Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)

Compiling Model Representations for Querying Large ABoxes in Expressive DLs

Labinot Bajraktari, Magdalena Ortiz and **Mantas Šimkus**
Faculty of Informatics, TU Wien, Austria
bajraktari@kr.tuwien.ac.at, ortiz@kr.tuwien.ac.at, simkus@dbai.tuwien.ac.at

Abstract

Answering ontology mediated queries (OMQs) has received much attention in the last decade, but the big gap between practicable algorithms for lightweight ontologies, that are supported by implemented reasoners, and purely theoretical algorithms for expressive ontologies that are not amenable to implementation, has only increased. Towards narrowing the gap, we propose an algorithm to compile a representation of sets of models for $\mathcal{ALCHI}$ ontologies, which is sufficient for answering any *monotone* query. Rather than reasoning for specific ABoxes, or being fully data-independent, we use generic descriptions of families of ABoxes, given by what we call *profiles*. Our model compilation algorithm runs on TBoxes and sets of profiles, and supports the incremental addition of new profiles. To illustrate the potential of our approach for OMQ answering, we implement a rewriting into an extension of Datalog for OMQs comprising reachability queries, and provide some promising evaluation results.

1 Introduction

Answering ontology mediated queries (OMQs) has been a very active field of research over the last decade. However, there is a large gap between two main research lines. On the one hand, for the so-called *lightweight* Description Logics (DLs), algorithms have been developed, improved, and implemented in reasoners, from DL-Lite (e.g., [Calvanese *et al.*, 2005; Rosati and Almatelli, 2010; Rodriguez-Muro *et al.*, 2013]), to $\mathcal{EL}$ [Stefanoni *et al.*, 2014; Pérez-Urbina *et al.*, 2010], and other *Horn DLs* [Eiter *et al.*, 2012b; Ortiz *et al.*, 2011]. Most works use *query rewriting approaches*, where an OMQ $(\mathcal{T}, q)$ comprising of a DL ontology $\mathcal{T}$ (a.k.a. TBox) and a query q in a standard language (e.g., *conjunctive queries (CQs)*) is written into a new query q' in a target query language. Obtaining q' may be costly, but it is independent from a concrete dataset (*ABox*, in DL jargon), and then q' can be evaluated over any ABox using existing engines for the target language. On the other hand, for expressive DLs containing $\mathcal{ALC}$, most of the research on OMQ answering has had theory-oriented goals, like understanding decidability and worst-case complexity [Lutz, 2008]. Many algorithms employ tools that are not amenable to implementation, like *automata* [Calvanese *et al.*, 2008; 2014]. Rewritings have been proposed (e.g.,[Bienvenu *et al.*, 2014b; Ahmetaj *et al.*, 2016; Eiter *et al.*, 2012a]) but they appear unpracticable, and to our knowledge, they have not led to implementation attempts. A rewriting into Datalog for $\mathcal{SHIQ}$ was implemented a decade ago in the KAON2 reasoner, but only for instance queries. A published extension to CQs did not yield a data-independent rewriting, and was never implemented [Hustadt *et al.*, 2004]. State-of-the-art reasoners for expressive DLs can handle very large ontologies (e.g., Pellet [Sirin *et al.*, 2007], HermiT [Glimm *et al.*, 2014], Konclude [Steigmiller *et al.*, 2014]), but they usually aim at deciding if some model exists, and it remains unclear whether they could be adapted for OMQ answering.

To start bridging this gap, we propose an algorithm for the DL $\mathcal{ALCHI}$ that can efficiently compute a representation of a set of models for answering OMQs. A key feature of our approach is that we *compromise data independence*, and use generic descriptions of the ABoxes of interest. Our contributions can be summarized as follows:

- We propose *profile sets* as a simple yet general way to describe families of ABoxes. Profiles are combinations of concepts, role domains, and role ranges that an object may be asserted to participate in. We claim that, in many cases, only a moderate number of profiles is relevant, even when the datasets are large, or ontologies are very complex; preliminary experimental evidence backs this claim.
- We provide an algorithm that takes as an input a TBox $\mathcal{T}$ in $\mathcal{ALCHI}$, and a set of profiles $\mathbb{P}$, and effectively computes a structure $\mathbb{T}$ that represents a set of relevant models, for all knowledge bases of interest, and that can be used for answering OMQs. Specifically, for any ABox $\mathcal{A}$ that complies to the description given by $\mathbb{P}$, we can construct from $\mathbb{T}$ a set of models of $(\mathcal{T}, \mathcal{A})$ that is sufficient for answering any *monotone* query preserved under homomorphisms.
- To illustrate the potential of our algorithm for OMQ answering, we consider two kinds of queries: *instance queries* and *reachability queries*. For both of them, we provide a rewriting into ASP programs, reducing query answering to cautious entailment over the answer sets of the rewriting.
- Our algorithm supports *incremental reasoning* for ABoxes: if the model compilation has been obtained for a set of profiles, and a new family of ABoxes becomes of interest,

new profiles can be incorporated easily. The ASP rewriting can also be efficiently updated by adding new rules.

• Experiments carried out with a proof of concept implementation reveal promising results. Indeed, compiling models and rewriting into ASP is feasible even for complex ontologies. Query answering with our ASP rewritings is scalable for large ABoxes, using off-the-shelf ASP solvers.

2 Preliminaries

We briefly recall the syntax and semantics of the DL $\mathcal{ALCHI}$. We assume countably infinite sets $\mathsf{N_C}$, $\mathsf{N_R}$, and $\mathsf{N_I}$ of *concept names*, *role names*, and *individuals*. If $r \in \mathsf{N_R}$, then r and r^- are roles; the set of all roles is denoted $\overline{\mathsf{N_R}}$. For readability, r^- stands for s whenever $r = s^-$ for $s \in \mathsf{N_R}$. (Complex) *concepts* are defined as usual: (a) $\top, \bot$ and every concept name $A \in \mathsf{N_C}$ is a concept, and (b) if C, D are concepts and r is a role, then $C \sqcap D$, $C \sqcup D$, $\neg C$, $\forall r.C$, $\exists r.C$ are also concepts. A *TBox* (or, *ontology*) $\mathcal{T}$ is a finite set of *axioms* of the forms $C \sqsubseteq D$ (*concept inclusions*), where C and D are concepts, and $r \sqsubseteq s$ (*role inclusions*), where r and s are roles. An *ABox* $\mathcal{A}$ is a finite set of *assertions* of the forms $A(a)$ (called *concept assertion*) and $r(a, b)$ (called *role assertion*), where $a, b \in \mathsf{N_I}$, $A \in \mathsf{N_C}$, $r \in \mathsf{N_R}$. $\mathsf{N_I}(\mathcal{A})$ denotes the individuals occurring in $\mathcal{A}$. An *interpretation* $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ consists of a non-empty *domain* $\Delta^\mathcal{I}$ and a *valuation function* $\cdot^\mathcal{I}$ that maps each individual $a \in \mathsf{N_I}$ to an element $a^\mathcal{I} \in \Delta^\mathcal{I}$, each $A \in \mathsf{N_C}$ to a set $C^\mathcal{I} \subseteq \Delta^\mathcal{I}$, and each $r \in \mathsf{N_R}$ to a set $r^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$. The function $\cdot^\mathcal{I}$ is extended to all concepts in the usual way [Baader, 2003]. An interpretation $\mathcal{I}$ is a *model* of a TBox $\mathcal{T}$ if $C^\mathcal{I} \subseteq D^\mathcal{I}$ for each concept inclusion $C \sqsubseteq D \in \mathcal{T}$, and $r^\mathcal{I} \subseteq s^\mathcal{I}$ for each role inclusion $r \sqsubseteq s \in \mathcal{T}$. An interpretation $\mathcal{I}$ is a *model* of an ABox $\mathcal{A}$ if $a^\mathcal{I} \in C^\mathcal{I}$ for each assertion $C(a) \in \mathcal{A}$, and $\langle a^\mathcal{I}, b^\mathcal{I} \rangle \in r^\mathcal{I}$ for each assertion $r(a, b) \in \mathcal{A}$. An interpretation $\mathcal{I}$ is a *model* of $(\mathcal{T}, \mathcal{A})$, if it's a *model* of both $\mathcal{T}$ and $\mathcal{A}$. A TBox or an ABox is *consistent* (or, *satisfiable*) if it has some model. For a TBox $\mathcal{T}$, we use $\sqsubseteq^*_\mathcal{T}$ for the transitive closure of the relation $\{(r, s) \mid r \sqsubseteq s \in \mathcal{T} \text{ or } r^- \sqsubseteq s^- \in \mathcal{T}\} \cup \{(r, r) | r \in \overline{\mathsf{N_R}}\}$.

