following proposition:

Proposition 6. *Given an enriched system $\mathcal{S}$ with the relational representation $\mathit{Rel}(\mathcal{S})$, $\mathcal{S}$ has a solution over $\mathbb{N}^*$ iff $\mathcal{P}^{\mathcal{T},\Sigma}_{sys} \cup \mathit{Rel}(\mathcal{S})$ has an answer set. Moreover, $\mathcal{P}^{\mathcal{T},\Sigma}_{sol}$ is polynomial in the size of $\mathcal{T}$ and Σ.*

Recall that every answer set of $\mathcal{P}^{\mathcal{T},\Sigma}_{sys} \cup \hat{\mathcal{A}}$ corresponds to the relational representation of $\mathcal{S}$, for a given ABox $\mathcal{A}$ and $\mathcal{K} = (\mathcal{T},\Sigma,\mathcal{A})$. Thus, in view of Proposition 6, we have the following result:

Proposition 7. *Given an input ABox $\mathcal{A}$ over the signature of $\mathcal{T}$, let $\mathcal{K} = (\mathcal{T},\Sigma,\mathcal{A})$. The program $\mathcal{P}^{\mathcal{T},\Sigma}_{sol}$ 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 $\mathcal{P}^{\mathcal{T},\Sigma}_{sys} \cup \hat{\mathcal{A}}$ such that $\mathcal{P}^{\mathcal{T},\Sigma}_{sol} \cup I$ has an answer set.*

Theorem 3. *For a TBox $\mathcal{T}$ and $\Sigma \subseteq \mathsf{N}_C \cup \mathsf{N}_R$, we can obtain a program $\mathcal{P}^{\mathcal{T},\Sigma}_{sat}$ in polynomial time such that $\mathcal{P}^{\mathcal{T},\Sigma}_{sat} \cup \hat{\mathcal{A}}$ 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 $\mathcal{P}^{\mathcal{T},\Sigma}_{sat}$ is simply the union of $\mathcal{P}^{\mathcal{T},\Sigma}_{sys}$ and $\mathcal{P}^{\mathcal{T},\Sigma}_{sol}$, both of which can be obtained in polynomial time from $\mathcal{T}$ and Σ (Propositions 5 and 7). Moreover, all constants of $\mathcal{P}^{\mathcal{T},\Sigma}_{sol}$ occur also in $\mathcal{P}^{\mathcal{T},\Sigma}_{sys}$, the predicates occurring in $\hat{\mathcal{A}}$ are EDB predicates of both $\mathcal{P}^{\mathcal{T},\Sigma}_{sys}$ and $\mathcal{P}^{\mathcal{T},\Sigma}_{sol}$, and all shared predicates of $\mathcal{P}^{\mathcal{T},\Sigma}_{sys}$ and $\mathcal{P}^{\mathcal{T},\Sigma}_{sol}$ are EDB predicates in $\mathcal{P}^{\mathcal{T},\Sigma}_{sol}$. Due to Proposition 4, the answer sets of $\mathcal{P}^{\mathcal{T},\Sigma}_{sat} \cup \hat{\mathcal{A}}$ correspond to the set $\{I : I \text{ is an answer set of } \mathcal{P}^{\mathcal{T},\Sigma}_{sol} \cup J, \text{ for some answer set } J \text{ of } \mathcal{P}^{\mathcal{T},\Sigma}_{sys} \cup \hat{\mathcal{A}}\}$. Finally, combining this with the result from Proposition 7, we get that $(\mathcal{T},\Sigma,\mathcal{A})$ is satisfiable, if $\mathcal{P}^{\mathcal{T},\Sigma}_{sat} \cup \hat{\mathcal{A}}$ has an answer set.

