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 ACCHI 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 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 $(T, q)$ comprising of a DL ontology $T$ (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 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 ACCHI 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 $T$ in ACCHI, and a set of profiles $\mathcal{P}$, and effectively computes a structure $\mathcal{T}$ 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 $A$ that complies to the description given by $\mathcal{P}$, we can construct from $\mathcal{T}$ a set of models of $(T, A)$ 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 imple-
mentation reveal promising results. Indeed, compiling mod-
els and rewriting into ASP is feasible even for complex on-
tologies. Query answering with our ASP rewritings is scal-
able for large ABoxes, using off-the-shelf ASP solvers.

2 Preliminaries

We briefly recall the syntax and semantics of the DL
AC{CLASS}. We assume countably infinite sets NC, NR, and NL of
concept names, role names, and individuals. If r ∈ NR, then r and r− are roles; the set of all roles is \( \mathbb{N}_R \).
For readability, r− stands for s whenever r = s− for s ∈ NR.

(Complex) concepts are defined as usual: (a) \( T, \sqcup \), \( \sqcap \) and every name \( A \in \mathcal{N}_C \) is a concept, and (b) if C, D are con-
cepts 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) \( T \) is a finite set of ax-
ioms of the forms \( C \subseteq D \) (concept inclusions), where C and D are con-
cepts, and \( r \sqsubseteq s \) (role inclusions), where r and s are roles. An ABox \( A \) is a finite set of assertions of the forms
\( (A) \) called concept assertion) and \( r(a, b) \) (called role assertion),
where \( a, b \in \mathcal{N}_I \). Each \( r(a, b) \in A \) denotes the individuals occurring in \( A \). An interpretation \( I = (\Delta^I, \tau^I) \) consists of a non-empty domain \( \Delta^I \) and a valuation function \( \tau^I \) that maps each individual \( a \in \mathcal{N}_I \) to an element \( a^I \in \Delta^I \), each \( A \in \mathcal{N}_C \) to a set \( C^A \subseteq \Delta^I \), and each r ∈ NR to a set \( r^I \subseteq \Delta^I \times \Delta^I \). The function \( \tau^I \) is extended to all con-
cepts in the usual way [Baader, 2003]. An interpretation \( I \) is a model of a TBox \( T \) if \( C^A \subseteq \Delta^I \) for each concept in-
clusion \( C \subseteq D \in \mathcal{T} \), and \( r^I \subseteq s^I \) for each role in-
clusion \( r \sqsubseteq s \in \mathcal{T} \). An interpretation \( I \) is a model of an ABox \( A \) if \( a^I \in C^A \) for each assertion \( C(a) \in A \), and \( r^I \subseteq s^I \) for each assertion \( r(a, b) \in A \). An interpretation \( I \) is a model of \( (T, A) \), if it’s a model of both \( T \) and \( A \). A TBox or an ABox is consistent (or, satisfiable) if it has some model.

For a TBox \( T \), we use \( C^I \) for the transitive closure of the relation \( \{ (a, b) \mid a^I \in C \in \mathcal{T} \} \). We write \( \sqsubseteq_I \) for two interpretations \( I, J \), if there exists a mapping \( h \) from \( \Delta^I \) to \( \Delta^J \) such that:

- \( d \in A^I \) implies \( h(d) \in A^J \) for all \( A \in \mathcal{N}_C \), and
- \( (d, d') \in r^I \) implies \( (h(d), h(d')) \in r^J \) for all \( r \in \mathcal{N}_R \).

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

- \( (A) \), \( A \subseteq C \), \( A \sqcap C \), \( A \sqcup C \), \( \neg A \), \( \forall r.A \), \( \exists r.A \), \( r \sqsubseteq s \), \( \exists r \). A concept name \( A \) is a \( \Sigma \) concept if \( A \in \mathcal{N}_C \) or \( A \in \mathcal{N}_R \).

A profile of a concept \( A \) is a set of concept names. Given an ABox \( A \),
the profile of a \( a \) in \( A \) is:

\[
\text{prof}^A(a) = \{ A \mid A \in \mathcal{N}_C, A(a) \in A \} \cup \{ \exists r \mid r \in \mathcal{N}_R, r(a, b) \in A \} \\
\cup \{ \exists r^- \mid r \in \mathcal{N}_R, r(a, b) \in A \}
\]

A set \( \mathbb{P} \) of profiles covers \( \mathcal{A} \) if \( \text{prof}^A(a) \in \mathbb{P} \) for all \( a \in \mathcal{N}_I(\mathcal{A}) \).

