Effective Query Rewriting with Ontologies over DBoxes

İnanç Seylan and **Enrico Franconi** and **Jos de Bruijn**
Faculty of Computer Science
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 [Baader *et al.*, 2007] (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 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* [Etzioni *et al.*, 1997] or *exact views* [Abiteboul and Duschka, 1998; Segoufin and Vianu, 2005; Nash *et al.*, 2007].

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 Marx [2007], (i) corresponds to *implicit definability* [Tarski, 1956], and can be reduced to checking entailment and (ii) corresponds to *explicit definability* [Beth, 1953]. Inspired by the results of Craig [1957], 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, $\bot$, $C \sqcup D$, 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 $(\forall R.C)^{\mathcal{I}} = \{\delta \in \Delta^{\mathcal{I}} \mid \forall \delta' \, (\langle \delta, \delta' \rangle \in R^{\mathcal{I}} \rightarrow \delta' \in C^{\mathcal{I}})\}$.

$\mathcal{I}$ 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 $\dot{\neg} C$.

With $\mathsf{sig}(C)$ (resp. $\mathsf{sig}(\mathcal{T})$) 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 $\mathsf{sig}(C, \mathcal{T})$ 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 $\mathscr{D}$ is a set of *assertions* of the forms $A(a)$ and $R(a, b)$, where $A \in \sigma_{\mathscr{D}}(C)$, $R \in \sigma_{\mathscr{D}}(R)$, and $a, b \in \sigma_{\mathscr{D}}(I)$. The *signature* $\sigma_{\mathscr{D}}$ of $\mathscr{D}$ consists of the *DBox concepts* $\sigma_{\mathscr{D}}(C) \subseteq N_C$, the *DBox roles* $\sigma_{\mathscr{D}}(R) \subseteq N_R$, and the *DBox individuals* $\sigma_{\mathscr{D}}(I) \subseteq N_I$; we call *DBox predicates* the set $\sigma_{\mathscr{D}}(P) = \sigma_{\mathscr{D}}(C) \cup \sigma_{\mathscr{D}}(R)$. The *active domain* of $\mathscr{D}$ is the set of individuals in $\sigma_{\mathscr{D}}(I)$ appearing in the assertions of $\mathscr{D}$.

An interpretation $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ is a model of $\mathscr{D}$, written $\mathcal{I} \models \mathscr{D}$, iff $a^{\mathcal{I}} = a$ for every DBox individual $a \in \sigma_{\mathscr{D}}(I)$, and for every concept (resp., role) name P in $\sigma_{\mathscr{D}}(P)$ 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 $P(u) \in \mathscr{D}$ (resp., $(u, v) \in P^{\mathcal{I}}$ iff $P(u, v) \in \mathscr{D}$). In other words, in every model of $\mathscr{D}$ 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 $\mathscr{D}$ is not fixed, but it includes all the DBox individuals in $\sigma_{\mathscr{D}}(I)$, which in turn includes the active domain of $\mathscr{D}$.

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

The *unique names assumption* (UNA) holds for a DL $\mathscr{L}$ if for any interpretation $\mathcal{I}$ considered in $\mathscr{L}$, $a^{\mathcal{I}} \neq b^{\mathcal{I}}$ 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 $\mathscr{L}$ with UNA that includes $\mathcal{ALC}$, for every DBox $\mathscr{D}$ and TBox $\mathcal{T}$ in $\mathscr{L}$ there exists a TBox $\mathcal{T}^{\mathscr{D}}$ in $\mathscr{L}$ extended with nominals with size polynomially bounded by the size of $\mathcal{T} \cup \mathscr{D}$, such that $\mathscr{D}$ is satisfiable with respect to $\mathcal{T}$ iff $\mathcal{T}^{\mathscr{D}}$ is satisfiable, and vice versa.*