5. Query answering 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 answered via a Datalog$^\neg$ program.

To this end, we first recall the notion of a database query expressed in FO logic. Let N_V be a countably infinite set of *variables*. A *first-order query (FO query)* q is simply a first-order formula over the predicates in $\mathsf{N}_\mathsf{C} \cup \mathsf{N}_\mathsf{R}$, the variables from N_V, and the constants from N_I. We call the free variables of q the *answer variables* and we may write $q(\mathbf{x})$ to denote a FO query q with answer variables $\mathbf{x}$. Given an interpretation $\mathcal{I}$, a *match* π for a FO query q with answer variables $\mathbf{x} = (x_1,\ldots,x_n)$ in $\mathcal{I}$ is a function that maps every constant to itself and every free variable of q onto an element of $\Delta^{\mathcal{I}}$ such that $\mathcal{I}$ satisfies the query obtained from q by substituting

$\pi(x)$ for each free variable x in q. A tuple $\mathbf{a} = (a_1, \dots, a_n)$ of domain elements in $\Delta^{\mathcal{I}}$ is an *answer to* $\mathbf{q}(\mathbf{x})$ *in* $\mathcal{I}$ if there exists a match π for q in $\mathcal{I}$ such that $\pi(x_i) = a_i$, for $1 \leq i \leq n$.

In our setting, an *ontology-mediated query* (OMQ) 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. Given an ABox $\mathcal{A}$, a tuple of constants $\mathbf{a} = (a_1, \dots, a_n)$ from $\mathsf{N_I}(\mathcal{A} \cup \mathcal{T})$ is a *certain answer* to Q over $\mathcal{A}$, if $\mathbf{a}$ is an answer to q in every model $\mathcal{I}$ of $(\mathcal{T}, \Sigma, \mathcal{A})$. The *query answering problem* is the problem of deciding, given an OMQ Q, an ABox $\mathcal{A}$, and a tuple of constants $\mathbf{a}$, whether $\mathbf{a}$ is a certain answer to Q over $\mathcal{A}$.

Example 4. Let $Q = (\mathcal{T}, \{B\}, q(x))$ be an OMQ, where

$$\mathcal{T} = \{A \sqsubseteq\!= 1r.B, C \sqsubseteq\!= 1r.B, A \sqcap C \sqsubseteq \bot\}$$

$$q(x) = \exists y \exists z.r(y, x) \wedge r(z, x) \wedge A(y) \wedge C(z)$$

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 $\mathcal{A}' = \mathcal{A} \cup \{B(d)\}$.

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 $\mathcal{T}$.

We say that an OMQ $Q = (\mathcal{T}, \Sigma, q(\mathbf{x}))$ is *Datalog$^{\neg}$-rewritable* if we can find a Datalog$^{\neg}$ query whose certain answers over $\hat{\mathcal{A}}$ coincide with the certain answers of Q over $\mathcal{A}$, for any ABox $\mathcal{A}$ over the signature of $\mathcal{T}$. In the remainder of this section we define a large class of OMQs that are Datalog$^{\neg}$-rewritable.

5.1. 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 mediated, 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* that 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 $\mathcal{A} \cup \mathcal{T}$ 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 [43] 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 schemas* [44,45] and *existential rules with closed variables* [46].

We begin by formally defining safe-range OMQs. Much like the traditional approach for safe-range FO queries, we provide a multi-step syntactic characterization of safe-range OMQs consisting of the following:

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';
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 [47]) 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 $\forall \mathbf{x}\psi$ by $\neg\exists\mathbf{x}\neg\psi$.
Eliminate implications and equivalences: Replace $\psi \rightarrow \xi$ 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 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 [47] 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 11. An OMQ $Q = (\mathcal{T}, \Sigma, q)$ is *safe-range* if $\mathsf{rr}(\mathsf{SRNF}(q), \Sigma) \neq {}'fail'$.