Example 1. Consider the set of profiles \( \mathbb{P} = \{ p_1, p_2 \} \) with
\( p_1 = \{ A, B, r \} \) and \( p_2 = \{ A, \exists r \} \). Then \( \mathbb{P} \) covers
the ABox \( A_1 = \{ (A, a), (A, b), (B, a, b) \} \), but it doesn’t cover
\( A_2 = \{ (A, a), (r(a, b), A) \} \), as \( \text{prof}^{A_2}(a) = \{ A, \exists r \} \notin \mathbb{P} \).

In the rest of this section, we assume we are given a fixed
TBox \( T \), and a set \( \mathbb{P} \) of profiles that covers all the ABoxes of interest.
We then expand the profiles \( \mathbb{P} \in \mathbb{P} \) with concept names \( A \) such that, for some \( A \) covered by \( \mathbb{P} \), and some \( a \) with \( \text{prof}^A(a) = p \), it may be the case that \( a \in A^T \) holds in the models \( I \) of \( (T, A) \). 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 \( T \). This gives us base types, that in the next stage are further expanded to satisfy the axioms in \( T \), or eliminated if we infer that they cannot occur in models.

Definition 2. The relevant guesses for a profile \( p \) are:

- \( \text{Guess}^T(p) = \{ B \mid \exists r \in p, B \sqsubseteq r^I, B \notin p \} \cup \{ B \mid \exists r^- \in p, A \sqsubseteq r^I, B \in r^I, B \notin p \} \)

The subsets of the guesses induce base types of \( p \). A type \( T \) is a set \( T \subseteq \mathcal{N}_C \cup \{ \bot, \top \} \). For a type \( T \), we let

\[
\text{detCl}^T(p) = \{ B \mid \{ A_1, \ldots, A_n \} \subseteq T \} \\
\text{detCl}^T(p) = \{ B \mid \exists r \in p, B \sqsubseteq r^I, B \notin p \} \cup \{ B \mid \exists r^- \in p, A \sqsubseteq r^I, B \in r^I, B \notin p \} \)

To understand the relevant guesses, we observe that each
ABox \( A \), by asserting specific relations between individuals,
stipulates concepts that the individuals will need to partici-
bate in to satisfy the axioms in \( T \), particularly the ones of the
forms (NF4) and (NF5). For example, assume \( \exists r.A \sqsubseteq B \) is a
concept in a concrete ABox \( A \), then \( a \in B^T \) must hold in any model \( I \) of \( (T, A) \) with \( b \in A^T \). However, this is not enforced if no relations between a and b are asserted in
(\( A \), T), or if b turns out not to be an instance of A^T. 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 \( A \). That is, we reason explicitly about both \( a \in B^T \) and \( a \notin B^T \), but only if a participates in a relation r involved in an axiom of
(\( A \), T) or \( N \). 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 $T$:
\[\begin{align*}
&\exists s. A \sqsubseteq C \\
&\forall s. B \\
&C \sqsubseteq C_1 \cup C_2 \\
&C \sqsubseteq \exists s. D \\
&C \sqsubseteq \exists r. C \\
&r \sqsubseteq s
\end{align*}\]

For $\mathbb{P}$ from Example 1, we get these guesses and base types:
\[\begin{align*}
\text{Guess}^T(p_1) &= \{C\} \\
\text{btyp}^T(p_1) &= \{\tau_1, \tau_2\}
\end{align*}\]
\[\begin{align*}
\text{Guess}^T(p_2) &= \{B\} \\
\text{btyp}^T(p_2) &= \{\tau_3, \tau_1\}
\end{align*}\]

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 $T$ a set of types, a type table $T$ is a pair $(L, S)$ with $S \subseteq T \times (\mathbb{N}_R \times \mathbb{N}_C) \times T, L \subseteq (T \times T)$. We let $L(\tau) = \{\tau' \mid (\tau, \tau') \in L\}$, and $S(\tau, r, B) = \{\tau' \mid (\tau, (r, B), \tau') \in S\}$.

A type table $T$ covers a profile $p$ if $L(\tau) \neq \emptyset$ for all $\tau \in \text{btyp}^T(p)$, and it covers a set $\mathbb{P}$ of profiles if it covers all $p \in \mathbb{P}$. The set $\mathbb{T}^G_i$ of good types in $T$ contains each $\tau$ such that (i) $\perp \notin \tau$, and (ii) there is some $\tau_0$ with $\tau \in L(\tau_0)$ and $\perp \not\in \tau_0$.