We write $\mathcal{I} \triangleright \mathcal{J}$ for two interpretations $\mathcal{I}$ and $\mathcal{J}$, if there exists a mapping h from $\Delta^\mathcal{I}$ to $\Delta^\mathcal{J}$ such that: (i) $d \in A^\mathcal{I}$ implies $h(d) \in A^\mathcal{J}$ for all $A \in \mathsf{N_C}$, and (ii) $(d, d') \in r^\mathcal{I}$ implies $(h(d), h(d')) \in r^\mathcal{J}$ for all $r \in \mathsf{N_R}$.

Normal Form We assume w.l.o.g. that TBoxes are *normalized* so that they only contain axioms of the following forms:

(NF1) $\bigsqcap A_i \sqsubseteq B$ (NF3) $A \sqsubseteq \exists r.B$ (NF5) $A \sqsubseteq \forall r.B$
(NF2) $A \sqsubseteq \bigsqcup B_i$ (NF4) $\exists r.A \sqsubseteq B$ (NF6) $r \sqsubseteq s$

where A, B, A_i, B_i are concept names in $\mathsf{N_C}$, $\top$ or $\bot$, and r, s are roles. It is well known that by means of fresh concept names any TBox $\mathcal{T}$ can be normalized into a TBox $\mathcal{T}'$ in polynomial time so that (i) the models of $\mathcal{T}'$ are models of $\mathcal{T}$, and (ii) each model of $\mathcal{T}$ can be extended to a model of $\mathcal{T}'$.

3 Compiling Models for Families of ABoxes

To describe families of ABoxes, we define *profiles*.

Definition 1 (Profiles)**.** *Concept names in* $\mathsf{N_C}$*, and concepts of the forms* $\exists r$ *and* $\exists r^-$ *with* $r \in \mathsf{N_R}$ *are called* basic concepts*. A* profile *is a set of basic concepts. Given an ABox* $\mathcal{A}$*, the* profile of a in $\mathcal{A}$ *is:*

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

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

Example 1. *Consider the set of profiles* $\mathbb{P} = \{p_1, p_2\}$ *with* $p_1 = \{A, B, \exists r\}$ *and* $p_2 = \{A, \exists r^-\}$*. Then* $\mathbb{P}$ *covers the ABox* $\mathcal{A}_1 = \{A(a), B(a), r(a, b), A(b)\}$*, but it doesn't cover* $\mathcal{A}_2 = \{A(a), r(a, b), A(b)\}$*, as* $prof^{\mathcal{A}_2}(a) = \{A, \exists r\} \notin \mathbb{P}$.

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

Definition 2. *The* relevant guesses for a profile p *are:*

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

The subsets of the guesses induce base types of p*. A* type *is a set* $\tau \subseteq \mathsf{N_C} \cup \{\bot, \top\}$*. For a type* τ*, we let*

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

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

(d1) $det^\mathcal{T}(\tau) \subseteq \tau$*,*
(d2) if $\exists r \in p$ *and* $\exists s.\top \sqsubseteq B \in \mathcal{T}$ *with* $r \sqsubseteq^*_\mathcal{T} s$*, then* $B \in \tau$*,*
(d3) if $\exists r^- \in p$ *and* $\top \sqsubseteq \forall s.B \in \mathcal{T}$ *with* $r \sqsubseteq^*_\mathcal{T} s$*, then* $B \in \tau$*.*

For $S \subseteq Guess^\mathcal{T}(p)$*, we let* $\mathsf{btyp}(p, S) = detCl^\mathcal{T}(p \cup S)$*. We define* base types *induced by profiles and sets of profiles.*

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

To understand the relevant guesses, we observe that each ABox $\mathcal{A}$, by asserting specific relations between individuals, stipulates concepts that the individuals will need to participate in to satisfy the axioms in $\mathcal{T}$, particularly the ones of the forms (NF4) and (NF5). For example, assume $\exists r.A \sqsubseteq B \in \mathcal{T}$, if in a concrete $\mathcal{A}$ we have $r(a, b) \in \mathcal{A}$, then $a \in B^\mathcal{I}$ must hold in any model $\mathcal{I}$ of $(\mathcal{A}, \mathcal{T})$ with $b \in A^\mathcal{I}$. However, this is not enforced if no relations between a and b are asserted in $\mathcal{A}$, or if b turns out not to be an instance of $A^\mathcal{I}$. To abstract away from the relations asserted in each concrete ABox, and the specific concepts that the neighbours of an object may satisfy, we take a simple approach: we consider all possible combinations (or 'guesses') of sets of concepts that may be enforced at an individual due to its neighbourhood in $\mathcal{A}$. That is, we reason explicitly about both $a \in B^\mathcal{I}$ and $a \notin B^\mathcal{I}$, but only if a participates in a relation r involved in an axiom of the form (NF4) or (NF5). As shown by our experiments (Section 5), despite being quite naive, this approach already leads to manageable type sets for many large ontologies.

Example 2. *For our running example, we use the TBox $\mathcal{T}$:*

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

For $\mathbb{P}$ from Example 1, we get these guesses and base types:

$$Guess^{\mathcal{T}}(p_1) = \{C\} \qquad \mathsf{btyp}^{\mathcal{T}}(p_1) = \{\tau_1, \tau_2\}$$
$$Guess^{\mathcal{T}}(p_2) = \{B\} \qquad \mathsf{btyp}^{\mathcal{T}}(p_2) = \{\tau_3, \tau_1\}$$

where $\tau_1 = \{A, B\}$, $\tau_2 = \{A, B, C\}$, and $\tau_3 = \{A\}$.