Example 5. Consider again the OMQ $Q = (\mathcal{T}, \Sigma, q(x))$ from Example 4, where

$$\mathcal{T} = \{A \sqsubseteq = 1r.B, C \sqsubseteq = 1r.B, A \sqcap C \sqsubseteq \bot\},$$

$$\Sigma = \{B\},$$

$$q(x) = \exists y \exists z.r(y,x) \wedge r(z,x) \wedge A(y) \wedge C(z).$$

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 $\mathsf{rr}(\mathsf{SRNF}(q))$ 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 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 $(\mathcal{T}, \Sigma, \mathcal{A})$, $\mathbf{a}$ 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. Formally, given a TBox $\mathcal{T}$ and a set of closed predicates Σ, we say that an ABox $\mathcal{A}' \supseteq \mathcal{A}$ is a *completion of $\mathcal{A}$ w.r.t. $\mathcal{T}$ and Σ* if the following holds: (i) for each $a \in \mathsf{N_I}(\mathcal{T})$ and concept name $C \in \mathsf{N_C}(\mathcal{T})$, either $C(a) \in \mathcal{A}'$ or $\neg C(a) \in \mathcal{A}'$, (ii) for each $\{a, b\} \subseteq \mathsf{N_I}(\mathcal{T})$ and role name $r \in \mathsf{N_R}(\mathcal{T})$, either $r(a,b) \in \mathcal{A}'$ or $\neg r(a,b) \in \mathcal{A}'$, (iii) $C(a) \in \mathcal{A}'$ and $C \in \Sigma$ implies $C(a) \in A$, and (iv) $r(a,b) \in \mathcal{A}'$ 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 8. *Let $Q = (\mathcal{T}, \Sigma, q(x_1, \ldots, x_n))$ be a safe-range OMQ. A tuple of constants $\mathbf{a} = (a_1, \ldots, a_n)$ 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 Datalog$^\neg$ program $\mathcal{P}^{\mathcal{T},\Sigma}_{OMQ}$ 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 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 Datalog$^\neg$ query $(\mathcal{P}_q, p_q)$, where $\mathcal{P}_q$ is a 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, and so it has exactly one stable model [48]. Thus, we have that $\mathbf{a}$ is an answer to q over I iff $p_q(\mathbf{a})$ occurs in the stable model of $\mathcal{P}_q \cup I$. Now, given an arbitrary safe-range OMQ $Q = (\mathcal{T}, \Sigma, q(x_1, \dots, x_n))$, the FO query $q(x_1, \dots, x_n)$ might not satisfy the syntactic criterion that makes it safe-range in the traditional sense, because that additionally requires that each $x_1, \dots, x_n$ 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 $q'(x_1, \dots, x_n) = q(x_1, \dots, x_n) \wedge \bigwedge_{i=1,\dots,n} \mathsf{Adom}(x_i)$, 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 Datalog$^\neg$ and so can $q(x_1, \dots, x_n)$.

To deal with (i), consider the program $\mathcal{P}^{\mathcal{T},\Sigma}_{sat}$ from the previous section. Even though each stable model of $\mathcal{P}^{\mathcal{T},\Sigma}_{sat} \cup \hat{\mathcal{A}}$ implicitly corresponds to a consistent completion of $\mathcal{A}$ and vice versa, due to the stable model semantics, this program does not 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 $\mathcal{P}^{\mathcal{T},\Sigma}_{cmpl}$ 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 $\mathcal{P}^{\mathcal{T},\Sigma}_{cmpl}$:

$$A(x) \leftarrow \mathsf{Adom}(x), \mathit{not}\ \overline{A}(x),$$

$$\overline{B}(x) \leftarrow \mathsf{Adom}(x), \mathit{not}\ B(x),$$

$$r(x,y) \leftarrow \mathsf{Adom}(x), \mathsf{Adom}(y), \mathit{not}\ \overline{r}(x,y),$$

$$\overline{s}(x,y) \leftarrow \mathsf{Adom}(x), \mathsf{Adom}(y), \mathit{not}\ s(x,y).$$

Let $\mathcal{P}^{\mathcal{T},\Sigma}_{OMQ} = \mathcal{P}^{\mathcal{T},\Sigma}_{sat} \cup \mathcal{P}^{\mathcal{T},\Sigma}_{cmpl}$. 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 $\mathcal{P}^{\mathcal{T},\Sigma}_{OMQ} \cup \hat{\mathcal{A}}$ with $\hat{\mathcal{A}}' \subseteq I$, for any ABox $\mathcal{A}$ over the signature of $\mathcal{T}$. Note that $\mathcal{P}^{\mathcal{T},\Sigma}_{OMQ}$ is obtained from $\mathcal{T}$ and Σ in polynomial time, which leads to the following result.