We let, for each $\tau \in T$, and each $r \in \mathbb{N}_R$,
\[\begin{align*}
\text{fwd}^T(\tau, r) &= \{B \mid A \sqsubseteq \exists r. B \in T, A \in \tau, \text{ and } r \sqsubseteq s\}, \\
\text{bck}^T(\tau, r) &= \{B \mid \exists s. A \sqsubseteq B \in T, A \in \tau, \text{ and } r \sqsubseteq s\}
\end{align*}\]

The algorithm for computing $T$ works as follows:

(S1) Initialize $L_0 = \text{btyp}^T(\emptyset) \times \text{btyp}^T(\emptyset), S_0 = \emptyset$.

(S2) We obtain $(L_{i+1}, S_{i+1})$ from $(L_i, S_i)$ by applying one of the following rules:

(rule-mark) If there exists $\tau, r, B$ with $S_i(\tau, r, B) \neq \emptyset$ such that $\tau \in T^G_i$ and $S_i(\tau, r, B) \cap T^G_i \neq \emptyset$, then replace each $(\tau_0, \tau) \in L_i$ by $(\tau_0, \tau \cup \{B\})$, and each $(\tau, (r, B), \tau') \in S_i$ by $(\tau \cup \{\perp\}, (r, B), \tau')$.

(rule-det) If there exists some $(\tau_0, \tau) \in L_i$ with $\text{det}^T(\tau) \subseteq \tau$ and $\tau \in T^G_i$, then replace each $(\tau_0, \tau) \in L_i$ by $(\tau_0, \tau \cup \text{det}^T(\tau))$, each $(\tau, (r, B), \tau') \in S_i$ by $(\tau \cup \text{det}^T(\tau), (r, B), \tau')$, each $(\tau', (r, B), \tau) \in S_i$ by $(\tau', (r, B), \tau \cup \text{det}^T(\tau))$.

(rule-nondet) If there exists some $(\tau_0, \tau) \in L_i$ with $\tau \in T^G_i$ and an axiom $A \sqsubseteq B_1 \sqsubseteq \cdots \sqsubseteq B_n$ of the form (NF2) in $T$ such that $A \in \tau$ and $\{B_1, \ldots, B_n\} \cap T^G_i = \emptyset$, then replace, for $1 \leq i \leq n$:
\[\begin{align*}
&\text{each } (\tau_0, \tau) \in L_i \\
&\text{each } (\tau, (r, B), \tau') \in S_i \\
&\text{each } (\tau', (r, B), \tau) \in S_i
\end{align*}\]

(rule-addSucc) If there exists some $(\tau_0, \tau) \in L_i$ with $\tau \in T^G_i$, and some $A \sqsubseteq \exists r. B \in T$ such that $A \in \tau$ and $S_i(\tau, (r, B), \emptyset) = \emptyset$, then let $\tau_n = \{B \cup \text{fwd}^T(\tau, r)\}$ and:

If $L_i(\tau_n) = \emptyset$, then add $(\tau_n, \tau_n)$ to $L_i$, and add $(\tau, (r, B), \tau_n)$ to $S_i$.

If $L_i(\tau_n) \neq \emptyset$, then add $(\tau, (r, B), \tau)$ to $S_i$ for each $\tau \in L_i(\tau_n)$.

(rule-back) If there exists some $(\tau, (r, B), \tau') \in S_i$ with $\tau$ and $\tau'$ in $T^G_i$ and $\text{fwd}^T(\tau, r) \subseteq \tau'$, then we:
\[\begin{align*}
&\text{let } \tau_n = \tau' \cup \text{fwd}^T(\tau, r), \text{ and:} \\
&\text{if } L_i(\tau_n) = \emptyset \text{ then add } (\tau_n, \tau_n) \text{ to } L_i, \text{ and replace } (\tau, (r, B), \tau') \text{ in } S_i \text{ by } (\tau, (r, B), \tau_n). \\
&\text{Otherwise, replace } (\tau, (r, B), \tau_n) \text{ in } S_i \text{ by } (\tau, (r, B), \tau') \text{ for each } \tau' \in L_i(\tau_n).
\end{align*}\]

(S3) If no rules are applicable, the type table $(L_i, S_i)$ is called $T$-complete and the algorithm terminates.

The algorithm starts with $(L_0, S_0)$, and generates $(L_1, S_1), (L_2, S_2), \ldots$ until a $T$-complete $(L_{\text{fin}}, S_{\text{fin}})$ is reached. Intuitively, in $L_i$ 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, $L_0 = \{(\tau_1, \tau_1), (\tau_2, \tau_2), (\tau_3, \tau_3)\}$.