3.1 Type Table Compilation

The goal of this section is to provide an algorithm that, starting from the base types of $\mathbb{P}$, computes a representation of *all the relevant models* of the KBs whose ABox is covered by $\mathbb{P}$. We represent models by means of what we call *type tables*.

Definition 3. *For $\mathbf{T}$ a set of types, a type table $\mathbb{T}$ is a pair $(\mathbf{L}, \mathbf{S})$ with $\mathbf{S} \subseteq \mathbf{T} \times \left(\overline{\mathsf{N_R}} \times \mathsf{N_C}\right) \times \mathbf{T}$, $\mathbf{L} \subseteq \left(\mathbf{T} \times \mathbf{T}\right)$. We let*

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

A type table $\mathbb{T}$ covers a profile p *if $\mathbf{L}(\tau) \neq \emptyset$ for all $\tau \in \mathsf{btyp}^{\mathcal{T}}(p)$, and it* covers a set $\mathbb{P}$ of profiles *if it covers all $p \in \mathbb{P}$. The set $\mathbb{T}^G$ of* good types in $\mathbb{T}$ *contains each τ such that (i) $\bot \notin \tau$, and (ii) there is some τ_o with $\tau \in \mathbf{L}(\tau_0)$ and $\bot \notin \tau_0$.*

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

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

The algorithm for computing $\mathbb{T}$ works as follows:

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

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

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

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

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

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

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

(rule-back) If there exists some $(\tau, (r, B), \tau') \in \mathbf{S}_i$ with τ and τ' in $\mathbb{T}^G_i$, such that $bck^{\mathcal{T}}(\tau', r) \not\subseteq \tau$, we:
let $\tau_n = \tau \cup bck^{\mathcal{T}}(\tau', r)$,
replace $(\tau, (r, B), \tau')$ in $\mathbf{S}_i$ by $(\tau_n, (r, B), \tau')$,
and then, for each τ_0 such that $(\tau_0, \tau) \in \mathbf{L}_i$:
If there is some $(\hat{\tau}, (r, B), \tau')$ in $\mathbf{S}_i$ with $(\tau_0, \hat{\tau}) \in \mathbf{L}_i$ and $\hat{\tau} \neq \tau_n$, then add (τ_0, τ_n) to $\mathbf{L}_i$
Otherwise, replace (τ_0, τ) in $\mathbf{L}_i$ by (τ_0, τ_n).

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

The algorithm starts with $(\mathbf{L}_0, \mathbf{S}_0)$, and generates $(\mathbf{L}_1, \mathbf{S}_1)$, $(\mathbf{L}_2, \mathbf{S}_2)$, ... until a $\mathcal{T}$-complete $(\mathbf{L}_{fin}, \mathbf{S}_{fin})$ is reached. Intuitively, in $\mathbf{L}_i$ we keep track of 'fresh' types and how they are modified. Initially, it only contains the base types, and when the algorithm expands some type, the second component of the corresponding pair is modified. For instance, in our running example, $\mathbf{L}_0 = \{(\tau_1, \tau_1), (\tau_2, \tau_2), (\tau_3, \tau_3)\}$. When τ_2 is expanded to satisfy $C \sqsubseteq C_1 \sqcup C_2$ into $\tau_{21} = \{A, B, C, C_1\}$ and $\tau_{22} = \{A, B, C, C_2\}$ using **rule-nondet**, the pair $(\tau_2, \tau_2) \in \mathbf{L}_{i-1}$ is replaced by $\{(\tau_2, \tau_{21}), (\tau_2, \tau_{22})\} \subseteq \mathbf{L}_i$.

In $\mathbf{S}$ we store links to the types that objects of the current type can use to satisfy the existential axioms. $\mathbf{S}_0$ starts empty. When $(\tau_2, \tau_{22}) \in \mathbf{L}_i$, rule **rule-addSucc** becomes applicable due to $C_2 \sqsubseteq \exists r.C$, and a new successor for τ_{22} is created by setting $\mathbf{S}_{i+1} = \{(\tau_{22}, (r, C), \tau_4)\}$ with $\tau_4 = \{B, C\}$; the fresh type is also added to $\mathbf{L}_{i+1}$, that is, $(\tau_4, \tau_4) \in \mathbf{L}_{i+1}$. A similar rule application will create a $\tau_5 = \{B, D\}$ successor for τ_{22} and $C_2 \sqsubseteq \exists s.D$. Type τ_4 will also be expanded by **rule-nondet** into $\tau_{41} = \{B, C, C_1\}$ and $\tau_{42} = \{B, C, C_2\}$, and **rule-addSucc** will become applicable for the latter and both $C_2 \sqsubseteq \exists r.C$ and $C_2 \sqsubseteq \exists s.D$. However, the fresh successors will be the same as for τ_{22} and the entries already in the table will be reused. The final $\mathbb{T}_{fin}$ for this example contains:

$$\mathbf{L}_{fin} = \{(\tau_3, \tau_3), (\tau_1, \tau_1), (\tau_2, \tau_{21}), (\tau_2, \tau_{22}), (\tau_4, \tau_{41}), (\tau_4, \tau_{42}), (\tau_5, \tau_5)\}$$
$$\mathbf{S}_{fin} = \{(\tau_{22}, (r, C), \tau_{41}), (\tau_{22}, (r, C), \tau_{42}), (\tau_{22}, (s, D), \tau_5), (\tau_{42}, (r, C), \tau_{41}), (\tau_{42}, (r, C), \tau_{42}), (\tau_{42}, (s, D), \tau_5)\}$$

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

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

Incremental Reasoning for dynamic ABoxes. Note that if a type table covers $\mathbb{P}$, then it also covers every $\mathbb{P}' \subseteq \mathbb{P}$. Moreover, the information stored in the $\mathbf{L}$ table allows us to expand a type table if we need to cover additional profiles, while reusing as much as possible from previous computations. Let $\mathbb{T} = (\mathbf{L}, \mathbf{S})$ be a $\mathcal{T}$-completed type table, let $\mathbb{P}$ be a set of profiles, and let $\mathbb{P}'$ contain those $p \in \mathbb{P}$ that are not covered by $\mathbb{T}$. Then we can obtain a $\mathcal{T}$-completed type table $\mathbb{T}'$ that covers $\mathbb{P}$ by applying exhaustively the rules in (S2) above to $(\mathbf{L} \cup (\mathsf{btyp}^{\mathcal{T}}(\mathbb{P}') \times \mathsf{btyp}^{\mathcal{T}}(\mathbb{P}')), \mathbf{S})$.

3.2 Type Tables as Model Representations

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

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

Definition 4. *For $p \in \mathbb{P}$, the set of* good types for p in $\mathbb{T}$ *is:*

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

To capture different models of $(\mathcal{T}, \mathcal{A})$, we need to consider the different ways of assigning types from $GT_{\mathbb{T}}(\mathsf{prof}^{\mathcal{A}}(a))$ to the individuals, so that the axioms of the forms (NF4) and (NF5) in $\mathcal{T}$ are compatible with the role assertions in $\mathcal{A}$.