Theorem 4. *Let $Q = (\mathcal{T}, \Sigma, q)$ be a safe-range OMQ. We can obtain in polynomial time a program $\mathcal{P}^{\mathcal{T},\Sigma}_{OMQ}$ from $\mathcal{T}$ and Σ such that the certain answers to Q over $\mathcal{A}$ coincide with the certain answers of $(\mathcal{P}^{\mathcal{T},\Sigma}_{OMQ} \cup \mathcal{P}_q, p_q)$ over $\hat{\mathcal{A}}$, for any ABox $\mathcal{A}$ over the signature of $\mathcal{T}$, where $(\mathcal{P}_q, p_q)$ is a Datalog$^\neg$ rewriting of q. In other words, there is a polynomial Datalog$^\neg$-rewriting of safe-range queries mediated by $\mathcal{ALCHOIQ}$ ontologies with closed predicates.*

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 5 *([49]). Answering ontology-mediated IQs is co-*NP*-complete in data complexity for $\mathcal{ALC}$ even without closed predicates.*

Theorem 6. *Answering safe-range OMQs is co-*NP*-complete in data complexity for $\mathcal{ALCHOIQ}$.*

Proof *(Proof sketch).* Checking whether a tuple of constants is a certain answer to a Datalog$^\neg$ query is co-NP-complete in terms of data complexity [41]. 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 5.

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 predicated (qfUCQs)*[12]. We can thus transfer our rewritability and complexity results to IQs and qfUCQs mediated by $\mathcal{ALCHOIQ}$ ontologies with closed predicates.

Theorem 7. *There is a polynomial rewriting of IQs and qfUCQs mediated by $\mathcal{ALCHOIQ}$ ontologies with closed predicates into Datalog$^\neg$.*

Theorem 8. *IQs and qfUCQs mediated by $\mathcal{ALCHOIQ}$ ontologies with closed predicates are co-*NP*-complete in data complexity. The knowledge base satisfiability problem in $\mathcal{ALCHOIQ}$ with closed predicates is* NP*-complete in data complexity.*

Input: a constant-free FO query $q(\mathbf{x})$ in SRNF and a set Σ of predicates
Result: a subset of free variables of q or *'fail'*
case q **of**

$R(\mathbf{x})$:
  if $R \in \Sigma$ **then**
    $\mathsf{rr}(q, \Sigma) = \mathbf{x}$
  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'*

Algorithm 1: Computation of Σ-range-restricted free variables in a FO query $q(\mathbf{x})$.

6. Discussion

In this paper, 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. 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. We then showed how to further extend this 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., [28]) 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. As a by-product of our polynomial translation, we obtained a complexity result stating that answering these queries is co-NP-complete in data complexity. We must note that our approach assumes unary encoding of integers that occur in number restrictions. 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 and we cannot infer the desired co-NP upper data complexity bound. At this moment, it is unclear how our technique could be adapted to also provide tight data complexity bounds in the case of binary encoding, and it 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 [33], 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. In our future work, we would like to investigate the absolute expressive power of OMQs mediated by $\mathcal{ALCHOIQ}$ ontologies with closed predicates, i.e., we would like to know whether the considered query languages are powerful enough to capture all database queries computable in co-NP. 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 [50]. 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 [16]. Thus, under standard complexity assumptions, there cannot exist a polynomial translation of $\mathcal{ALCHOIQ}$ mediated CQs into Datalog with stable negation. 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.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Magdalena Ortiz reports financial support was provided by Knut and Alice Wallenberg Foundation. Mantas Simkus reports financial support was provided by Knut and Alice Wallenberg Foundation.

Data availability

No data was used for the research described in the article.

Acknowledgements

Funding This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. It was also partially supported by the Austrian Science Fund (FWF) projects P30360, P30873, W1255, and by The Vienna Business Agency project 2780339.