Proof sketch. ($\Rightarrow$) $\mathcal{T}^{\mathscr{D}}$ is obtained from $\mathcal{T}$ by adding completion and closure axioms. For every DBox concept A add the axiom $A \equiv \{a_1\} \sqcup \cdots \sqcup \{a_n\}$, such that $A(a_1), \ldots, A(a_n)$ are the assertions involving A in $\mathscr{D}$. For every DBox role R add the axioms $\{a_i\} \sqsubseteq \exists R.\{b_j\}$, $\{a_i\} \sqsubseteq \forall R.\{b_1\} \sqcup \cdots \sqcup \{b_n\}$, and $\exists R.\top \sqsubseteq \{a_1\} \sqcup \cdots \sqcup \{a_m\}$, such that $R(a_i, b_j)$ for $i = 1, \cdots, m$ and $j = 1, \cdots, n$ are the assertions involving R in $\mathscr{D}$.
($\Leftarrow$) For every occurrence of $\{a\}$ in $\mathcal{T}^{\mathscr{D}}$, the assertion $A^a(a)$ is added to $\mathscr{D}$, where A^a is a new DBox concept name, and every occurrence of $\{a\}$ in $\mathcal{T}^{\mathscr{D}}$ 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 $\mathscr{D}$, let Q be a query concept (i.e., a query asking for all the instances of the concept) and let $\mathsf{sig}(Q, \mathcal{T}) = \{B_1, \ldots, B_m, D_1, \ldots, D_n\}$ be the combined signature where $\{D_1, \ldots, D_n\} \subseteq \sigma_{\mathscr{D}}(P)$ and $\{B_1, \ldots, B_m\} \cap \sigma_{\mathscr{D}}(P) = \emptyset$; $\{D_1, \ldots, D_n\}$ 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 $B_1, \ldots, B_m$ replaced by distinct occurrences of $B'_1, \ldots, B'_m \notin \mathsf{sig}(Q, \mathcal{T})$. Then, Q is* implicitly definable *from $D_1, \ldots, D_n$ 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 $D_1, \ldots, D_n$ in $\mathcal{T}$ iff there is some concept C such that $Q \equiv_{\mathcal{T}} C$ and $\mathsf{sig}(C) \subseteq \{D_1, \ldots, D_n\}$.*

Clearly, explicit definability implies implicit definability. Beth [1953] shows that the converse also holds for the case of first-order logic: if Q is implicitly definable from $D_1, \ldots, D_n$ 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 $\mathsf{sig}(C) \subseteq \mathsf{sig}(Q, \mathcal{T}) \cap \mathsf{sig}(Q', \mathcal{T}')$, $Q \sqsubseteq_{\mathcal{T}\cup\mathcal{T}'} C$, and $C \sqsubseteq_{\mathcal{T}\cup\mathcal{T}'} Q'$.*

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

Proof. We have that $Q \equiv_{\mathcal{T}\cup\mathcal{T}'} Q'$. Now, by Lemma 1 there is an interpolant C of Q and Q' under $\mathcal{T}$ and $\mathcal{T}'$. Since it is an interpolant, $\mathsf{sig}(C) \subseteq \mathsf{sig}(Q, \mathcal{T}) \cap \mathsf{sig}(Q', \mathcal{T}') \subseteq \{D_1, \ldots, D_n\}$, and both (a) $Q \sqsubseteq_{\mathcal{T}\cup\mathcal{T}'} C$ and (b) $C \sqsubseteq_{\mathcal{T}\cup\mathcal{T}'} Q'$. By (a) and $Q' \sqsubseteq_{\mathcal{T}\cup\mathcal{T}'} Q$, we have $C \sqsubseteq_{\mathcal{T}\cup\mathcal{T}'} Q$, from which $Q \equiv_{\mathcal{T}\cup\mathcal{T}'} C$ follows by (b). From the structure of $\mathcal{T}'$ and the fact that $\mathsf{sig}(Q), \mathsf{sig}(C) \subseteq \mathsf{sig}(\mathcal{T})$ straightforwardly follows that $Q \equiv_{\mathcal{T}} C$. □

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

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.*

Proof sketch. Let $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ and $\mathcal{I}' = \langle \Delta^{\mathcal{I}'}, \cdot^{\mathcal{I}'} \rangle$ be two interpretations as in Definition 3, and let δ be in both $\Delta^{\mathcal{I}}$ and $\Delta^{\mathcal{I}'}$. We will show by induction on the structure of concept D in negation normal form that $\delta \in D^{\mathcal{I}}$ implies $\delta \in D^{\mathcal{I}'}$.