Definition 5 ($\mathbb{T}$-assignment)**.** *A $\mathbb{T}$-assignment for $\mathcal{A}$ is a mapping t that assigns a type from $GT_{\mathbb{T}}(\mathsf{prof}^{\mathcal{A}}(a))$ to each $a \in \mathsf{N_I}(\mathcal{A})$ so that:*

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

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

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

Example 3. *In our running example, we get:*

$$GT_{\mathbb{T}_{fin}}(p_1) = \{\tau_1, \tau_{21}, \tau_{22}\} \qquad GT_{\mathbb{T}_{fin}}(p_2) = \{\tau_1, \tau_3\}$$

and there are two possible $\mathbb{T}_{fin}$-assignments for the ABox $\mathcal{A}$:

$$t_1(a) = \tau_{21} \qquad t_2(a) = \tau_{22}$$
$$t_1(b) = \tau_1 \qquad t_2(b) = \tau_1$$

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

Definition 6. *Let t be a $\mathbb{T}$-assignment. An $(\mathcal{A}, t, \mathbb{T})$-*interpretation *$\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is defined as follows:*

- *Its domain $\Delta^{\mathcal{I}}$ is a set of sequences of the form $ar_1B_1\tau_1 \cdots r_nB_n\tau_n$ with $n \geq 0$, $a \in \mathsf{N_I}(\mathcal{A})$, and for each $0 \leq i < n$, where τ_0 denotes $t(a)$, we have $(\tau_i, (r_{i+1}, B_{i+1}), \tau_{i+1}) \in \mathbf{S}$ and τ_{i+1} is in $\mathbb{T}^G$.*
- *$\mathsf{N_I}(\mathcal{A}) \subseteq \Delta^{\mathcal{I}}$ and for each $a \in \mathsf{N_I}(\mathcal{A})$ and pair (r, B) with $\mathbf{S}(t(a), r, B) \neq \emptyset$, there is exactly one $arB\tau \in \Delta^{\mathcal{I}}$.*
- *For each $a \cdots \tau_n \in \Delta^{\mathcal{I}}$ and pair (r, B) with $\mathbf{S}(\tau_n, r, B) \neq \emptyset$, there is exactly one $a \cdots \tau_n r B \tau' \in \Delta^{\mathcal{I}}$.*
- *For each $a \in \mathsf{N_I}(\mathcal{A})$, $a^{\mathcal{I}} = a$.*
- *For each $A \in \mathsf{N_C}$, $A^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} \mid A \in tail(d)\}$, where $tail(d) = t(d)$ if $d \in \mathsf{N_I}$, and $tail(d) = \tau_n$ if $d = a \cdots \tau_n$.*
- *For all $r \in \mathsf{N_R}$, $r^{\mathcal{I}} = \{(a,b) \mid s(a,b) \in \mathcal{A}$ with $s \sqsubseteq^*_{\mathcal{T}} r\} \cup \{(b,a) \mid s(a,b) \in \mathcal{A}$ with $s \sqsubseteq^*_{\mathcal{T}} r^-\} \cup \{(d, dsB\tau) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid s \sqsubseteq^*_{\mathcal{T}} r\} \cup \{(dsB\tau, d) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid s \sqsubseteq^*_{\mathcal{T}} r^-\}$*

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

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

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

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

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

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

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

4 Answering Reachability Queries

Our model compilation is sufficient for answering OMQs subject to a reasonable restriction. We call a query q *monotone* if $\mathcal{I} \triangleright \mathcal{J}$ and $\mathcal{I} \models q$ implies $\mathcal{J} \models q$. Practically all families of queries that have been considered in the context of DLs are monotone, including CQs, regular path queries (RPQs), fragments of Datalog, etc. In fact, decidability results for non-monotone OMQs are very limited, e.g., [Gutiérrez-Basulto *et al.*, 2015].

Propositions 1 and 2 imply that the models in $mods(\mathcal{A}, \mathbb{T})$ are sufficient for answering OMQs $(\mathcal{T}, q)$ where q is monotone. However, if q goes beyond instance queries, we still need an algorithm to evaluate q over $mods(\mathcal{A}, \mathbb{T})$. For example, if q is a CQ, the algorithm in [Eiter *et al.*, 2012a] assumes a very similar representation of models and could be adapted rather easily.[1] Here we consider a simpler but nevertheless interesting family of queries, that illustrates the usefulness of our model compilation for OMQ answering:

Definition 7. *A* reachability query (RQ) *q takes the form*

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

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

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

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

Example 4. *Assume we have an ontology describing items in an inventory system. It may contain, e.g., the following:*

$$Phone5 \sqsubseteq \exists hasProcessor.AtmZ \qquad Tab2 \sqsubseteq \exists hasPart.PU25$$
$$PU25 \sqsubseteq \exists hasProcessor.AtmX \qquad AtmX \sqsubseteq AtomProcessor$$
$$Watch3 \sqsubseteq \exists hasPart.PU25 \qquad AtmZ \sqsubseteq AtomProcessor$$

[1] We did not include the adaptation here, since it needs more machinery than what can be included in this paper, and an efficient implementation would still need some care.

Algorithm 1: Retrieve types that reach C through r

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

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

as well as the role inclusion $hasProcessor \sqsubseteq hasPart$. *Assume a dataset containing instances of concepts like* $Phone5$, $Tab2$, *etc. If the prices of Atom processors increase, we may want to find all items that contain one. It would then be valuable to be able support RQs like* $(\exists y\ hasPart^*(x, y), AtomProcessor(y))$, *which can navigate the hasPart relation to different levels of depth.*

We note that in $\mathcal{ALCHI}$, RQs can be reduced to instance queries by modifying the TBox (see [Bienvenu *et al.*, 2014a]), but then the 'expensive' TBox reasoning step would depend on the query. RQs are a restricted case of the well-known RPQs (see e.g., [Bienvenu *et al.*, 2015a; Calvanese *et al.*, 2003; 2002]), and as they are monotone, they can be answered with our model compilation.

Lemma 3. *Let* $(\mathcal{T}, q)$ *be OMRQ,* $\mathbb{P}$ *be a set of profiles, and* $\mathbb{T}$ *be a* $\mathcal{T}$*-complete type table that covers* $\mathbb{P}$. *Given an ABox* $\mathcal{A}$ *covered by* $\mathbb{P}$, *and an individual* a, *we have* $(\mathcal{T}, \mathcal{A}) \not\models q(a)$ *iff there is some* $\mathcal{I} \in mods(\mathcal{A}, \mathbb{T})$ *such that* $\mathcal{I} \not\models q(a)$.

We now provide a technique to decide whether $\mathcal{I} \models q(a)$ for all models in $mods(\mathcal{A}, \mathbb{T})$. Below, for an interpretation $\mathcal{I}$ and $d \in \Delta^{\mathcal{I}}$, we let $type^{\mathcal{I}}(d) = \{A \in \mathsf{N_C} \mid d \in A^{\mathcal{I}}\}$. The core component of our technique is Algorithm 1, which runs on $\mathbb{T} = (\mathbf{L}, \mathbf{S})$. It takes as an input a role r and a concept $C = A_1 \sqcap \cdots \sqcap A_n$, and collects the set $reach_{[r,C]}$ of all the good types τ from $\mathbb{T}$ such that each d with $type^{\mathcal{I}}(d) = \tau$ r-reaches some $d' \in C^{\mathcal{I}}$, for every $\mathcal{I} \in mods(\mathcal{A}, \mathbb{T})$. For convenience, we write $C \in \tau$ to denote $\{A_1, \ldots, A_n\} \subseteq \tau$.

