

Effective Query Rewriting with Ontologies over DBoxes (extended abstract)*

İnanç Seylan and Enrico Franconi and Jos de Bruijn

Free University of Bozen-Bolzano, Italy
{seylan,franconi,debruijn}@inf.unibz.it

Abstract. We consider query answering on Description Logic (DL) ontologies with DBoxes, where a DBox is a set of assertions on individuals involving atomic concepts and roles called DBox predicates. The extension of a DBox predicate is exactly defined in every interpretation by the contents of the DBox, i.e., a DBox faithfully represents a database whose table names are the DBox predicates and the tuples are the DBox assertions. Our goals are (i) to find out whether the answers to a given query are solely determined by the DBox predicates and, if so, (ii) to find a rewriting of the query in terms of them. The resulting query can then be efficiently evaluated using standard database technology. We have that (i) can be reduced to entailment checking and (ii) can be reduced to finding an interpolant. We present a procedure for computing interpolants in the DL \mathcal{ALC} with general TBoxes. We extend the procedure with standard tableau optimisations, and we discuss abduction as a technique for amending ontologies to gain definability of queries of interest.

1 Introduction

We address the problem of concept query answering on databases with ontologies. An ontology provides a conceptual view of the database and it is composed by constraints on a vocabulary extending the basic vocabulary (tables and attributes) of the data. Querying a database using the terms in such a richer ontology allows for more flexibility than using only the basic vocabulary of the relational database directly.

Description Logics [1] (DLs) are a prominent formalism for representing ontologies. However, DLs such as \mathcal{ALC} or OWL or DL-lite were not originally developed with this use case in mind. In particular, the data in a DL knowledge base is represented using an ABox, which can be seen as an *incomplete* database. This means the extensions of the concepts and roles contain at least the data mentioned in the ABox, but may contain additional data, and this may vary among the models of the knowledge base. This is in contrast to relational

* This paper is an excerpt from the IJCAI-09 paper “Effective Query Rewriting with Ontologies over DBoxes” by the same authors. We wish to thank Alex Borgida, Tommaso Di Noia, Umberto Straccia and David Toman with whom we are studying a more general framework on query rewriting based on Beth definability and abduction, and the anonymous reviewers for insightful comments. The work presented in this paper has been partially funded by the European project ONTORULE.

databases, which are *complete*: the extensions of the predicates (i.e., the tables) contain exactly the data in the database and nothing more.

We introduce in this paper the notion of a *DBox*, which is syntactically similar to an ABox; it is a set of ground atomic concept and role assertions. However, semantically it behaves like a database, i.e., the extensions of the concepts and roles mentioned in the DBox are exactly defined by the contents of the DBox. We call the concepts and roles appearing in the DBox the *DBox predicates*. Observe that the DBox predicates are *closed*, i.e., their extensions are the same in every interpretation, whereas the other predicates in the knowledge base are *open*, i.e., their extensions may vary among different interpretations. This has been called also *locally closed world* [2] or *exact views* [3,4,5].

The queries we consider in this paper are concept expressions (and the answers are their instances), and the ontologies are general \mathcal{ALC} TBoxes. Our goals are (i) to check whether the answers to a given query under a TBox are *solely* determined by the extension of the DBox predicates and, if so, (ii) to find an equivalent rewriting of the query in terms of the DBox predicates to allow the use of standard database technology for answering the query. This means we benefit from the low computational complexity in the size of the data of answering first-order queries on relational databases. In addition, we would like as much as possible to use standard techniques for DL reasoning to find rewritings. As was pointed out recently also by [6], (i) corresponds to *implicit definability* [7], and can be reduced to checking entailment and (ii) corresponds to *explicit definability* [8]. Inspired by the results of [9], this problem can be reduced to finding an *interpolant*.

Our contributions in this paper are as follows. We introduce the notion of DBoxes as a faithful encoding of databases, and show how to find a query rewriting over DBoxes, using Beth definability and interpolation; we present a procedure for calculating interpolants of \mathcal{ALC} concepts under general TBoxes, using an adaptation of standard DL tableau techniques; we extend the procedure to deal with common tableau optimisation techniques found in implemented systems, in particular, non-atomic closure, semantic branching, lazy unfolding and absorption, and backjumping; and we show how abduction techniques can be used for amending TBoxes to gain definability of queries of interest.

2 \mathcal{ALC} and DBoxes

\mathcal{ALC} TBoxes. Let N_C and N_R be countably infinite sets of concept and role names, respectively. The set of \mathcal{ALC} *concepts* is the smallest set that includes the atomic concepts $N_C \cup \{\top\}$, and if C, D are concepts and $R \in N_R$, then $\neg C$, $C \sqcap D$, and $\forall R.C$ are concepts. As usual, \perp , $C \sqcup D$, and $\exists R.C$ are short for $\neg \top$, $\neg(\neg C \sqcap \neg D)$, and $\neg \forall R.\neg C$, respectively. An \mathcal{ALC} *TBox* \mathcal{T} is a finite set of axioms of the form $C \sqsubseteq D$, where C and D are \mathcal{ALC} concepts.

An *interpretation* \mathcal{I} is a pair $\langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ where $\Delta^{\mathcal{I}}$ is a non-empty set and $\cdot^{\mathcal{I}}$ is a function that maps concept names A to subsets $A^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$ and role names R to binary relations $R^{\mathcal{I}}$ on $\Delta^{\mathcal{I}}$; $\cdot^{\mathcal{I}}$ extends to concepts as follows: $\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$; $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$; $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$; and $(\forall R.C)^{\mathcal{I}} = \{\delta \in \Delta^{\mathcal{I}} \mid \forall \delta' ((\delta, \delta') \in R^{\mathcal{I}} \rightarrow \delta' \in C^{\mathcal{I}})\}$.

\mathcal{I} is a *model* of a TBox \mathcal{T} , written $\mathcal{I} \models \mathcal{T}$, iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for all axioms $C \sqsubseteq D \in \mathcal{T}$. Since $C \sqsubseteq D$ is equivalent to $\top \sqsubseteq \neg C \sqcup D$, we assume that all axioms in \mathcal{T} are of this form. We can thus view the TBox as a set of universal concepts $\{C \mid \top \sqsubseteq C \in \mathcal{T}\}$.