The atomic cases are trivial: if $D = A$ or $D = \neg A$, for some atomic concept A, clearly $\delta \in D^{\mathcal{I}}$ iff $\delta \in D^{\mathcal{I}'}$, since $A^{\mathcal{I}} = A^{\mathcal{I}'}$. Suppose now $D = C_1 \sqcap C_2$. By the inductive hypothesis, $\delta \in (C_1)^{\mathcal{I}}$ implies $\delta \in (C_1)^{\mathcal{I}'}$, and $\delta \in (C_2)^{\mathcal{I}}$ implies $\delta \in (C_2)^{\mathcal{I}'}$. But then $\delta \in (C_1 \sqcap C_2)^{\mathcal{I}}$ implies $\delta \in (C_1 \sqcap C_2)^{\mathcal{I}'}$. Analogous for when $D = C_1 \sqcup C_2$. Suppose $D = \exists R.E$. Since $R^{\mathcal{I}} = R^{\mathcal{I}'}$, there is some $\delta' \in \Delta^{\mathcal{I}}$ such that $\langle \delta, \delta' \rangle \in R^{\mathcal{I}}$ iff $\delta' \in \Delta^{\mathcal{I}'}$ and $\langle \delta, \delta' \rangle \in R^{\mathcal{I}'}$. Moreover, by the induction hypothesis we have that $\delta' \in E^{\mathcal{I}}$ implies $\delta' \in E^{\mathcal{I}'}$. Analogous for $D = \forall R.E$. □

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

Given a query concept Q, a TBox $\mathcal{T}$, and a DBox $\mathscr{D}$, in order to find the answer set $\{a \mid (\mathcal{T}, \mathscr{D}) \models Q(a)\}$ by means of the rewriting $\{a \mid \mathscr{D} \models C(a)\}$, one can use the equivalent database query $Q(x) \text{ :- } C^{\mathsf{FO}}(x) \wedge \top_{\mathscr{D}}(x)$ over the database $\mathscr{D}$, where C^{FO} is the standard first order translation of C and $\top_{\mathscr{D}}$ is the relation containing exactly the DBox individuals $\sigma_{\mathscr{D}(I)}$. 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 $\mathcal{ALC}$

Fitting [1983] and Rautenberg [1983] give constructive proofs using symmetric Gentzen systems, respectively tableau calculi. Schlobach [2004] 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.

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 $succ(x, R, S)$ 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, $\{(x : A)^{\lambda}, (x : \neg A)^{\kappa}\} \subseteq S$.

The $\mathsf{R}_\sqcap$ 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 $\mathsf{R}_\sqcup$ 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 $\mathsf{R}_\exists$ rule
Condition: $(x : \exists R.C)^\lambda \in L(g)$, there is no variable y in $L(g)$ such that $y : \{C^\lambda\} \cup succ(x, R, S) \subseteq L(g)$
Effect: $L(g') = L(g) \cup y : \{C^\lambda\} \cup succ(x, R, S) \cup \mathcal{T}$, where y is fresh for S

Figure 1: Biased completion rules for $\mathcal{ALC}$

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_χ 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 $L(\langle g, g' \rangle) = \mathsf{R}_\sqcup$, 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: $\mathsf{R}_\exists$ is applied only when $\mathsf{R}_\sqcup$ is not applicable which in turn is applied only when $\mathsf{R}_\sqcap$ 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.

Definition 4. *Let g be a node in a tableau and let $\{(x : C_1)^l, \ldots, (x : C_n)^l, (x : D_1)^r, \ldots, (x : D_m)^r\}$ 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 $C_1 \sqcap \ldots \sqcap C_n$ and $\neg D_1 \sqcup \ldots \sqcup \neg D_m$ under $\mathcal{T}^l$ and $\mathcal{T}^r$. (Take the empty intersection to be $\top$ and the empty union to be $\bot$.)*

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

Interpolants at nodes g are stored in hash tables $int(g)$.

$$\mathsf{C}_\neg(ll) \quad \frac{(x : A)^l, (x : \neg A)^l \in L(g)}{int(g)(x) := \bot}$$

$$\mathsf{C}_\neg(rr) \quad \frac{(x : A)^r, (x : \neg A)^r \in L(g)}{int(g)(x) := \top}$$

$$\mathsf{C}_\neg(lr) \quad \frac{(x : A)^l, (x : \neg A)^r \in L(g)}{int(g)(x) := A}$$

$$\mathsf{C}_\neg(rl) \quad \frac{(x : A)^r, (x : \neg A)^l \in L(g)}{int(g)(x) := \neg A}$$