Proposition 3. *Let* $\tau \in \mathbb{T}^G$. *Then* $\tau \in reach_{[r,C]}$ *iff for every* $\mathcal{I} \in mods(\mathcal{A}, \mathbb{T})$ *and every* $d \in \Delta^{\mathcal{I}}$, $type^{\mathcal{I}}(d) = \tau$ *implies that* d r*-reaches in* $\mathcal{I}$ *some* $d' \in C^{\mathcal{I}}$.

Example 5. *Let* $q(x) = \exists y\ s^*(x, y), D(y)$ *be a reachability query that we would like to answer against the knowledge base in our running example. After the computation of the Algorithm 1 the* $reach_{[s,D]}$ *would contain* τ_{22}, τ_{42} *and* τ_5. τ_5 *would be contained trivially since it already contains* D, *where as types* τ_{22}, τ_{42} *will be added since each of them has an* s*-successor in the table* $\mathbf{S}_{fin}$ *that contains the concept* D.

To test $\mathcal{I} \models q(a)$ for all $\mathcal{I} \in mods(\mathcal{A}, \mathcal{T})$, we consider a set of ABoxes, which intuitively capture models of $mods_t(\mathcal{A}, \mathcal{T})$ for the different $\mathbb{T}$-assignments t, and add to them the information from $reach_{[r,C]}$, using a fresh concept name R_{rC}.

Definition 8. *Let* t *be a* $\mathbb{T}$*-assignment.* $\mathcal{A}^t$ *is the smallest* $\mathcal{A} \subseteq \mathcal{A}^t$ *such that:*
(a1) $A(a) \in \mathcal{A}^t$ *for all* $A \in t(a)$ *and all* $a \in \mathsf{N_I}(\mathcal{A})$.
(a2) $s(a, b) \in \mathcal{A}^t$ *for each* $r(a, b) \in \mathcal{A}$ *and* $r \sqsubseteq^*_{\mathcal{T}} s$.

Let $q(x) = \exists y\ r^*(x, y), C(y)$ *with* $C = A_1 \sqcap \cdots \sqcap A_n$ *be an RQ. Then* $\mathcal{A}^{t,q}$ *is the smallest* $\mathcal{A}^t \subseteq \mathcal{A}^{t,q}$ *such that:*
(q1) $R_{rC}(a) \in \mathcal{A}^{t,q}$ *if* $t(a) \in reach_{[r,C]}$.
(q2) $R_{rC}(a) \in \mathcal{A}^{t,q}$ *if* $A_i(a) \in \mathcal{A}^t$ *for all* $1 \leq i \leq n$.
(q3) $R_{rC}(a) \in \mathcal{A}^{t,q}$ *if* $R(a, b) \in \mathcal{A}^{t,q}$ *and* $R_{rC}(b) \in \mathcal{A}^{t,q}$.

Already the ABoxes $\mathcal{A}^t$ suffice for answering instance queries, and the assertions added in $\mathcal{A}^{t,q}$ correctly capture the entailment of q in all the models in $mods_t(\mathcal{A}, \mathbb{T})$.

Lemma 4. *Let* α *be an assertion and* q *an RQ. Then:*
- $\mathcal{I} \models \alpha$ *for all* $\mathcal{I} \in mods_t(\mathcal{A}, \mathbb{T})$ *iff* $\alpha \in \mathcal{A}^t$.
- $\mathcal{I} \models q(a)$ *for all* $\mathcal{I} \in mods_t(\mathcal{A}, \mathbb{T})$ *iff* $R_{rC}(a) \in \mathcal{A}^{t,q}$.

4.1 A Practicable ASP Rewriting

ASP programs. We briefly introduce ASP, for more details refer to [Brewka *et al.*, 2011]. We use an alphabet of *unary and binary predicates*, which includes $\mathsf{N_C}$ (unary) and $\mathsf{N_R}$ (binary), and an infinite countable set of variables V. A term t is a variable $x \in V$ or an individual $a \in \mathsf{N_I}$. Atoms take the form $p_u(t)$ or $p_b(t, t')$, where t, t' are terms, p_u is a unary predicate and p_b a binary one. If the terms in an atom do not contain variables, we say they are *ground*.

$$h_1 \vee \ldots \vee h_k \leftarrow b_1, \ldots, b_\ell, not\ b_{\ell+1}, \ldots, not\ b_m$$

where the $h_l, \ldots h_k$ are the *head atoms*, and $b_1, \ldots, b_m$ are the *body atoms*. Among the latter, $b_1, \ldots, b_\ell$ are *positive*, and $b_{\ell+1}, \ldots, b_m$ are *negative*. A rule with no head atoms (i.e., $k = 0$) is a *constraint*. A rule $p(\vec{a}) \leftarrow$ consisting of a single ground head is called a *fact*. A *ground program* contains only ground atoms. For a program $\mathcal{P}$, its *grounding* $ground(\mathcal{P})$ is the program obtained by replacing each rule by all its *ground instances* obtained by applying a *substitution*, that is, a mapping from variables to individuals.

The semantics of ASP programs is given by *Herbrand interpretations*, which are sets of ground atoms. Note that ABoxes are Herbrand interpretations, and a Herbrand interpretation becomes an ABox when restricted to atoms over the predicates in $\mathsf{N_C} \cup \mathsf{N_R}$. An *answer set* (a.k.a. *stable model*) of $\mathcal{P}$ is a Herbrand interpretation M that is a minimal model of the *GL-reduct* [Gelfond and Lifschitz, 1988] of $ground(\mathcal{P})$ w.r.t. M, obtained by: (i) deleting every rule ρ that contains a negative body atom $r(\vec{u})$ with $r(\vec{u}) \in M$, and (ii) deleting all negated atoms in the remaining rules.