A concept C is *satisfiable* w.r.t. \mathcal{T} iff there is some model \mathcal{I} of \mathcal{T} such that $C^{\mathcal{I}} \neq \emptyset$; D *subsumes* C w.r.t. \mathcal{T} , written $C \sqsubseteq_{\mathcal{T}} D$, iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for all models \mathcal{I} of \mathcal{T} . We write $C \equiv_{\mathcal{T}} D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ and $D^{\mathcal{I}} \subseteq C^{\mathcal{I}}$.

For ease of presentation, we assume all concepts to be in *negation normal form* (NNF), which means the negation signs appear only in front of atomic concepts. The negation normal form of the complement of a concept C is written $\neg C$. With $\text{sig}(C)$ (resp. $\text{sig}(\mathcal{T})$) we denote the set of all concept and role names occurring in C (resp. \mathcal{T}), i.e., the *signature* of C (resp. \mathcal{T}). With $\text{sig}(C, \mathcal{T})$ we denote the set of all concept and role names occurring in C or \mathcal{T} .

DBoxes. Let N_I be a countably infinite set of individual names. A DBox \mathcal{D} is a set of *assertions* of the forms $A(a)$ and $R(a, b)$, where $A \in \sigma_{\mathcal{D}}(C)$, $R \in \sigma_{\mathcal{D}}(R)$, and $a, b \in \sigma_{\mathcal{D}}(I)$. The *signature* $\sigma_{\mathcal{D}}$ of \mathcal{D} consists of the *DBox concepts* $\sigma_{\mathcal{D}}(C) \subseteq N_C$, the *DBox roles* $\sigma_{\mathcal{D}}(R) \subseteq N_R$, and the *DBox individuals* $\sigma_{\mathcal{D}}(I) \subseteq N_I$; we call *DBox predicates* the set $\sigma_{\mathcal{D}}(P) = \sigma_{\mathcal{D}}(C) \cup \sigma_{\mathcal{D}}(R)$. The *active domain* of \mathcal{D} is the set of individuals in $\sigma_{\mathcal{D}}(I)$ appearing in the assertions.

An interpretation $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ is a model of \mathcal{D} , written $\mathcal{I} \models \mathcal{D}$, iff $a^{\mathcal{I}} = a$ for every DBox individual $a \in \sigma_{\mathcal{D}}(I)$, and for every concept (resp., role) name P in $\sigma_{\mathcal{D}}(P)$ and every $u \in \Delta^{\mathcal{I}}$ (resp., $(u, v) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$), we have that $u \in P^{\mathcal{I}}$ iff $P(u) \in \mathcal{D}$ (resp., $(u, v) \in P^{\mathcal{I}}$ iff $P(u, v) \in \mathcal{D}$). In other words, in every model of \mathcal{D} the extensions of the DBox predicates are given by the contents of the DBox, and are the same in every model. Please note that the domain $\Delta^{\mathcal{I}}$ of a model of \mathcal{D} is not fixed, but it includes all the DBox individuals in $\sigma_{\mathcal{D}}(I)$, which in turn includes the active domain of \mathcal{D} .

A DBox \mathcal{D} is *satisfiable* with respect to a TBox \mathcal{T} iff there is a model of \mathcal{D} that is also a model of \mathcal{T} . An interpretation \mathcal{I} is a model of $C(a)$, where C is a concept and a an individual name, written $\mathcal{I} \models C(a)$, iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$. A pair $(\mathcal{T}, \mathcal{D})$ *entails* a membership expression $C(a)$ – this is the instance checking problem, or *concept querying* problem – written $(\mathcal{T}, \mathcal{D}) \models C(a)$, iff for every model \mathcal{I} of \mathcal{T} and \mathcal{D} it holds that \mathcal{I} is a model of $C(a)$. Finally, \mathcal{D} *entails* $C(a)$, written $\mathcal{D} \models C(a)$, iff $(\emptyset, \mathcal{D}) \models C(a)$.

The *unique names assumption* (UNA) holds for a DL \mathcal{L} if for any interpretation \mathcal{I} considered in \mathcal{L} , $a^{\mathcal{I}} \neq b^{\mathcal{I}}$ whenever $a \neq b$, for any two $a, b \in N_I$. If we assume UNA, reasoning in a DL with DBoxes can be reduced to reasoning in the same DL extended with nominals – i.e., concept expressions of the form $\{a\}$, where a is an individual name, such that $\{a\}^{\mathcal{I}} = \{a^{\mathcal{I}}\}$ – and vice versa.

Proposition 1. *Given a description logic \mathcal{L} with UNA that includes \mathcal{ALC} , for every DBox \mathcal{D} and TBox \mathcal{T} in \mathcal{L} there exists a TBox $\mathcal{T}^{\mathcal{D}}$ in \mathcal{L} extended with nominals with size polynomially bounded by the size of $\mathcal{T} \cup \mathcal{D}$, such that \mathcal{D} is satisfiable with respect to \mathcal{T} iff $\mathcal{T}^{\mathcal{D}}$ is satisfiable, and vice versa.*

Proof (sketch). (\Rightarrow) $\mathcal{T}^{\mathcal{D}}$ is obtained from \mathcal{T} by adding completion and closure axioms. For every DBox concept A add the axiom $A \equiv \{a_1\} \sqcup \dots \sqcup \{a_n\}$, such that $A(a_1), \dots, A(a_n)$ are the assertions involving A in \mathcal{D} . For every DBox role

R add the axioms $\{a_i\} \sqsubseteq \exists R.\{b_j\}$, $\{a_i\} \sqsubseteq \forall R.\{b_1\} \sqcup \dots \sqcup \{b_n\}$, and $\exists R.\top \sqsubseteq \{a_1\} \sqcup \dots \sqcup \{a_m\}$, such that $R(a_i, b_j)$ for $i = 1, \dots, m$ and $j = 1, \dots, n$ are the assertions involving R in \mathcal{D} .