When $\tau_2$ is expanded to satisfy $C \sqsubseteq C_1 \sqsubseteq C_2$ into $\tau_2 = \{A, B, C, C_1\}$ and $\tau_2 = \{A, B, C, C_2\}$ using rule-nondet, the pair $(\tau_2, \tau_2)$ is replaced by $(\tau_2, (\tau_2, 2), (\tau_2, 2)) \subseteq L_i$.

In $S$ we store links to the types that objects of the current type can use to satisfy the existential axioms. $S_0$ starts empty. When $(\tau_2, \tau_2) \in L_i$, rule-addSucc becomes applicable due to $C_2 \sqsubseteq \exists r. C$, and a new successor for $\tau_2$ is created by setting $S_{i+1} = \{(\tau_2, (r, C), \tau_4)\}$ with $\tau_4 = \{B, C\}$. The fresh type is also added to $L_{i+1}$, that is, $(\tau_2, \tau_2) \in L_{i+1}$. A similar rule application will create a $\tau_3 = \{B, D\}$ successor for $\tau_3$ and $C_2 \sqsubseteq \exists s. D$. Type $\tau_3$ will also be expanded by rule-nondet into $\tau_4 = \{B, C, C_1\}$ and $\tau_4 = \{B, C, C_2\}$, 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 $\tau_2$, and the entries already in the table will be reused. The final $T_{\text{fin}}$ for this example contains:
\[\begin{align*}
L_{\text{fin}} &= \{(\tau_1, \tau_3), (\tau_1, \tau_1), (\tau_2, \tau_2), (\tau_2, \tau_2), \\
&\quad (\tau_4, \tau_4), (\tau_4, \tau_4), (\tau_5, \tau_5)\}, \\
S_{\text{fin}} &= \{\tau_2, (r, C), \tau_4), (\tau_2, (r, C), \tau_4), (\tau_2, (r, D), \tau_5), \\
&\quad \tau_4, (r, C), \tau_4), (\tau_4, (r, C), \tau_4), (\tau_4, (r, D), \tau_5)\}
\end{align*}\]

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

Lemma 1. The number of different tables $(L_i, S_i)$ that can be produced by the rule applications, and the number of rule applications required to reach a $T$-complete $(L_{\text{fin}}, S_{\text{fin}})$, are bounded by an exponential in $T$, and by a polynomial in $\mathbb{P}$. 
Incremental Reasoning for dynamic ABoxes. Note that if a type table covers $P$, then it also covers every $P' \subseteq P$. Moreover, the information stored in the $L$ 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 $\mathcal{T} = (L, S)$ be a $\mathcal{T}$-completed type table, let $P$ be a set of profiles, and let $P'$ contain those $p \in P$ that are not covered by $\mathcal{T}$. Then we can obtain a $\mathcal{T}$-completed type table $\mathcal{T}'$ that covers $P$ by applying exhaustively the rules in (S2) above to $(L \cup (btyp^T(P') \times btyp^T(P')), S)$.

3.2 Type Tables as Model Representations

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

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

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

$$GT_\mathcal{T}(p) = \{ \tau \in TG | (p, \tau) \in L \text{ for some } \tau_0 \in btyp^T(p) \}$$

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

**Definition 5 ($\mathcal{T}$-assignment).** A $\mathcal{T}$-assignment for $A$ is a mapping $t$ that assigns a type from $GT_\mathcal{T}(\text{prof}^A(a))$ to each $a \in N_0(A)$ so that:

1. $(t1)$ for each $r(a, b) \in A$ with $A \in t(b)$, if $\exists b \in B \in \mathcal{T}$ for some $r \in TG$ such that $b \in t(a)$, and
2. $(t2)$ for each $r(a, b) \in A$ with $A \in t(b)$, if $A \subseteq \forall s.B \in \mathcal{T}$ for some $r \in TG$ such that $b \in t(a)$.

Note that $GT_\mathcal{T}(\text{prof}^A(a)) \neq \emptyset$ for each $a \in N_0(A)$ is necessary (but not sufficient) for the existence of $\mathcal{T}$-assignments.