We now provide a *rewriting of OMRQs* into logic programs that extend Datalog with disjunction and negation under the answer set semantics. In this section we assume a $\mathcal{T}$-complete type table $\mathbb{T} = (\mathbf{L}, \mathbf{S})$ that covers a given set $\mathbb{P}$ containing all the profiles of interest. We show how to obtain a program $\mathcal{P}_{\mathbb{T},q}$ such that, for an input ABox $\mathcal{A}$ represented as a program $\mathcal{P}_{\mathcal{A}}$ (see below), the answer sets of $\mathcal{P}_{\mathbb{T},q} \cup \mathcal{P}_{\mathcal{A}}$ are in close correspondence with the ABoxes $\mathcal{A}^{t,q}$, so that answering OMRQs amounts to cautious entailment. As anticipated, the rewriting does not depend on a specific ABox,

$$\bigvee_{t \in GT_{\mathbb{T}}(p)} type_\tau(x) \leftarrow prof_p(x) \quad \text{for each } p \in \mathbb{P} \quad (1)$$

$$A(x) \leftarrow type_\tau(x) \quad \text{for each } \tau \in \mathbb{T}^G \text{ and each } A \in \tau \quad (2)$$

$$s(x,y) \leftarrow r(x,y) \quad \text{for each } r \sqsubseteq^*_{\mathcal{T}} s \quad (3)$$

$$\bot \leftarrow r(x,y), A(y), notB(x) \quad \text{for each } \exists r.A \sqsubseteq B \in \mathcal{T} \quad (4)$$

$$\bot \leftarrow r(x,y), A(x), notB(y) \quad \text{for each } A \sqsubseteq \forall r.B \in \mathcal{T} \quad (5)$$

$$R_{rC}(x) \leftarrow type_\tau(x) \quad \text{for each } \tau \in reach_{[r,C]} \quad (6)$$

$$R_{rC}(x) \leftarrow A_1(x), \cdots, A_n(x) \quad \text{with } C = A_1 \sqcap \cdots \sqcap A_n(x) \quad (7)$$

$$R_{rC}(x) \leftarrow r(x,y), R_{rC}(y) \quad (8)$$

Figure 1: ASP rewriting for reachability queries in $\mathcal{ALCHI}$

but only on a $\mathbb{T}$ that covers a set $\mathbb{P}$ of profiles, and it can be used for answering q over any ABox that is covered by $\mathbb{P}$.

First we define the program $\mathcal{P}_\mathbb{T}$ comprising the rules (1) – (5) in the upper part of Figure 1. We use names in $\mathsf{N_C}$ as unary predicates and (possibly inverse) roles in $\overline{\mathsf{N_R}}$ as binary predicates. We also use a unary predicate $prof_p$ for each $p \in \mathbb{P}$ and a unary predicate $type_\tau$ for each $\tau \in \mathbb{T}^G$. Intuitively, assuming that the fact $prof_p(a)$ holds for each $a \in \mathsf{N_I}(\mathcal{A})$ with $p = \mathsf{prof}^\mathcal{A}(a)$, rule (1) guesses assignments of types to individuals. Rules (2) and (3) generate the assertions *(a1)* and *(a2)* in $\mathcal{A}^t$ for each guess, while (4) and (5) verify conditions (t1) and (t2) in Definition 5.

For a given RQ $q(x) = \exists y\ r^*(x,y), C(y)$, the rules (6) – (8) in the lower part of the figure define the program $\mathcal{P}_q$; generate the assertions *(q1)* – *(q3)* in the definition of $\mathcal{A}^{t,q}$.

Finally, we represent each given ABox $\mathcal{A}$ as a program $P_\mathcal{A}$, with facts $\alpha \leftarrow$ for all assertions $\alpha \in \mathcal{A}$, and facts $prof_p(a) \leftarrow$ for each $a \in \mathsf{N_I}(\mathcal{A})$ with $p = \mathsf{prof}^\mathcal{A}(a)$. [2]

Example 6. *Let $\mathcal{A}, \mathcal{T}, \mathbb{T}$ and $reach_{[s,D]}$ be taken from our running example. We get the following ASP rewriting (here we use τ_i and p_i rather than $type_{\tau_i}$ and $prof_{p_i}$ as predicates):*

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

$$\begin{aligned}
\mathcal{P}_\mathbb{T} = \quad & \tau_1(X) \vee \tau_{21}(X) \vee \tau_{22}(X) \leftarrow p_1(X). \\
& \tau_1(X) \vee \tau_3(X) \leftarrow p_2(X). \\
& A(X) \leftarrow \tau_3(X). \\
& A(X) \leftarrow \tau_1(X). \quad B(X) \leftarrow \tau_1(X). \\
& A(X) \leftarrow \tau_{21}(X). \quad B(X) \leftarrow \tau_{21}(X). \\
& C(X) \leftarrow \tau_{21}(X). \quad C_1(X) \leftarrow \tau_{21}(X). \\
& A(X) \leftarrow \tau_{22}(X). \quad B(X) \leftarrow \tau_{22}(X). \\
& C(X) \leftarrow \tau_{22}(X). \quad C_2(X) \leftarrow \tau_{22}(X). \\
& s(X,Y) \leftarrow r(X,Y). \\
& \bot \leftarrow r(x,y), A(y), not\ C(x). \\
& \bot \leftarrow r(x,y), C(x), not\ B(y).
\end{aligned}$$

$$\begin{aligned}
\mathcal{P}_q = \quad & R_{sD}(X) \leftarrow \tau_{22}(X). \quad R_{sD}(X) \leftarrow \tau_{42}(X). \\
& R_{sD}(X) \leftarrow \tau_5(X). \\
& R_{sD}(X) \leftarrow D(X). \\
& R_{sD}(X) \leftarrow s(X,Y), R_{sD}(Y).
\end{aligned}$$

The ASP program $\mathcal{P}_\mathbb{T} \cup \mathcal{P}_q \cup \mathcal{P}_\mathcal{A}$ has two answer sets: $\{\tau_{21}(a), \tau_1(b), A(a), B(a), C(a), C_1(a), A(b), B(b)\}$, and $\{\tau_{22}(a), \tau_1(b), A(a), B(a), C(a), C_2(a), A(b), B(b).R_{sD}(a)\}$. Note their correspondence with the $\mathbb{T}$-assignments from Example 3. $C(a)$ is found in all the answers sets, as a is an instance of C. In contrast, q from Example 5 has no certain answers: $R_{sD}(a)$ is in only one answer set, $R_{sD}(a)$ in none.

The answer sets of $\mathcal{P}_\mathbb{T} \cup \mathcal{P}_\mathcal{A}$ coincide (on the common vocabulary) with the ABoxes $\mathcal{A}^t$ for the different $\mathbb{T}$-assignments. Similarly, the answer sets of $\mathcal{P}_\mathbb{T} \cup \mathcal{P}_q \cup \mathcal{P}_\mathcal{A}$ coincide with the $\mathcal{A}^{t,q}$. From this and Lemma 4 we get:

Theorem 1. *Let $\mathcal{A}$ be an ABox covered by $\mathbb{P}$.*

- *For any assertion α, we have $(\mathcal{T}, \mathcal{A}) \models \alpha$ iff $\alpha \in M$ for all answer sets M of $\mathcal{P}_\mathbb{T} \cup \mathcal{P}_\mathcal{A}$.*
- *Given RQ $q(x) = \exists y\ r^*(x,y), C(y)$, we have $(\mathcal{T}, \mathcal{A}) \models q(a)$ iff $R_{rC}(a) \in M$ for all answer sets M of $\mathcal{P}_\mathbb{T} \cup \mathcal{P}_q \cup \mathcal{P}_\mathcal{A}$.*

5 Implementation and Experiments

We have implemented our approach in a prototype reasoner Mod4Q[3] for the DL $\mathcal{ALCH}$. Given a TBox $\mathcal{T}$ and a set $\mathbb{P}$ of profiles, the reasoner: (1) transforms $\mathcal{T}$ into normal form and drops axioms not in $\mathcal{ALCH}$; (2) computes $\mathsf{btyp}^\mathcal{T}(\mathbb{P})$; (3) runs the algorithm described in Section 3.1 to obtain a $\mathcal{T}$-complete type table $\mathbb{T}$ that covers $\mathbb{P}$, and (4) generates the ASP program $\mathcal{P}_\mathbb{T} \cup \mathcal{P}_\mathcal{A}$ as described in Section 4.1. If a set of RQs is given, it adds the rules in $\mathcal{P}_q$ for each q in the set.