(\Leftarrow) For each occurrence of $\{a\}$ in $\mathcal{T}^\mathcal{D}$, the assertion $A^a(a)$ is added to \mathcal{D} , where A^a is a new concept name; each occurrence of $\{a\}$ in $\mathcal{T}^\mathcal{D}$ is replaced with A^a .

Please note that this reduction encodes the entire DBox into a TBox, and that, although concept querying in \mathcal{ALCO} has the same combined complexity as in \mathcal{ALC} , if we consider more expressive logics such as \mathcal{SHIQ} we do get an increase in combined complexity when nominals are added (from EXPTIME to NEXPTIME). It can also be shown that the data complexity of query answering with DBoxes (i.e., with closed data predicates) is at least as hard as query answering with ABoxes (i.e., with open data predicates), and that it is strictly harder if we consider conjunctive query answering in simple description logics such as DL-lite (it increases from LOGSPACE to CONP-hard , which can be shown by a reduction from non-existence of 3-colourings).

3 Definability and Query Answering

We introduce in this Section implicit and explicit *definability* for queries, and discuss how *explicit* definitions can be used for concept query rewriting. In the following, given a TBox \mathcal{T} and a DBox \mathcal{D} , let Q be a query concept (i.e., a query asking for all the instances of the concept) and let $\text{sig}(Q, \mathcal{T}) = \{B_1, \dots, B_m, D_1, \dots, D_n\}$ be the combined signature where $\{D_1, \dots, D_n\} \subseteq \sigma_{\mathcal{D}}(P)$ and $\{B_1, \dots, B_m\} \cap \sigma_{\mathcal{D}}(P) = \emptyset$; $\{D_1, \dots, D_n\}$ is the set of DBox predicates appearing in \mathcal{T} or in Q .

Definition 1 (Implicit definability). *Let a concept φ' (resp. a TBox \mathcal{T}') be like φ (resp. \mathcal{T}) but with occurrences of B_1, \dots, B_m replaced by distinct occurrences of $B'_1, \dots, B'_m \notin \text{sig}(Q, \mathcal{T})$. Then, Q is implicitly definable from D_1, \dots, D_n in \mathcal{T} iff $Q \equiv_{\mathcal{T} \cup \mathcal{T}'} Q'$.*

In other words, given a TBox, a concept Q is implicitly definable if the set of all its instances depends only on the extension of the DBox predicates. This means that it may be possible to find an expression using only predicates in the DBox whose instances are the same as in the original concept: this would be its explicit definition.

Definition 2 (Explicit definability). *Q is explicitly definable from D_1, \dots, D_n in \mathcal{T} iff there is some concept C such that $Q \equiv_{\mathcal{T}} C$ and $\text{sig}(C) \subseteq \{D_1, \dots, D_n\}$.*

Clearly, explicit definability implies implicit definability. [8] shows that the converse also holds for the case of first-order logic: if Q is implicitly definable from D_1, \dots, D_n in \mathcal{T} , then it is explicitly definable. To prove this property for \mathcal{ALC} with general TBoxes, we exploit interpolation.

Lemma 1 (Interpolation for \mathcal{ALC} with TBoxes). *Let Q be an \mathcal{ALC} concept and let \mathcal{T} be an \mathcal{ALC} TBox such that $Q \sqsubseteq_{\mathcal{T} \cup \mathcal{T}'} Q'$, where Q' and \mathcal{T}' are obtained as above. Then, there exists some concept C , called the interpolant of Q and Q' under \mathcal{T} and \mathcal{T}' , such that $\text{sig}(C) \subseteq \text{sig}(Q, \mathcal{T}) \cap \text{sig}(Q', \mathcal{T}')$, $Q \sqsubseteq_{\mathcal{T} \cup \mathcal{T}'} C$, and $C \sqsubseteq_{\mathcal{T} \cup \mathcal{T}'} Q'$.*

Theorem 1 (Beth Definability for \mathcal{ALC} with TBoxes). *If Q is implicitly definable from D_1, \dots, D_n in \mathcal{T} then Q is explicitly definable from D_1, \dots, D_n in \mathcal{T} .*

The proof of Beth definability for \mathcal{ALC} with general TBoxes is constructive, provided we have a constructive method of finding interpolants as defined in Lemma 1. In Section 4 we present such a method, and we prove its correctness and completeness.

We can now combine these results to reduce answering definable queries to query answering using only the DBox.

Theorem 2. *Let \mathcal{D} be a DBox that is satisfiable with respect to a TBox \mathcal{T} , let Q be a concept and let $a \in \sigma_{\mathcal{D}}(I)$ be a DBox individual. If Q is implicitly definable from $\sigma_{\mathcal{D}}(P)$, then there is an \mathcal{ALC} concept C with $\text{sig}(C) \subseteq \sigma_{\mathcal{D}}(P)$ such that $(\mathcal{T}, \mathcal{D}) \models Q(a)$ iff $\mathcal{D} \models C(a)$.*

To enable rewriting $C(a)$ to standard database query languages we first need to prove its *domain independence*.