References

[1] F. Baader, I. Horrocks, C. Lutz, U. Sattler, An Introduction to Description Logic, Cambridge University Press, 2017.
[2] I. Horrocks, Ontologies and the semantic web, Commun. ACM 51 (12) (2008) 58–67, https://doi.org/10.1145/1409360.1409377, http://doi.acm.org/10.1145/1409360.1409377.
[3] W. OWL Working Group, OWL 2 web ontology language: document overview, W3C Recommendation, available at http://www.w3.org/TR/owl2-overview/, 27 October 2009.
[4] A. Poggi, D. Lembo, D. Calvanese, G.D. Giacomo, M. Lenzerini, R. Rosati, Linking data to ontologies, J. Data Semant. 10 (2008) 133–173, https://doi.org/10.1007/978-3-540-77688-8_5.
[5] A. Borgida, On the relative expressiveness of description logics and predicate logics, Artif. Intell. 82 (1–2) (1996) 353–367.
[6] F.M. Donini, D. Nardi, R. Rosati, Description logics of minimal knowledge and negation as failure, ACM Trans. Comput. Log. 3 (2) (2002) 177–225, https://doi.org/10.1145/505372.505373.
[7] T. Eiter, G. Ianni, T. Lukasiewicz, R. Schindlauer, H. Tompits, Combining answer set programming with description logics for the semantic web, Artif. Intell. 172 (12–13) (2008) 1495, https://doi.org/10.1016/j.artint.2008.04.002.
[8] P.A. Bonatti, C. Lutz, F. Wolter, The complexity of circumscription in description logics, J. Artif. Intell. Res. 35 (2009) 717–773.
[9] K. Sengupta, A.A. Krisnadhi, P. Hitzler, Local closed world semantics: grounded circumscription for OWL, in: Proc. of ISWC 2011, Springer, 2011.
[10] E. Franconi, Y.A. Ibáñez-García, I. Seylan, Query answering with dboxes is hard, Electron. Notes Theor. Comput. Sci. 278 (2011) 71–84, https://doi.org/10.1016/j.entcs.2011.10.007.
[11] C. Lutz, I. Seylan, F. Wolter, Ontology-based data access with closed predicates is inherently intractable (sometimes), in: Proc. of IJCAI 2013, IJCAI/AAAI, 2013, pp. 1024–1030, http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6870.
[12] C. Lutz, I. Seylan, F. Wolter, The data complexity of ontology-mediated queries with closed predicates, Log. Methods Comput. Sci. 15 (3) (2019), https://doi.org/10.1145/505372.505373.
[13] S. Tobies, The complexity of reasoning with cardinality restrictions and nominals in expressive description logics, J. Artif. Intell. Res. 12 (2000) 199–217.
[14] I. Seylan, E. Franconi, J. de Bruijn, Effective query rewriting with ontologies over dboxes, in: Proc. of IJCAI 2009, 2009, pp. 923–925, http://ijcai.org/papers09/Papers/IJCAI09-157.pdf.
[15] M. Benedikt, P. Bourhis, B. ten Cate, G. Puppis, Querying visible and invisible information, in: Proc. of LICS 2016, ACM, 2016, pp. 297–306, http://doi.acm.org/10.1145/2933575.2935306.
[16] B. Glimm, Y. Kazakov, C. Lutz, Status QIO: an update, in: R. Rosati, S. Rudolph, M. Zakharyaschev (Eds.), Proc. of the 24th International Workshop on Description Logics (DL 2011), Barcelona, Spain, July 13–16, 2011, in: CEUR Workshop Proceedings, vol. 745, CEUR-WS.org, 2011, http://ceur-ws.org/Vol-745/paper_44.pdf.
[17] S. Rudolph, B. Glimm, Nominals, inverses, counting, and conjunctive queries or: why infinity is your friend!, J. Artif. Intell. Res. 39 (2010) 429–481, https://doi.org/10.1613/jair.3029.