Figure 2: Atomic interpolant calculation rules for $\mathcal{ALC}$

$$\mathsf{C}_\sqcap(\lambda) \quad \frac{(x : C_1 \sqcap C_2)^\lambda \in L(g) \quad L(g') = L(g) \cup \{(x : C_1)^\lambda, (x : C_2)^\lambda\}}{int(g) := int(g')}$$

$$\mathsf{C}_\sqcup(l) \quad \frac{(x : C_1 \sqcup C_2)^l \in L(g) \quad L(g') = L(g) \cup \{(x : C_1)^l\} \quad L(g'') = L(g) \cup \{(x : C_2)^l\}}{int(g) := int(g') \sqcup int(g'')}$$

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

Figure 3: Propositional interpolant calculation rules for $\mathcal{ALC}$

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 $\mathsf{C}_\chi(\lambda)$ 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.

We need to show that our interpolant calculation algorithm is correct and that it terminates. The latter is a consequence of the complete tableau being finite, which in turn is a consequence of our tableau calculus being a decision procedure for $\mathcal{ALC}$ [Baader *et al.*, 2007]. We show correctness in two steps: first we prove that our interpolant calculation rules are sound, i.e., they compute interpolants; second we prove completeness, i.e., that we always find an interpolant.

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 $L(g)$.*

Proof sketch. We proceed by induction on the tableau tree. For each leaf node g, $L(g)$ must contain a clash involving some variable x. Assume $(x : A)^l, (x : \neg A)^r \in L(g)$, then clearly $\cdots \sqcap A \sqsubseteq_\mathcal{T} A \sqsubseteq_\mathcal{T} \cdots \sqcup A$, and so $int(g)(x) = A$ is an interpolant; similar for the other atomic rules.

For reasons of space we further only show soundness of $\mathsf{C}_\exists(l)$. We have that y is not in $L(g)$. For any other variable $z \neq x$, we have that $int(g)(z)$ is an interpolant, by induction. If $int(g')(y) = \mathsf{null}$, $int(g)(x)$ must be an interpolant (if it is not null): if $X \sqsubseteq_\mathcal{T} int(g)(x)$, then clearly $X \sqcap \exists R.C \sqsubseteq_\mathcal{T} int(g)(x)$. Now assume $int(g')(x) = \mathsf{null}$ and $int(g')(y) \neq \bot$. If $(y : D)^\lambda \in L(g')$ for some concept $D \neq C$, either $(x : \forall R.D)^\lambda \in L(g)$ or $(D)^\lambda \in \mathcal{T}$.

$$\mathsf{C}_{\exists}(l)\quad \frac{(x:\exists R.C)^l \in L(g) \qquad L(g') = L(g) \cup \big(y : \{C^l\} \cup X\big)}{\begin{array}{ll} int(g)(z) := int(g')(z) & (z \neq x \;\&\; z \neq y) \\ int(g)(y) := \mathsf{null} & I := int(g')(y) \\ int(g)(x) := int(g')(x) & [\text{if } I = \mathsf{null}] \\ int(g)(x) := int(g')(x) \sqcup \bot & [\text{if } I = \bot] \\ int(g)(x) := int(g')(x) \sqcup \exists R.I & [\text{otherwise}] \end{array}}$$

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

Figure 4: Modal interpolant calculation rules for $\mathcal{ALC}$

In the former case we have that $C \sqcap D \sqsubseteq_{\mathcal{T}} int(g)(y)$ implies $\exists R.C \sqcap \forall R.D \sqsubseteq_{\mathcal{T}} \exists int(g)(x)$ and $int(g')(y) \sqsubseteq_{\mathcal{T}} \neg D$ implies $\exists R.int(g')(y) \sqsubseteq_{\mathcal{T}} \neg\forall R.D$. For the latter case, observe that $\forall R.D \equiv_{\mathcal{T}} \top$. The cases $int(g')(x) \neq \mathsf{null}$ and/or $int(g')(y) = \bot$ can be shown analogously. □

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,* $int(g_0)(x_0) \neq \mathsf{null}$.