Definition 3 (Domain independence). *Let C be a concept and let a be an individual name. $C(a)$ is said to be domain independent if for every interpretation $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ such that $\mathcal{I} \models C(a)$ it is the case that $\mathcal{I}' \models C(a)$ for every interpretation $\mathcal{I}' = \langle \Delta^{\mathcal{I}'}, \cdot^{\mathcal{I}'} \rangle$ with $\Delta^{\mathcal{I}'} \supseteq \Delta^{\mathcal{I}}$ and $\cdot^{\mathcal{I}} = \cdot^{\mathcal{I}'}$.*

Theorem 3. *Let C be an \mathcal{ALC} concept and let a be an individual name. Then, $C(a)$ is domain independent.*

Consider a query $C(a)$ with $a \in \sigma_{\mathcal{D}}(I)$ and $\text{sig}(C) \subseteq \sigma_{\mathcal{D}}(P)$. Since $C(a)$ is domain independent, deciding $\mathcal{D} \models C(a)$ can be reduced to checking $a^{\mathcal{I}} \in C^{\mathcal{I}}$ for an arbitrary model \mathcal{I} of \mathcal{D} ; in particular we choose the smallest model $\mathcal{I}^{\mathcal{D}} = \langle \Delta^{\mathcal{I}^{\mathcal{D}}}, \cdot^{\mathcal{I}^{\mathcal{D}}} \rangle$ of \mathcal{D} , where $\Delta^{\mathcal{I}^{\mathcal{D}}}$ is equal to the set of the DBox individuals in $\sigma_{\mathcal{D}}(I)$ and $\cdot^{\mathcal{I}^{\mathcal{D}}}$ considers only the set of DBox assertions. In other words, the entailment check reduces to model checking over $\mathcal{I}^{\mathcal{D}}$ and a standard database query language over the DBox database can be used for deciding $\mathcal{D} \models C(a)$.

Given a query concept Q , a TBox \mathcal{T} , and a DBox \mathcal{D} , in order to find the answer set $\{a \mid (\mathcal{T}, \mathcal{D}) \models Q(a)\}$ by means of the rewriting $\{a \mid \mathcal{D} \models C(a)\}$, one can use the equivalent database query $Q(x) :- C^{\text{FO}}(x) \wedge \top_{\mathcal{D}}(x)$ over the database \mathcal{D} , where C^{FO} is the standard first order translation of C and $\top_{\mathcal{D}}$ is the relation containing exactly the DBox individuals $\sigma_{\mathcal{D}}(I)$. It is possible to pre-compute offline the rewriting of all definable atomic concepts in the ontology as materialised SQL views, so that at run time query answering on \mathcal{ALC} ontologies with DBoxes is reduced to answering SQL queries over the DBox database with materialised views.

4 Constructive Interpolation for \mathcal{ALC}

[10] and [11] give constructive proofs using symmetric Gentzen systems, respectively tableau calculi. [12] provides an algorithm for calculating interpolants between \mathcal{ALC} concepts; however, he does not consider TBoxes, nor does he provide a reference with standard or optimised implemented tableau algorithms.

The R_{\sqcap} rule

Condition: $(x : C_1 \sqcap C_2)^\lambda \in L(g)$ and $\{(x : C_1)^\lambda, (x : C_2)^\lambda\} \not\subseteq L(g)$

Effect: $L(g') = L(g) \cup \{(x : C_1)^\lambda, (x : C_2)^\lambda\}$

The R_{\sqcup} rule

Condition: $(x : C_1 \sqcup C_2)^\lambda \in L(g)$ and $\{(x : C_1)^\lambda, (x : C_2)^\lambda\} \cap L(g) = \emptyset$

Effect: $L(g') = L(g) \cup \{(x : C_1)^\lambda\}$, $L(g'') = L(g) \cup \{(x : C_2)^\lambda\}$

The R_{\exists} rule

Condition: $(x : \exists R.C)^\lambda \in L(g)$, there is no variable y in $L(g)$ such that $y : \{C^\lambda\} \cup \text{succ}(x, R, S) \subseteq L(g)$

Effect: $L(g') = L(g) \cup y : \{C^\lambda\} \cup \text{succ}(x, R, S) \cup \mathcal{T}$, where y is fresh for S

Fig. 1. Biased completion rules for \mathcal{ALC}

Let N_V be a countably infinite set of variable names and $<$ a well-order relation on N_V . A *biased constraint* is an expression of the form $(x : D)^\lambda$ where $x \in N_V$, D is a concept, and $\lambda \in \{l, r\}$ is a bias. A *biased constraint system* S for $\langle C, \mathcal{T} \rangle$ is a finite, non-empty set of biased constraints.

We say a variable $x \in N_V$ is in S if S contains a mention of x ; x is *fresh* for S if x is not in S and $y < x$ for all y in S . We assume that when a variable x is in S , the constraints $(x : \top)^l$, $(x : \top)^r$ are also in S .

Let x be a variable in S . The set of *x -constraints* of S is defined as $\{x : C \in S\}$. For a role name R , with $\text{succ}(x, R, S)$ we denote the set $\{C^\lambda \mid (x : \forall R.C)^\lambda \in S\}$. If X is a set of labelled concepts, then $x : X$ is a shorthand for $\{(x : C)^\lambda \mid C^\lambda \in X\}$.

S is said to contain a *clash* if for some variable x and some concept name A , $\{(x : A)^\lambda, (x : \neg A)^\kappa\} \subseteq S$.

Let H, J be \mathcal{ALC} concepts and $\mathcal{T} = \mathcal{T}^l \cup \mathcal{T}^r$ a set of biased concepts with \mathcal{T}^l being l - and \mathcal{T}^r being r -labelled concept. A *biased tableau* for $\langle H \sqcap \dot{\neg} J, \mathcal{T} \rangle$ is a triple $\mathbf{T} = \langle V, E, L \rangle$, where $\langle V, E \rangle$ is a finite tree, with $g_0 \in V$ being the root node, and L is a labelling function associating with each node $g \in V$ a biased constraint system for $\langle H \sqcap \dot{\neg} J, \mathcal{T} \rangle$ and with each edge $\langle g, g' \rangle \in E$ a *biased completion rule* R_x from Figure 1. In addition, we require $L(g_0) = x_0 : \{H^l\} \cup \{(\dot{\neg} J)^r\} \cup \mathcal{T} \cup \mathcal{T}'$. A biased tableau \mathbf{T}_0 for $\langle H \sqcap \dot{\neg} J, \mathcal{T} \rangle$ that contains only the root node is called the *initial biased tableau* for $\langle H \sqcap \dot{\neg} J, \mathcal{T} \rangle$. A completion rule is *applicable* in a node g if its condition is satisfied in g .