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

$$GT_{\mathcal{T}_{\text{fin}}}(p_1) = \{ \tau_1, \tau_{21}, \tau_{22} \}$$

$$GT_{\mathcal{T}_{\text{fin}}}(p_2) = \{ \tau_1, \tau_3 \}$$

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

$$t_1(a) = \tau_{21} \quad t_2(a) = \tau_{22}$$

$$t_1(b) = \tau_1 \quad t_2(b) = \tau_1$$

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

**Definition 6.** Let $t$ be a $\mathcal{T}$-assignment. An $(\mathcal{A}, t, \mathcal{T})$-interpretation $\mathcal{I} = ((\Delta^2, \cdot^2), \mathcal{T})$ is defined as follows:

- Its domain $\Delta^2$ is a set of sequences of the form $a_1, B_1, \tau_1 \cdots a_n, B_n, \tau_n$ with $n \geq 0$, $a \in N_0(A)$, and for each $0 \leq i < n$, where $\tau_0$ denotes $t(a)$, we have $(\tau_0, (\tau_{i+1}, B_{i+1}), \tau_{i+1}) \in S$ and $\tau_{i+1} \in TG$.
- $N_0(A) \subseteq \Delta^2$ and for each $a \in N_0(A)$ and pair $(r, B)$ with $S(t(a), r, B) \neq \emptyset$, there is exactly one $a \cdot B^r \in \Delta^2$.
- For each $a \cdot \cdots \cdot a \in \Delta^2$ and pair $(r, B)$, with $S(\tau_0, \cdots, \tau_n, \cdots, B \cdot), B \neq \emptyset$, there is exactly one $a \cdot \cdots \cdot a \cdot \tau_0 \cdot B \cdot \tau^2 \in \Delta^2$.
- For each $a \in N_0(A)$, $a \cdot \tau = a$.
- For each $A \in N_0(A)$, $\mathcal{I} = \{ d \in \Delta^2 \mid A \in \text{tail}(d) \}$, where $\text{tail}(d) = t(d)$ if $d \in N_1$, and $\text{tail}(d) = \tau_0$ if $d = a \cdot \cdots \cdot a$.
- For all $r \in N_r$, $\mathcal{I} = \{ (a, b) \mid s(a, b) \in A \text{ with } s \subseteq r \} \cup \{ (b, a) \mid s(b, a) \in A \text{ with } s \subseteq r \}$.

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

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

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

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

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

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

**Lemma 2.** $(\mathcal{T}, A)$ is satisfiable iff $\mathcal{A}$ has a $\mathcal{T}$-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 $\mathcal{I} \models \mathcal{J}$ 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 $mods(A, \mathcal{T})$ 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 $mods(A, \mathcal{T})$. 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. 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

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

where $r$ is a (possibly inverse) role, and $C$ is of the form $A_1 \cap \cdots \cap A_n$ with $n \geq 1$ and $A_i \in N_C$ 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 $e_1, e_2 \in \Delta^2$, and let $r$ be a role. We say that $e_1$ $r$-reaches $e_2$ if there is a sequence $d_1, \cdots, d_n$ of objects from $\Delta^2$ such that $d_1 = e_1$, $d_n = e_2$, $n \geq 1$, and for each $1 \leq i < n$, $(d_i, d_{i+1}) \in r^2$.

We write $\mathcal{I} = q(a)$, if $a^2$ $r$-reaches some $d \in \Delta^2$ with $d \in C^2$. We call $a \in N_0$ an answer to $(\mathcal{T}, q)$ over $A$, and write $(\mathcal{T}, A) = q(a)$ if $\mathcal{I} = q(a)$ for all $\mathcal{I} = (\mathcal{T}, A)$.

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

```
Phone5 ⊏ hasProcessor AtmZ Tab2 ⊏ hasPart.PhuP25
PU25 ⊏ hasProcessor AtmX AtmX ⊏ AtomProcessor
Watch3 ⊏ hasPart.PhuP25 AtmZ ⊏ AtomProcessor
```

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

Let $reach_{[r; C]} := \{ \tau \mid \tau \in T^G \text{ and } C \in \tau \}$
Let $reach_{[r; C]} := reach_{[r; C]}$
repeat
  forall $\tau \in T^G$ and $s, B$ with $s \sqsubseteq r$ such that
  $S(\tau, s, B) \cap T^\tau \neq \emptyset$ do
    if $\tau' \in reach_{[r; C]}$
    for each $(\tau, (s, B), \tau') \in S$ with $\tau' \in T^G$ 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 $\text{hasProcessor} \sqsubseteq \text{hasPart}$. 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 ($\exists y \text{hasPart}^*(x, y), \text{AtomProcessor}(y)$), which can navigate the hasPart relation to different levels of depth.