The reasoner can also take a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$ as an input. In this case, it first extracts the set $\{\mathsf{prof}^\mathcal{A}(a) \mid a \in \mathsf{N_I}(\mathcal{A})\}$ of profiles occurring in $\mathcal{A}$, and then executes steps (1)–(3) above. It also generates the program $\mathcal{P}_\mathcal{A}$ for the given $\mathcal{A}$, additionally to $\mathcal{P}_\mathbb{T}$ and possibly $\mathcal{P}_q$. Moreover, the reasoner supports *incremental reasoning* over ABoxes: if after running over some set $\mathbb{P}$ of profiles, an ABox that is not covered by $\mathbb{P}$ is given, it detects which individuals have profiles not found in $\mathbb{P}$, and runs the algorithm for these new profiles only, reusing the previous type table.

Mod4Q is written in Java, and uses a PostgreSQL database to store computed type tables for later use. Experiments were run on a PC with an i7 2.4 GHz CPU running 64bit Linux-Mint 17, with a JAVA heap space of 12GB. The generated ASP programs were evaluated with Clingo [Gebser *et al.*, 2011]. In all cases, we used the profile set obtained from the ABox, and did the following sets of experiments:

Querying large ABoxes We considered some large real-world data sets, including NPD, a petroleum ontology, IMDb, a film ontology, and MyITS, a transport ontology. The first two were obtained from [Glimm *et al.*, 2017] while the latter from [Bajraktari *et al.*, 2017]. These are shown in Table 1. The time reported for producing $\mathcal{P}_\mathcal{A}$ includes the extraction of $\mathbb{P}$ from the ABox, while the time for producing $\mathcal{P}_\mathbb{T}$ includes the full computation of $\mathbb{T}$. Our experiments showed that:

- Very large real-world datasets are covered by small sets of profiles. The upper part of the table shows that the ratio of profiles to individuals found in these ABoxes is very small.
- Our approach to compiling models and rewriting into ASP is feasible even for large ABoxes. The time to compute $\mathbb{T}$ and $\mathcal{P}_\mathbb{T}$ is very short. The generated ASP programs are simple and perform well when evaluated with Clingo.

[2] We chose to use negation and constraints in the program, but transforming it into plain disjunctive Datalog is easy. Using Datalog rules to infer the facts $prof_p(a)$ from an input ABox is also easy.

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

Ontology	$\lvert\mathcal{A}\rvert$	$\lvert\mathsf{N_I}(\mathcal{A})\rvert$	$\lvert\mathbb{P}\rvert$	$\lvert\mathbb{P}\rvert/\lvert\mathsf{N_I}(\mathcal{A})\rvert$
MyITS50	125K	20K	206	1.056%
MyITS150	501K	20K	206	1.056%
MyITS250	1073K	20K	206	1.056%
NPD	856K	1510K	173	0.011%
IMDb	4736K	3765K	190	0.005%
	$\mathcal{P}_\mathcal{A}$	$\mathcal{P}_\mathbb{T}$	RQs (avg)	IQs (avg)
MyITS50	4.66s	1.21s	2.55s	1.67s
MyITS150	7.12s	1.19s	5.08s	4.53s
MyITS250	10.17s	1.29s	10.21s	8.87s
NPD	89s	2.02s	8.89s	8.47s
IMDb	1034s	2.42s	56.48	55.13s

Table 1: Querying large ABoxes

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

Table 2: Compilation of complex ontologies. CN denotes $\lvert\mathsf{N_C}(\mathcal{T})\rvert$, $\mathcal{T}'$ contains the axioms of forms (NF2,NF4,NF5) in the normalized TBox; $BT_{max}{=}max_{p\in\mathbb{P}}(\mathsf{btyp}^\mathcal{T}(p))$, and $GT{=}\lvert GT_\mathbb{T}(\mathbb{P})\rvert$.

• Query answering with our ASP rewritings scales to very large ABoxes. We evaluated two families of queries:
Instance queries: Using the program $(\mathcal{P}_\mathbb{T}\cup\mathcal{P}_\mathcal{A})$ and Clingo, we tested for the instances of all concept names in each ontology. This resulted in 370 (MyITS), 333 (NPD) and 84 (IMDb) IQs. The average answer time over all of them is reported in the table. For comparison, we ran the same instance queries using HermiT [Glimm *et al.*, 2014]; it took 4.5 hours for MyITS50, versus 10 minutes accumulated time with our prototype. For all other ontologies, HermiT either timed out after 6 hours, or crashed due to memory exhaustion.
Reachability queries: We generated all RQs paring a role name r and concept name A for which either (a) r occurs in a role assertions in an ABox, and A in a good type matching the profile of an individual in the range of r; or (b) there was an (r, B) entry in the $\mathbf{S}$ with A in its end type. Note that for all other pairs, the answers are trivially empty or coincide with the instances of concept A. This resulted in a total of 139 (MyITS), 121 (NPD), and 51 (IMDb) RQs. Answering RQs was on average slightly slower than answering IQs.

Compiling complex ontologies We took the 87 ontologies from the Oxford ontology library[4] that have both ABox assertions, and axioms of the forms NF2, NF4 and NF5. In Figure 2 we can observe that our prototype was successful on roughly 80% of the ontologies (70), while the remaining 17 cases were infeasible since we had $\geq 2^{15}$ base types for some profiles. For these 70 ontologies, on average the number of profiles per ontology was 10, while the number of base types 24, and the number of computed good types was 23. The time for compiling and producing $\mathcal{P}_\mathbb{T}$ ranged from 171 ms to 14 s, with an average of 1.3 s. We report selected results in Table 2.

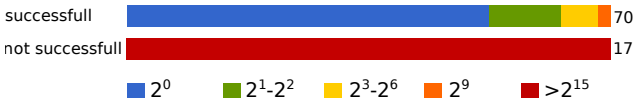

Figure 2: Ontologies for which the model compilation succeeds, categorised by maximal number of base types per profile.

Our experiments showed that:

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

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

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

6 Discussion and Conclusions

We have presented an algorithm for compiling the models of a set of relevant knowledge bases, which allows to answer OMQs over families of ABox that comply to a given description. We have proposed ASP rewritings that can be executed on standard ASP solvers to efficiently answer instance and reachability queries over large ABoxes. We plan to extend our technique to other families of OMQs, like CQs (using the ideas in [Eiter *et al.*, 2012a]), and RPQs (combining the former with the ideas in [Bienvenu *et al.*, 2015b]); and to more expressive DLs. Transitive and inverse roles can be easily incorporated. Supporting number restrictions seems feasible, albeit technical, while nominals seem more challenging.