A *branch* of a tableau is a path from the root down to a leaf. If in a tableau there is some successor g' of g such that $L(\langle g, g' \rangle) = R_{\sqcup}$, then g is called a *branching point* in the tableau. A branch is *closed* if the label of its leaf node contains a clash; otherwise it is open. A tableau is *closed* if all its branches are closed; otherwise it is open. A tableau is *complete* if no completion rule is applicable in the leaf nodes of any of its open branches.

The *biased tableau algorithm* takes as input the initial biased tableau \mathbf{T}_0 for $\langle H \sqcap \dot{\neg} J, \mathcal{T} \rangle$. \mathbf{T}_0 is then expanded by repeatedly applying the completion rules given in Figure 1 with the following strategy: R_{\exists} is applied only when R_{\sqcup} is not applicable which in turn is applied only when R_{\sqcap} is not applicable. If the obtained complete tableau is closed, the algorithm returns false, otherwise true.

If the biased tableau algorithm returns false, then a second phase is initiated to extract an interpolant I .

$$\begin{array}{l}
C_{\neg}(ll) \frac{(x : A)^l, (x : \neg A)^l \in L(g)}{int(g)(x) := \perp} \\
C_{\neg}(rr) \frac{(x : A)^r, (x : \neg A)^r \in L(g)}{int(g)(x) := \top} \\
C_{\neg}(lr) \frac{(x : A)^l, (x : \neg A)^r \in L(g)}{int(g)(x) := A} \\
C_{\neg}(rl) \frac{(x : A)^r, (x : \neg A)^l \in L(g)}{int(g)(x) := \neg A}
\end{array}$$

Fig. 2. Atomic interpolant calculation rules for \mathcal{ALC}

$$\begin{array}{l}
C_{\sqcap}(\lambda) \frac{(x : C_1 \sqcap C_2)^\lambda \in L(g)}{L(g') = L(g) \cup \{(x : C_1)^\lambda, (x : C_2)^\lambda\}} \\
\frac{int(g) := int(g')}{C_{\sqcap}(l)} \\
C_{\sqcap}(l) \frac{(x : C_1 \sqcup C_2)^l \in L(g)}{L(g') = L(g) \cup \{(x : C_1)^l\}} \\
\frac{L(g'') = L(g) \cup \{(x : C_2)^l\}}{int(g) := int(g') \sqcup int(g'')} \\
C_{\sqcap}(r) \frac{(x : C_1 \sqcup C_2)^r \in L(g)}{L(g') = L(g) \cup \{(x : C_1)^r\}} \\
\frac{L(g'') = L(g) \cup \{(x : C_2)^r\}}{int(g)(y) := int(g')(y) \sqcup int(g'')(y) (y \neq x)} \\
\frac{int(g)(x) := int(g')(x) \sqcap int(g'')(x)}{C_{\sqcap}(r)}
\end{array}$$

Fig. 3. Propositional interpolant calculation rules for \mathcal{ALC}

Definition 4. Let g be a node in a tableau and let $\{(x : C_1)^l, \dots, (x : C_n)^l, (x : D_1)^r, \dots, (x : D_m)^r\}$ be the x -constraints of $L(g)$. A concept I is called an interpolant for this set under \mathcal{T} if I is an interpolant for $C_1 \sqcap \dots \sqcap C_n$ and $\neg D_1 \sqcup \dots \sqcup \neg D_m$ under \mathcal{T}^l and \mathcal{T}^r . (Take the empty intersection to be \top and the empty union to be \perp .)

Let C and D be concepts. We define the binary infix operator \sqcup as follows: $C \sqcup D = C \sqcup D$, $C \sqcup \perp = \perp \sqcup C = C$, $C \sqcup \text{null} = \text{null} \sqcup C = C$, and $\text{null} \sqcup \text{null} = \text{null}$. If $ht1$ and $ht2$ are hash tables, $ht' = ht1 \sqcup ht2$ is defined as: $ht'(x) = ht1(x) \sqcup ht2(x)$, for every variable x . The operator \sqcap is defined analogously.

Interpolants at nodes g are stored in hash tables $int(g)$. Every key in $int(g)$ is a variable in $L(g)$ and every value is a concept or the special symbol null . With $int(g)(x)$ we denote the value associated with the key x . The algorithm used in the second phase starts from the leaves of the tableau tree and applies the interpolant calculation rules $C_\chi(\lambda)$ in Figures 2, 3, and 4, which are if-then rules. If the rules do not assign a value, it is assumed to be null .

Lemma 2 (Soundness). Let \mathbf{T} be a closed biased tableau for $\langle H \sqcap \dot{\neg} J, \mathcal{T} \rangle$, g a node in the tableau, and x a variable in $L(g)$. If $int(g)(x)$ is not null , it is an interpolant for the x -constraints of $L(g)$.

Lemma 3 (Completeness). Let \mathbf{T} be a closed biased tableau for $\langle H \sqcap \dot{\neg} J, \mathcal{T} \rangle$ with root node g_0 . Then, $int(g_0)(x_0) \neq \text{null}$.

$$\begin{array}{c}
\text{C}_{\exists(l)} \frac{(x : \exists R.C)^l \in L(g) \quad L(g') = L(g) \cup (y : \{C^l\} \cup X)}{\begin{array}{l} \text{int}(g)(z) := \text{int}(g')(z) \quad (z \neq x \ \& \ z \neq y) \\ \text{int}(g)(y) := \text{null} \quad I := \text{int}(g')(y) \\ \text{int}(g)(x) := \text{int}(g')(x) \quad [\text{if } I = \text{null}] \\ \text{int}(g)(x) := \text{int}(g')(x) \sqcup \perp \quad [\text{if } I = \perp] \\ \text{int}(g)(x) := \text{int}(g')(x) \sqcup \exists R.I \quad [\text{otherwise}] \end{array}} \\
\text{C}_{\exists(r)} \frac{(x : \exists R.C)^r \in L(g) \quad L(g') = L(g) \cup (y : \{C^r\} \cup X)}{\begin{array}{l} \text{int}(g)(z) := \text{int}(g')(z) \quad (z \neq x \ \& \ z \neq y) \\ \text{int}(g)(y) := \text{null} \quad I := \text{int}(g')(y) \\ \text{int}(g)(x) := \text{int}(g')(x) \quad [\text{if } I = \text{null}] \\ \text{int}(g)(x) := \top \quad [\text{if } I = \top] \\ \text{int}(g)(x) := \text{int}(g')(x) \sqcup \forall R.I \quad [\text{otherwise}] \end{array}}
\end{array}$$