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

We now provide a technique to decide whether $\mathcal{I} \models q(a)$ for all models in $\text{mods}(A, \mathcal{T})$. Below, for an interpretation $\mathcal{I}$ and $d \in \Delta^2$, we let $\text{type}^2(d) = \{ \lambda \in \mathbb{N}_C \mid d \in \Delta^2 \}$. The core component of our technique is Algorithm 1, which runs on $\mathcal{T} = (L; S)$. It takes as input a role $r$ and a concept $C = A_1 \sqcap \cdots \sqcap A_n$, and collects the set $\text{reach}_{[r; C]}$ of all the good types $\tau$ from $T$ such that each $d$ with $\text{type}^2(d) = \tau$ $r$-reaches some $d' \in C^2$, for every $\mathcal{I} \in \text{mods}(A, \mathcal{T})$. For convenience, we write $\mathcal{C} \in \tau$ to denote $\{ A_1, \ldots, A_n \} \subseteq \tau$.

Proposition 3. Let $\tau \in T^G$. Then $\tau \in \text{reach}_{[r; C]}$ iff for every $\mathcal{I} \in \text{mods}(A, \mathcal{T})$ and every $d \in \Delta^2$, $\text{type}^2(d) = \tau$ implies that $d$ $r$-reaches in $\mathcal{I}$ some $d' \in C^2$.

Example 5. Let $q(x) = \exists y \, x^s(y, x)$, $D(y)$ 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 $\text{reach}_{[r; D]}$ would contain $\tau_{22}$, $\tau_{12}$ and $\tau_5$, $\tau_5$ would be contained trivially since it already contains $D$, whereas types $\tau_{22}$, $\tau_{12}$ will be added since each of them has an $s$-successor in the table $S_{\text{rel}}$ that contains the concept $D$.

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

Definition 8. Let $t$ be a $\mathcal{T}$-assignment. $A^t$ is the smallest $A \subseteq A^t$ such that:

(a1) $A(a) \in A^t$ for all $A \in t(a)$ and all $a \in N_1(A)$.

(a2) $s(a, b) \in A^t$ for all $s(a, b) \in A$ and $a \in t(S)$.

Let $q(x) = \exists y \, r^*(x, y), C(y)$ with $C = A_1 \sqcap \cdots \sqcap A_n$ be an RQ. Then $A^t$ is the smallest $A^t \subseteq A^t$ such that:

(q1) $R_{rC}(a) \in A^t$ if $t(a) \in \text{reach}_{[r; C]}$.

(q2) $R_{rC}(a) \in A^t$ if $A(a) \in A^t$ for all $1 \leq i \leq n$.

(q3) $R_{rC}(a) \in A^t$ if $R(a, b) \in A^t$ and $R_{rC}(b) \in A^t$.

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

Lemma 4. Let $\alpha$ be an assertion and $q$ an RQ. Then:

- $\mathcal{I} \models q$ for all $\mathcal{I} \in \text{mods}(A, \mathcal{T})$ iff $\alpha \in A^t$.

- $\mathcal{I} \models q(a)$ for all $\mathcal{I} \in \text{mods}(A, \mathcal{T})$ iff $R_{rC}(a) \in A^t$.

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 $\text{Ne}$ (unary) and $\text{Nr}$ (binary), and an infinite countable set of variables $V$. A term is a variable $x \in V$ or an individual $a \in N_0$. Atoms take the form $p_n(t)$ or $p_n(t, t')$, where $t, t'$ are terms, $p_n$ is a unary predicate and $p_n$ a binary one. If the terms in an atom do not contain variables, we say they are ground.

$$h_1 \lor \cdots \lor h_k \leftarrow b_1, \ldots, b_k, \text{ not } b_{k+1}, \ldots, \text{ not } b_m$$