As a compromise between data-centric and data-independent reasoning, our techniques work for families of ABoxes, represented by sets of *profiles*. Despite their simplicity, profiles seem useful, and even large ABoxes seem to use few profiles. This may sometimes be partially explained by automated processes that produce the data (mappings, forms, scripts, etc), which naturally restrict its shape. Investigating this is an interesting path for future research. Currently, the main bottleneck of our algorithm is the computation of 'base types' from profiles, which expands the profiles with sets of guesses. Although the number of guesses was usually small for the considered ontologies, it became unmanageable in roughly 20% of the cases. Investigating more refined alternatives than this naive guessing seems crucial.

Acknowledgments

This work was supported by the Austrian Science Fund (FWF) via projects P30360, P30873, and W1255.

[4] http://www.cs.ox.ac.uk/isg/ontologies/

References

[Ahmetaj *et al.*, 2016] Shqiponja Ahmetaj, Magdalena Ortiz, and Mantas Simkus. Polynomial datalog rewritings for expressive description logics with closed predicates. In *IJCAI 2016, USA*, 2016.

[Baader, 2003] Franz Baader. *The description logic handbook: theory, implementation, and applications*. Cambridge university press, 2003.

[Bajraktari *et al.*, 2017] Labinot Bajraktari, Magdalena Ortiz, and Mantas Simkus. Clopen knowledge bases: Combining description logics and answer set programming. In *Proceedings of DL2017, Montpellier, France*, 2017.

[Bienvenu *et al.*, 2014a] Meghyn Bienvenu, Diego Calvanese, Magdalena Ortiz, and Mantas Šimkus. Nested regular path queries in description logics. AAAI Press, 2014.

[Bienvenu *et al.*, 2014b] Meghyn Bienvenu, Balder ten Cate, Carsten Lutz, and Frank Wolter. Ontology-based data access: A study through disjunctive datalog, CSP, and MMSNP. *ACM Trans. Database Syst.*, 39(4):33:1–33:44, 2014.

[Bienvenu *et al.*, 2015a] Meghyn Bienvenu, Magdalena Ortiz, and Mantas Simkus. Regular path queries in lightweight description logics: Complexity and algorithms. *J. Artif. Intell. Res.*, 53:315–374, 2015.

[Bienvenu *et al.*, 2015b] Meghyn Bienvenu, Magdalena Ortiz, and Mantas Šimkus. Regular path queries in lightweight description logics: Complexity and algorithms. *J. Artif. Intell. Res. (JAIR)*, 53:315–374, 2015.

[Brewka *et al.*, 2011] Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski. Answer set programming at a glance. *Commun. ACM*, 54(12):92–103, 2011.

[Calvanese *et al.*, 2002] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Rewriting of regular expressions and regular path queries. *J. Comput. Syst. Sci.*, 64(3):443–465, 2002.

[Calvanese *et al.*, 2003] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Reasoning on regular path queries. *SIGMOD Record*, 32(4):83–92, 2003.

[Calvanese *et al.*, 2005] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Dl-lite: Tractable description logics for ontologies. In *In AAAI/IAAI 2005, USA*, 2005.

[Calvanese *et al.*, 2008] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Conjunctive query containment and answering under description logic constraints. *ACM Trans. Comput. Log.*, 9(3), 2008.

[Calvanese *et al.*, 2014] Diego Calvanese, Thomas Eiter, and Magdalena Ortiz. Answering regular path queries in expressive description logics via alternating tree-automata. *Inf. Comput.*, 237:12–55, 2014.

[Eiter *et al.*, 2012a] Thomas Eiter, Magdalena Ortiz, and Mantas Šimkus. Conjunctive query answering in the description logic SH using knots. *J. Comput. Syst. Sci.*, (1), 2012.

[Eiter *et al.*, 2012b] Thomas Eiter, Magdalena Ortiz, Mantas Šimkus, Trung-Kien Tran, and Guohui Xiao. Query rewriting for Horn-SHIQ plus rules. AAAI Press, 2012.

[Gebser *et al.*, 2011] Martin Gebser, Benjamin Kaufmann, Roland Kaminski, Max Ostrowski, Torsten Schaub, and Marius Thomas Schneider. Potassco: The potsdam answer set solving collection. *AI Commun.*, 24(2):107–124, 2011.

[Gelfond and Lifschitz, 1988] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Proc. of ICLP/SLP 1988*, pages 1070–1080. MIT Press, 1988.

[Glimm *et al.*, 2014] Birte Glimm, Ian Horrocks, Boris Motik, Giorgos Stoilos, and Zhe Wang. Hermit: An OWL 2 reasoner. *J. Autom. Reasoning*, 53(3):245–269, 2014.

[Glimm *et al.*, 2017] Birte Glimm, Yevgeny Kazakov, and Trung-Kien Tran. Ontology materialization by abstraction refinement in horn SHOIF. In *In AAAI 2017, USA*, 2017.

[Gutiérrez-Basulto *et al.*, 2015] Víctor Gutiérrez-Basulto, Yazmin Angélica Ibáñez-García, Roman Kontchakov, and Egor V. Kostylev. Queries with negation and inequalities over lightweight ontologies. *J. Web Sem.*, 35:184–202, 2015.

[Hustadt *et al.*, 2004] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. A decomposition rule for decision procedures by resolution-based calculi. In *LPAR 2004, Uruguay*, 2004.

[Lutz, 2008] Carsten Lutz. The complexity of conjunctive query answering in expressive description logics. volume 5195 of *LNCS*, pages 179–193. Springer, 2008.

[Ortiz *et al.*, 2011] Magdalena Ortiz, Sebastian Rudolph, and Mantas Simkus. Query answering in the Horn fragments of the description logics SHOIQ and SROIQ. In *IJCAI 2011, Spain*, 2011.

[Pérez-Urbina *et al.*, 2010] Héctor Pérez-Urbina, Boris Motik, and Ian Horrocks. Tractable query answering and rewriting under description logic constraints. *J. Applied Logic*, 8(2):186–209, 2010.

[Rodriguez-Muro *et al.*, 2013] Mariano Rodriguez-Muro, Roman Kontchakov, and Michael Zakharyaschev. Ontology-based data access: Ontop of databases. volume 8218 of *LNCS*, pages 558–573. Springer, 2013.

[Rosati and Almatelli, 2010] Riccardo Rosati and Alessandro Almatelli. Improving query answering over dl-lite ontologies. In *KR 2010, Canada*. AAAI Press, 2010.

[Sirin *et al.*, 2007] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL reasoner. *J. Web Sem.*, 5(2):51–53, 2007.

[Stefanoni *et al.*, 2014] Giorgio Stefanoni, Boris Motik, Markus Krötzsch, and Sebastian Rudolph. The complexity of answering conjunctive and navigational queries over OWL 2 EL knowledge bases. *J. Artif. Intell. Res. (JAIR)*, 51:645–705, 2014.

[Steigmiller *et al.*, 2014] Andreas Steigmiller, Thorsten Liebig, and Birte Glimm. Konclude: System description. *J. Web Sem.*, 27:78–85, 2014.