Fig. 4. Modal interpolant calculation rules for \mathcal{ALC}

$$\begin{array}{c}
\text{C}_{\cdot}(ll) \frac{(x : C)^l, (x : \dot{\neg}C)^l \in L(g)}{\text{int}(g)(x) := \perp} \\
\text{C}_{\cdot}(rr) \frac{(x : C)^r, (x : \dot{\neg}C)^r \in L(g)}{\text{int}(g)(x) := \top} \\
\text{C}_{\cdot}(lr) \frac{(x : C)^l, (x : \dot{\neg}C)^r \in L(g)}{\text{int}(g)(x) := C}
\end{array}$$

Fig. 5. Interpolant calculation rules for non-atomic clashes

Finally, we remark that the interpolant calculation algorithm can be made more space-efficient by integrating it into the satisfiability checking algorithm (*one-pass*) and generating the tableau in a depth-first manner: interpolants are constructed for each subbranch while backtracking, so that the subbranch may be discarded from memory before exploring the next subbranch. Interpolants are combined at the branching points using the $\text{C}_{\sqcup}(\lambda)$ rule. Please note that, in this way, the final algorithm to compute the interpolant can be obtained by simply *adorning* a standard tableau algorithm for \mathcal{ALC} .

5 Integrating Tableau Optimisations

In this section, we extend the tableau algorithm with commonly used optimisations, which can be applied to the *one-pass* algorithm sketched above, so that the process to compute the interpolant is just an adorned variant of the standard and optimised implemented tableau algorithm for \mathcal{ALC} .

Non-atomic Closure. One can relax the definition of the closure condition for a tableau such that a constraint system S is said to contain a *clash* if for some variable x and some concept C , $\{x : C, x : \dot{\neg}C\} \subseteq S$. For example the unsatisfiability of the constraint system $S \supseteq \{C \sqcap D, \dot{\neg}(C \sqcap D)\}$ can be detected without branching for $\dot{\neg}(C \sqcap D)$. Figure 5 depicts the calculation rules for non-atomic clashes, replacing the ones in Figure 2. Correctness is proved analogously to the atomic case.

The R_{U1} rule

Condition: $(x : A)^\kappa \in L(g)$, $A \equiv C$ is in \mathcal{T}_U (or \mathcal{T}'_U), and $(x : C)^l \notin L(g)$ (or $(x : C)^r \notin L(g)$)

Effect: $L(g') = L(g) \cup \{(x : C)^l\}$ (or $\{(x : C)^r\}$)

The R_{U2} rule

Condition: $(x : \neg A)^\kappa \in L(g)$, $A \equiv C$ is in \mathcal{T}_U (or \mathcal{T}'_U), and $(x : \dot{\neg}C)^l \notin L(g)$ (or $(x : \dot{\neg}C)^r \notin L(g)$)

Effect: $L(g') = L(g) \cup \{(x : \dot{\neg}C)^l\}$ (or $\{(x : \dot{\neg}C)^r\}$)

The R_{U3} rule

Condition: $(x : A)^\kappa \in L(g)$, $A \sqsubseteq C$ is in \mathcal{T}_U (or \mathcal{T}'_U), and $(x : C)^l \notin L(g)$ (or $(x : C)^r \notin L(g)$)

Effect: $L(g') = L(g) \cup \{(x : C)^l\}$ (or $\{(x : C)^r\}$)

Fig. 6. Lazy unfolding rules

Semantic Branching. The standard definition of R_{\sqcup} (cf. Figure 1), which is based on *syntactic branching*, does not prevent the recurrence of unsatisfiable disjuncts in different branches of the tree. *Semantic branching* is a technique that allows the subtrees introduced by non-deterministic rules to be distinct in the sense that the satisfiability of a disjunct is searched only in a single subtree [13]. This requires a slight change in the R_{\sqcup} rule: the second effect becomes $L(g'') = L(g) \cup \{x : \dot{\neg}C_1, x : C_2\}$, obtaining the new rule $R_{\sqcup'}$. Replacing R_{\sqcup} with $R_{\sqcup'}$ in our tableau calculus requires modifying the third condition of $C_{\sqcup}(\lambda)$ in Figure 3 with $L(g'') = L(g) \cup \{(x : \dot{\neg}C_1)^\lambda, (x : C_2)^\lambda\}$. However, the way the interpolants are calculated does not change.

Lazy Unfolding and Absorption. Absorption is a TBox rewriting technique that has been developed to address the problem of the high level of non-determinism caused by general TBox axioms [13]. Being a rewriting technique, absorption of general axioms into simple axioms is performed before the satisfiability algorithm starts. For this reason, we are only concerned here with the structure of the output TBox. Given a general TBox \mathcal{T} , the absorption process rewrites and partitions \mathcal{T} into the *general* TBox \mathcal{T}_G and the *unfoldable* TBox \mathcal{T}_U [13] such that a concept is satisfiable w.r.t. \mathcal{T} iff it is satisfiable w.r.t. $\mathcal{T}_U \cup \mathcal{T}_G$.

The implicit definability problem takes as input two TBoxes \mathcal{T} and \mathcal{T}' . Therefore, we assume w.l.o.g. that the absorption process is only applied to \mathcal{T} , resulting in \mathcal{T}_U and \mathcal{T}_G , and \mathcal{T}'_U and \mathcal{T}'_G are obtained analogously to \mathcal{T}' . R_{\exists} is modified to use only \mathcal{T}_G and \mathcal{T}'_G and, in addition, we need the rules in Figure 6 that handle unfoldable TBoxes. In the literature, these rules constitute what is known as the *lazy unfolding* optimisation [13].