where the $h_1, \ldots, h_k$ are the head atoms, and $b_1, \ldots, b_m$ are the body atoms. Among the latter, $b_1, \ldots, b_k$ are positive, and $b_{k+1}, \ldots, b_m$ are negative. A rule with no head atoms (i.e., $k = 0$) is a constraint. A rule $\alpha(n) \leftarrow$ consisting of a single ground head is called a fact. A ground program contains only ground atoms. For a program $P$, its grounding $\text{ground}(P)$ 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 $\text{Ne} \cup \text{Nr}$. An answer set (a.k.a. stable model) of $P$ is a Herbrand interpretation $M$ that is a minimal model of the GL-reduct [Gelfond and Lifschitz, 1988] of $\text{ground}(P)$ w.r.t. $M$, obtained by: (i) deleting every rule $r$ that contains a negative body atom $r(\bar{u})$ with $r(\bar{u}) \in M$, 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 $T$-complete type table $T = (L; S)$ that covers a given set $\mathcal{P}$ containing all the profiles of interest. We show how to obtain a program $P_{\mathcal{T}, q}$ such that, for an input ABox $A$ represented as a program $P_A$ (see below), the answer sets of $P_{\mathcal{T}, q} \cup P_A$ are in close correspondence with the ABoxes $A^t$, so that answering OMRQs amounts to cautious entailment. As anticipated, the rewriting does not depend on a specific ABox,
\[
\bigvee_{p \in \mathbb{P}} \text{type}_p(x) \leftrightarrow \text{prof}_p(x) \quad \text{for each } p \in \mathbb{P} \quad (1)
\]
\[
A(x) \leftrightarrow \text{type}_\tau(x) \quad \text{for each } \tau \in \mathbb{T}' \text{ and each } A \in \tau \quad (2)
\]
\[
s(t,y) \leftrightarrow \text{r}(t,y) \quad \text{for each } r \subseteq T \text{ and } s \quad (3)
\]
\[
\bot \leftrightarrow \text{r}(x,y), A(y), \neg B(x) \quad \text{for each } \exists r.A \sqsubseteq B \in \mathbb{T} \quad (4)
\]
\[
\bot \leftrightarrow \text{r}(x,y), A(y), \neg B(y) \quad \text{for each } A \sqsubseteq \forall r.B \in \mathbb{T} \quad (5)
\]
\[
R_{C}(x) \leftrightarrow \text{type}_\tau(x) \quad \text{for each } \tau \in \mathbb{T} \cap [r,c] \quad (6)
\]
\[
R_{C}(x) \leftrightarrow \text{A}(x), \ldots, A_n(x) \quad \text{with } C = A_1 \sqcap \cdots \sqcap A_n(x) \quad (7)
\]
\[
R_{C}(x) \leftrightarrow \text{r}(x,y), R_{C}(y) \quad (8)
\]

Figure 1: ASP rewriting for reachability queries in ALC\textsc{hit}

but only on a \( T \) 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 \( \mathcal{P}_A \) comprising the rules (1) \textendash{} (5) in the upper part of Figure 1. We use names in \( \mathbb{N}_C \) as unary predicates and (possibly inverse) roles in \( \mathbb{N}_R \) as binary predicates. We also use a unary predicate \( \text{prof}_p \) for each \( p \in \mathbb{P} \) and a unary predicate \( \text{type}_\tau \) for each \( \tau \in \mathbb{T}' \). Intuitively, assuming that the fact \( \text{prof}_p(a) \) holds for each \( a \in \mathbb{N}_i(\mathcal{A}) \) with \( p = \text{prof}_A(a) \), rule (1) guesses assignments of types to individuals. Rules (2) and (3) generate the assertions (a1) and (a2) in \( A' \) for each guess, while (4) and (5) verify conditions (t1) and (t2) in Definition 5.

For a given RQ \( q(x) = \exists y \text{r}(x,y), C(y) \), the rules (6) \textendash{} (8) in the lower part of the figure define the program \( \mathcal{P}_T \); generate the assertions (q1) \textendash{} (q3) in the definition of \( A'^{-} \).

Finally, we represent each given ABox \( \mathcal{A} \) as a program \( \mathcal{P}_A \), with facts \( a \) \textendash{} for all assertions \( a \in \mathbb{A} \), and facts \( \text{prof}_p(a) \) \textendash{} for each \( a \in \mathbb{N}_i(\mathcal{A}) \) with \( p = \text{prof}_A(a) \).

Example 6. Let \( A, T, \mathbb{T} \), and reach[\mathbb{T}_D] be taken from our running example. We get the following ASP rewriting (here we use \( \tau_i \) and \( p_i \) rather than \( \text{type}_\tau \) and \( \text{prof}_p \) as predicates):

\[
\mathcal{P}_A = \begin{align*}
&\tau_i(X) \lor \tau_2(X) \lor \tau_3(X) \leftrightarrow p_1(X). \\
&\tau_i(X) \lor \tau_4(X) \leftrightarrow p_2(X).
\end{align*}
\]

\[
\mathcal{P}_T = \begin{align*}
&\tau_i(X) \lor \tau_2(X) \lor \tau_3(X) \leftrightarrow p_1(X). \\
&\tau_i(X) \lor \tau_4(X) \leftrightarrow p_2(X).
\end{align*}
\]

\[
\mathcal{P}_q = \begin{align*}
&R_{C}(x) \leftrightarrow \tau_2(x), R_{C}(x) \leftrightarrow \tau_4(x). \\
&R_{C}(x) \leftrightarrow \tau_6(x), R_{C}(x) \leftrightarrow \tau_4(x).
\end{align*}
\]

The ASP program \( \mathcal{P}_T \cup \mathcal{P}_A \cup \mathcal{P}_q \) has two answer sets:

\{ \{ \tau_2(a), \tau_4(b), A(a), B(a), C(a), C_1(a), A(b), B(b), R_{C}(a) \} \}

Note their correspondence with the \( T \)-assignments from Example 3. \( C(a) \) is found in all the answer sets, as \( a \) is an instance of \( C \). In contrast, \( q \) from Example 5 has no certain answers: \( R_{C}(a) \) is in only one answer set, \( R_{C}(a) \) in none.