[18] T. Gogacz, S. Lukumbuzya, M. Ortiz, M. Simkus, Datalog rewritability and data complexity of ALCHOIF with closed predicates, in: D. Calvanese, E. Erdem, M. Thielscher (Eds.), Proc. of KR 2020, 2020, pp. 434–444.
[19] S. Lukumbuzya, M. Simkus, Bounded predicates in description logics with counting, in: Z. Zhou (Ed.), Proc. of IJCAI 2021, ijcai.org, 2021, pp. 1966–1972.
[20] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, R. Rosati, Data complexity of query answering in description logics, in: Proc. of KR 2006, AAAI Press, 2006, pp. 260–270, http://www.aaai.org/Library/KR/2006/kr06-028.php.
[21] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, R. Rosati, Tractable reasoning and efficient query answering in description logics: the DL-Lite family, J. Autom. Reason. 39 (3) (2007) 385–429, https://doi.org/10.1007/s10817-007-9078-x.
[22] D. Calvanese, G.D. Giacomo, D. Lembo, M. Lenzerini, R. Rosati, Data complexity of query answering in description logics, Artif. Intell. 195 (2013) 335–360, https://doi.org/10.1016/j.artint.2012.10.003.
[23] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, R. Rosati, M. Ruzzi, D.F. Savo, The MASTRO system for ontology-based data access, Semant. Web 2 (1) (2011) 43–53, https://doi.org/10.3233/SW-2011-0029.
[24] M. Rodriguez-Muro, R. Kontchakov, M. Zakharyaschev, Ontology-based data access: ontop of databases, in: Proc. ISWC 2013, in: LNCS, vol. 8218, Springer, 2013, pp. 558–573.
[25] G. Gottlob, T. Schwentick, Rewriting ontological queries into small nonrecursive datalog programs, in: Proc. of KR 2012, AAAI Press, 2012, http://www.aaai.org/ocs/index.php/KR/KR12/paper/view/4510.
[26] G. Gottlob, S. Kikot, R. Kontchakov, V.V. Podolskii, T. Schwentick, M. Zakharyaschev, The price of query rewriting in ontology-based data access, Artif. Intell. 213 (2014) 42–59, https://doi.org/10.1016/j.artint.2014.04.004.
[27] C. Lutz, I. Seylan, F. Wolter, Ontology-mediated queries with closed predicates, in: Proc. of IJCAI 2015, AAAI Press, 2015, pp. 3120–3126, http://ijcai.org/papers15/Abstracts/IJCAI15-440.html.
[28] U. Hustadt, B. Motik, U. Sattler, Reasoning in description logics by a reduction to disjunctive datalog, J. Autom. Reason. 39 (3) (2007) 351–384, https://doi.org/10.1007/s10817-007-9080-3.
[29] T. Eiter, M. Ortiz, M. Šimkus, T. Tran, G. Xiao, Query rewriting for Horn-SHIQ plus rules, in: Proc. of AAAI 2012, AAAI Press, 2012, http://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/view/4931.
[30] M. Kaminski, Y. Nenov, B.C. Grau, Datalog rewritability of disjunctive datalog programs and non-Horn ontologies, Artif. Intell. 236 (2016) 90–118, https://doi.org/10.1016/j.artint.2016.03.006.
[31] M. Bienvenu, B. ten Cate, C. Lutz, F. Wolter, Ontology-based data access: a study through disjunctive datalog, CSP, and MMSNP, ACM Trans. Database Syst. 39 (4) (2014) 33:1–33:44, https://doi.org/10.1145/2661643, http://doi.acm.org/10.1145/2661643.
[32] M. Ortiz, S. Rudolph, M. Šimkus, Worst-case optimal reasoning for the Horn-DL fragments of OWL 1 and 2, in: Proc. of KR 2010, AAAI Press, 2010, http://aaai.org/ocs/index.php/KR/KR2010/paper/view/1296.
[33] S. Ahmetaj, M. Ortiz, M. Simkus, Polynomial rewritings from expressive description logics with closed predicates to variants of datalog, Artif. Intell. 280 (2020) 103220, https://doi.org/10.1016/j.artint.2019.103220.