To complement these rules we use the interpolant calculation rules given in Figure 7. One needs to verify the soundness of these rules. We show only $C_{U1}(l)$ for \mathcal{T}_U . Suppose $X = \{(x : C_1)^l, \dots, (x : C_n)^l, (x : D_1)^r, \dots, (x : D_m)^r\}$, g' is the result of applying R_{U1} to g for $A \equiv C \in \mathcal{T}_U$, and $X \cup \{(x : A)^\kappa, (x : C)^l\}$ are the x -constraints of $L(g')$. Note that although κ may not be equal to l , we can assume w.l.o.g. that $\kappa = l$ since $A \in \text{sig}(\mathcal{T}_U)$. By our assumption, $\text{int}(g')(x)$ is an interpolant for the x -constraints of $L(g')$ and by Definition 4, $A \sqcap C \sqcap C_1 \sqcap \dots \sqcap C_n \sqsubseteq_{\mathcal{T}} \text{int}(g')(x) \sqsubseteq_{\mathcal{T}} \neg D_1 \sqcup \dots \sqcup \neg D_m$. C_{U1} assigns $\text{int}(g')(x)$ to $\text{int}(g)(x)$. We have that (i) $A \sqcap C_1 \sqcap \dots \sqcap C_n \sqsubseteq_{\mathcal{T}} \text{int}(g')(x)$ follows from

$$\begin{array}{l}
\mathcal{C}_{U1}(\lambda) \quad \frac{(x : A)^\kappa \in L(g) \quad A \equiv C \in \mathcal{T}_U \text{ (or } \mathcal{T}'_U)}{L(g') = L(g) \cup \{(x : C)^\lambda\}} \\
\qquad \qquad \qquad \text{int}(g) := \text{int}(g') \\
\mathcal{C}_{U2}(\lambda) \quad \frac{(x : \neg A)^\kappa \in L(g) \quad (A \equiv C)^\lambda \in \mathcal{T}_U \text{ (or } \mathcal{T}'_U)}{L(g') = L(g) \cup \{(x : \neg C)^\lambda\}} \\
\qquad \qquad \qquad \text{int}(g) := \text{int}(g') \\
\mathcal{C}_{U3}(\lambda) \quad \frac{(x : A)^\kappa \in L(g) \quad (A \sqsubseteq C)^\lambda \in \mathcal{T}_U \text{ (or } \mathcal{T}'_U)}{L(g') = L(g) \cup \{(x : C)^\lambda\}} \\
\qquad \qquad \qquad \text{int}(g) := \text{int}(g')
\end{array}$$

Fig. 7. Lazy unfolding interpolant calculation rules

$(A \sqcap C \sqcap C_1 \sqcap \dots \sqcap C_n)^\mathcal{I} = (A \sqcap C_1 \sqcap \dots \sqcap C_n)^\mathcal{I}$, for all models \mathcal{I} of \mathcal{T} , since $A \equiv C \in \mathcal{T}_U$ and (ii) $\text{int}(g')(x) \sqsubseteq_{\mathcal{T}} \neg D_1 \sqcup \dots \sqcup \neg D_m$ follows from the fact that $L(g)$ has the same r -labelled x -constraints as $L(g')$.

Backjumping. Our tableau decision procedure for \mathcal{ALC} , upon discovering a clash, backtracks to the last branching point to which R_{\sqcup} is applicable. Blindly progressing in this way may be very inefficient if the given concept causes a lot of branching but the source of unsatisfiability does not depend on this branching (cf. [13]). *Backjumping* addresses this problem by identifying the causes of clashes and it is one of the most effective optimisation techniques, along with absorption + lazy unfolding, to deal with non-determinism. It works by associating a constraint to a branching point in the tableau such that the introduction of the constraint depends on that branching point. Because of this, tableau rules are equipped with dependency propagating information [13]. When a clash $\{x : A, x : \neg A\}$ is discovered, the dependencies of both of these constraints are combined, and the algorithm backjumps to the deepest branching point in the set. As a consequence, backjumping is an optimisation that modifies the shape of the tableau generated by the satisfiability checking algorithm by eliminating redundant branches from the tree. For this reason, no modification is required in our interpolant calculation rules.

6 Abduction

In our framework, finding the rewriting of a query in terms of the DBox predicates is possible only if the query is implicitly definable. However, not always a given query concept is implicitly definable given a TBox. To gain definability of queries of interest, it is necessary to modify the ontology by adding axioms to the TBox such that they become implicitly definable. We propose to use techniques from the area of *abduction* to find such axioms.

[14,15] define abductive reasoning tasks in DLs. Among these, we consider *TBox abduction* problems.

Definition 5 (TBox Abduction Problem). *Let \mathcal{T} be a TBox, and let C and D be concepts such that both are satisfiable w.r.t. \mathcal{T} and $C \not\sqsubseteq_{\mathcal{T}} D$. A TBox*

abduction problem (TAP) is denoted by $\langle \mathcal{T}, C, D \rangle$. A TBox \mathcal{T}_A is a solution to a TAP $\langle \mathcal{T}, C, D \rangle$ if $\mathcal{T} \cup \mathcal{T}_A$ is satisfiable and $C \sqsubseteq_{\mathcal{T} \cup \mathcal{T}_A} D$.

We look for solutions with particular syntactic shapes.

Definition 6 (Sub-concept Solution). Let $\mathcal{P} = \langle C, D, \mathcal{T} \rangle$ be a TAP. We say that a solution \mathcal{T}_A to \mathcal{P} is a sub-concept solution to \mathcal{P} iff \mathcal{T}_A is a set of axioms of the form $E_1 \sqsubseteq \neg E_2$, where concepts E_1, E_2 are sub-formulae of C, D , or \mathcal{T} .

Since the concepts in sub-concept solutions appear in the abduction problem, we conjecture that such amendment to the TBox are more easily understandable than amendments involving arbitrary concepts. Moreover, among sub-concept solutions we identify *semantically minimal* ones – i.e., the ones that minimally change the TBox in order to obtain definability of a query. With $\text{subSol}(\mathcal{P})$ we denote the set of all sub-concept solutions to a TAP \mathcal{P} and with $\mathcal{M}(\mathcal{T})$ the set of all models of the TBox \mathcal{T} .