The answer sets of \( \mathcal{P}_T \cup \mathcal{P}_A \) coincide (on the common vocabulary) with the ABoxes \( \mathbb{A}' \) for the different \( T \)-assignments. Similarly, the answer sets of \( \mathcal{P}_T \cup \mathcal{P}_q \cup \mathcal{P}_A \) coincide with the \( \mathbb{A}'^{-} \). From this and Lemma 4 we get:

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

\begin{itemize}
\item For any assertion \( a \), we have \( (T, A) \models a \iff a \in \mathbb{M} \) for all answer sets \( M \) of \( \mathcal{P}_T \cup \mathcal{P}_A \).
\item Given RQ \( q(x) = \exists y \text{r}(x,y), C(y) \), we have \( (T, A) \models q(a) \iff R_{C}(a) \in M \) for all answer sets \( M \) of \( \mathcal{P}_T \cup \mathcal{P}_A \).
\end{itemize}

5 Implementation and Experiments

We have implemented our approach in a prototype reasoner Mod4Q\textsuperscript{2} for the DL, ALC\textsc{hit}. Given a TBox \( T \) and a set \( \mathbb{P} \) of profiles, the reasoner: (1) transforms \( T \) into normal form and drops axioms not in ALC\textsc{hit}; (2) computes \( \text{btyp}^T(\mathbb{P}) \); (3) runs the algorithm described in Section 3.1 to obtain a \( T \)-complete type table \( T \) that covers \( \mathbb{P} \), and (4) generates the ASP program \( \mathcal{P}_T \cup \mathcal{P}_A \) 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 \( T \) and an ABox \( \mathbb{A} \) as an input. In this case, it first extracts the set \{ \( \text{prof}_A(a) \mid a \in \mathbb{N}_i(\mathcal{A}) \} \) of profiles occurring in \( \mathbb{A} \), and then executes steps (1) \textendash{} (3) above. It also generates the program \( \mathcal{P}_A \) for the given \( \mathbb{A} \), additionally to \( \mathcal{P}_T \) 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\textsuperscript{2} 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 64-bit 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 \( \mathcal{P}_A \) includes the extraction of \( \mathbb{P} \) from the ABox, while the time for producing \( \mathcal{P}_T \) includes the full computation of \( T \). Our experiments showed that:

\begin{itemize}
\item 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.
\item Our approach to compiling models and rewriting into ASP is feasible even for large ABoxes. The time to compute \( T \) and \( \mathcal{P}_A \) is very short. The generated ASP programs are simple and perform well when evaluated with Clingo.
\end{itemize}

\footnote{\url{http://www.kr.tuwien.ac.at/research/systems/Mod4Q/}}
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 \( P_T \) ranged from 171 ms to 14 s, with an average of 1.3 s. We report selected results in Table 2.

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 \( T \), and the time required to produce \( T \), 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 bottleneck 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 RQs (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.

Table 1: Querying large ABoxes

<table>
<thead>
<tr>
<th>CN</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DOLCE21</td>
<td>0.3K</td>
<td>1.3K</td>
<td>0.7K</td>
<td>11</td>
<td>2^9</td>
</tr>
<tr>
<td>Gardin.81</td>
<td>0.3K</td>
<td>0.4K</td>
<td>0.3K</td>
<td>27</td>
<td>2^9</td>
</tr>
<tr>
<td>Gardin.283</td>
<td>0.2K</td>
<td>1K</td>
<td>0.5K</td>
<td>11</td>
<td>2^9</td>
</tr>
<tr>
<td>Gardin.284</td>
<td>0.3K</td>
<td>1.3K</td>
<td>0.7K</td>
<td>22</td>
<td>2^9</td>
</tr>
<tr>
<td>OBI350</td>
<td>3.2K</td>
<td>10K</td>
<td>0.4K</td>
<td>38</td>
<td>2^28</td>
</tr>
<tr>
<td>OBO2034</td>
<td>4.5K</td>
<td>7.2K</td>
<td>0.6K</td>
<td>9</td>
<td>2^9</td>
</tr>
<tr>
<td>WINE781</td>
<td>0.6K</td>
<td>0.7K</td>
<td>0.2K</td>
<td>2</td>
<td>2^28</td>
</tr>
</tbody>
</table>

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