[34] D. Calvanese, M. Lenzerini, D. Nardi, A unified framework for class-based representation formalisms, in: Principles of Knowledge Representation and Reasoning, Elsevier, 1994, pp. 109–120.
[35] D. Calvanese, Finite model reasoning in description logics, in: L.C. Aiello, J. Doyle, S.C. Shapiro (Eds.), Proc. of KR 1996, Morgan Kaufmann, 1996, pp. 292–303.
[36] C. Lutz, U. Sattler, L. Tendera, The complexity of finite model reasoning in description logics, Inf. Comput. 199 (1–2) (2005) 132–171, https://doi.org/10.1016/j.ic.2004.11.002.
[37] T. Gogacz, V. Gutiérrez-Basulto, Y. Ibáñez-García, F. Murlak, M. Ortiz, M. Simkus, Ontology focusing: knowledge-enriched databases on demand, in: G.D. Giacomo, A. Catalá, B. Dilkina, M. Milano, S. Barro, A. Bugarín, J. Lang (Eds.), Proc. of ECAI 2020, in: Frontiers in Artificial Intelligence and Applications, vol. 325, IOS Press, 2020, pp. 745–752.
[38] I. Pratt-Hartmann, Complexity of the two-variable fragment with counting quantifiers, J. Log. Lang. Inf. 14 (3) (2005) 369–395, https://doi.org/10.1007/s10849-005-5791-1.
[39] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming, in: Proc. of ICLP/SLP 1988, MIT Press, 1988, pp. 1070–1080.
[40] T. Eiter, G. Gottlob, H. Mannila, Disjunctive datalog, ACM Trans. Database Syst. 22 (3) (1997) 364–418.
[41] E. Dantsin, T. Eiter, G. Gottlob, A. Voronkov, Complexity and expressive power of logic programming, ACM Comput. Surv. 33 (3) (2001) 374–425.
[42] C.H. Papadimitriou, On the complexity of integer programming, J. ACM 28 (4) (1981) 765–768.
[43] B. Motik, U. Sattler, R. Studer, Query answering for OWL-DL with rules, in: S.A. McIlraith, D. Plexousakis, F. van Harmelen (Eds.), The Semantic Web – ISWC 2004, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 549–563.
[44] M. Krötzsch, F. Maier, A. Krisnadhi, P. Hitzler, A better uncle for OWL: nominal schemas for integrating rules and ontologies, in: S. Srinivasan, K. Ramamritham, A. Kumar, M.P. Ravindra, E. Bertino, R. Kumar (Eds.), Proceedings of the 20th International Conference on World Wide Web, WWW 2011, Hyderabad, India, March 28 - April 1, 2011, ACM, 2011, pp. 645–654.
[45] M. Krötzsch, S. Rudolph, Nominal schemas in description logics: complexities clarified, in: C. Baral, G.D. Giacomo, T. Eiter (Eds.), Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20–24, 2014, AAAI Press, 2014, http://www.aaai.org/ocs/index.php/KR/KR14/paper/view/8027.
[46] G. Amendola, N. Leone, M. Manna, P. Veltri, Enhancing existential rules by closed-world variables, in: J. Lang (Ed.), Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13–19, 2018, Stockholm, Sweden, ijcai.org, 2018, pp. 1676–1682.
[47] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison-Wesley, 1995, http://webdam.inria.fr/Alice/.
[48] K.R. Apt, H.A. Blair, A. Walker, Chapter 2 - towards a theory of declarative knowledge, in: J. Minker (Ed.), Foundations of Deductive Databases and Logic Programming, Morgan Kaufmann, 1988, pp. 89–148, https://www.sciencedirect.com/science/article/pii/B9780934613408500063.
[49] A. Schaerf, On the complexity of the instance checking problem in concept languages with existential quantification, J. Intell. Inf. Syst. 2 (3) (1993) 265–278, https://doi.org/10.1007/BF00962071.
[50] V. Gutiérrez-Basulto, Y. Ibáñez-García, R. Kontchakov, E.V. Kostylev, Queries with negation and inequalities over lightweight ontologies, J. Web Semant. 35 (2015) 184–202.