Definition 7. Let $\mathcal{P} = \langle C, D, \mathcal{T} \rangle$ be a TAP, let \prec be a preference order, and let $\mathcal{T}_A, \mathcal{T}_B \in \text{subSol}(\mathcal{P})$. Then $\mathcal{T}_A \prec \mathcal{T}_B$ iff $\mathcal{M}(\mathcal{T}) \setminus \mathcal{M}(\mathcal{T}_A)$ is a subset of $\mathcal{M}(\mathcal{T}) \setminus \mathcal{M}(\mathcal{T}_B)$. A sub-concept solution \mathcal{T}_A is minimal iff there is no other sub-concept solution \mathcal{T}_B such that $\mathcal{T}_A \prec \mathcal{T}_B$.

The idea behind this minimality criteria is that it favours a solution \mathcal{T}_A over \mathcal{T}_B whenever \mathcal{T}_A has more common models with the original TBox \mathcal{T} than \mathcal{T}_B . In a way, \mathcal{T}_A , when added to \mathcal{T} , changes the models of \mathcal{T} in a *minimal* way. Note that \prec is a partial order since $(\mathcal{M}(\mathcal{T}) \setminus \mathcal{M}(\mathcal{T}_A)) \subseteq (\mathcal{M}(\mathcal{T}) \setminus \mathcal{M}(\mathcal{T}_B))$ does not always hold for arbitrary two sub-concept solutions \mathcal{T}_A and \mathcal{T}_B .

7 Conclusions

In this paper we considered the basic propositionally closed DL \mathcal{ALC} . However, several languages currently in use, e.g., OWL, are based on more expressive DLs, with constructs such as inverse roles and qualified cardinality restrictions. For example, in the presence of inverse roles, standard DL tableau algorithms may propagate concepts back and forth between individuals. Therefore, proving interpolation constructively by induction over the tableau is problematic. But one can overcome this problem by adapting a calculus with analytic cut rules [16] for inverse roles. Techniques like algebraic reasoning seem promising in the case of qualified cardinality restrictions [17].

Another interesting research direction is to increase the expressiveness of the query language beyond concept expressions towards, e.g., conjunctive queries. Deciding implicit definability reduces to checking query containment under Description Logic constraints [18]. However, standard Description Logic tableau techniques are not easily extended in this direction. One would need to develop new interpolation calculation methods or use techniques for first-order logic (e.g., [9]).

References

1. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F., eds.: The Description Logic Handbook: Theory, Implementation, and Applications. In Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F., eds.: Description Logic Handbook, Cambridge University Press (2003)
2. Etzioni, O., Golden, K., Weld, D.S.: Sound and efficient closed-world reasoning for planning. *Artificial Intelligence* **89**(1–2) (1997) 113–148
3. Abiteboul, S., Duschka, O.M.: Complexity of answering queries using materialized views. In: Proc. PODS. (1998) 254–263
4. Segoufin, L., Vianu, V.: Views and queries: determinacy and rewriting. In: Proc. PODS. (2005) 49–60
5. Nash, A., Segoufin, L., Vianu, V.: Determinacy and rewriting of conjunctive queries using views: A progress report. In: Proc. ICDT. (2007) 59–73
6. Marx, M.: Queries determined by views: pack your views. In: Proc. PODS. (2007) 23–30
7. Tarski, A.: Some methodological investigations on the definability of concepts. In: *Logic, Semantics and Metamathematics: Papers from 1923 to 1938*. Clarendon Press, Oxford, UK (1956) 296–319
8. Beth, E.W.: On Padoa’s methods in the theory of definitions. *Koninklijke Nederlandse Akademie van Wetenschappen, Proceedings* **56** (1953) 330–339 also *Indagationes mathematicae*, vol. 15.
9. Craig, W.: Three uses of the herbrand-gentzen theorem in relating model theory and proof theory. *The Journal of Symbolic Logic* **22**(3) (1957) 269–285
10. Fitting, M.: *Proof Methods for Modal and Intuitionistic Logics*. Volume 169 of *Synthese Library*. D. Reidel, Dordrecht, The Netherlands (1983)
11. Rautenberg, W.: Modal tableau calculi and interpolation. *Journal of Philosophical Logic* **12**(4) (1983) 403–423
12. Schlobach, S.: Explaining subsumption by optimal interpolation. In Alferes, J.J., Leite, J.A., eds.: *JELIA*. Volume 3229 of *Lecture Notes in Computer Science*., Springer (2004) 413–425
13. Horrocks, I.: Implementation and optimization techniques. [1] 306–346
14. Colucci, S., Noia, T.D., Sciascio, E.D., Donini, F.M., Mongiello, M.: A uniform tableaux-based approach to concept abduction and contraction in aln. In: Proc. of the 16th International Workshop on Description Logics (DL’04). (2004) 104
15. Elsenbroich, C., Kutz, O., Sattler, U.: A case for abductive reasoning over ontologies. In: *OWL: Experiences and Directions*. (2006) 10–11
16. Goré, R., Nguyen, L.A.: Exptime tableaux with global caching for description logics with transitive roles, inverse roles and role hierarchies. In: *TABLEAUX ’07: Proceedings of the 16th international conference on Automated Reasoning with Analytic Tableaux and Related Methods*, Berlin, Heidelberg, Springer-Verlag (2007) 133–148
17. Haarslev, V., Timmann, M., Möller, R.: Combining tableaux and algebraic methods for reasoning with qualified number restrictions. In Goble, C.A., McGuinness, D.L., Möller, R., Patel-Schneider, P.F., eds.: *Description Logics*. Volume 49 of *CEUR Workshop Proceedings*., CEUR-WS.org (2001)
18. Calvanese, D., Giacomo, G.D., Lenzerini, M.: Conjunctive query containment and answering under description logic constraints. *ACM Transactions on Computational Logic* **9**(3) (2008)