Proof sketch. Since our biased tableau algorithm is sound and complete, $H \sqsubseteq_{\mathcal{T}} J$ iff there exists a closed biased tableau $\mathbf{T} = \langle V, E, L\rangle$ for $\langle H \sqcap \dot{\neg} J, \mathcal{T}\rangle$. Since $\mathbf{T}$ is closed, all labels of its leaves contain clashes. Let g be such a leaf. Then, there is some y in $L(g)$ such that $\{y : A, y : \neg A\} \subseteq L(g)$ and y is introduced by a (possibly empty) series of $\mathsf{R}_{\exists}$ applications. Clearly, $int(g)(y) \neq \mathsf{null}$. One can verify by induction on the rule applications that $int(g_0)(x_0) \neq \mathsf{null}$. □

We are now ready to prove interpolation for $\mathcal{ALC}$ with general TBoxes.

Proof of Lemma 1. Let $\mathcal{T}^l = \{C^l \mid C \in \mathcal{T}\}$ and $\mathcal{T}^r = \{C^r \mid C \in \mathcal{T}'\}$. By Lemma 3 we have that $int(g_0)(x_0)$ is an interpolant for the x_0-constraints of $L(g_0)$ under $\mathcal{T}^l \cup \mathcal{T}^r$. But $L(g_0) = x_0 : \{H^l\} \cup \{(\dot{\neg} J)^r\} \cup \mathcal{T}^l \cup \mathcal{T}^r$ and so $int(g_0)(x_0)$ is an interpolant for H and J under $\mathcal{T}$ and $\mathcal{T}'$. □

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 $\mathsf{C}_{\sqcup}(\lambda)$ 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 $\mathcal{ALC}$.

$$\mathsf{C}_{\dot{\neg}}(ll)\quad \frac{(x:C)^l, (x:\dot{\neg}C)^l \in L(g)}{int(g)(x) := \bot}$$

$$\mathsf{C}_{\dot{\neg}}(rr)\quad \frac{(x:C)^r, (x:\dot{\neg}C)^r \in L(g)}{int(g)(x) := \top}$$

$$\mathsf{C}_{\dot{\neg}}(lr)\quad \frac{(x:C)^l, (x:\dot{\neg}C)^r \in L(g)}{int(g)(x) := C}$$

Figure 5: Interpolant calculation rules for non-atomic clashes

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.

Semantic Branching. The standard definition of $\mathsf{R}_{\sqcup}$ (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 [Horrocks, 2003]. This requires a slight change in the $\mathsf{R}_{\sqcup}$ rule: the second effect becomes $L(g'') = L(g) \cup \{x : \dot{\neg}C_1, x : C_2\}$, obtaining the new rule $\mathsf{R}_{\sqcup'}$. Replacing $\mathsf{R}_{\sqcup}$ with $\mathsf{R}'_{\sqcup}$ in our tableau calculus requires modifying the third condition of $\mathsf{C}_{\sqcup}(\lambda)$ 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 [Horrocks, 2003]. 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 $\mathcal{T}_U$ [Horrocks, 2003] 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 $\mathcal{T}'$. $\mathsf{R}_{\exists}$ 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 [Horrocks, 2003].

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 $\mathsf{C}_{U1}(l)$ for $\mathcal{T}_U$. Suppose $X = \{(x : C_1)^l, \ldots, (x : C_n)^l, (x : D_1)^r, \ldots, (x : D_m)^r\}$, 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 $L(g')$.

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

Figure 6: Lazy unfolding rules

Note that although κ may not be equal to l, we can assume w.l.o.g. that $\kappa = l$ since $A \in \mathsf{sig}(\mathcal{T}_U)$. By our assumption, $int(g')(x)$ is an interpolant for the x-constraints of $L(g')$ and by Definition 4, $A \sqcap C \sqcap C_1 \sqcap \ldots \sqcap C_n \sqsubseteq_{\mathcal{T}} int(g')(x) \sqsubseteq_{\mathcal{T}} \neg D_1 \sqcup \ldots \sqcup \neg D_m$. C_{U1} assigns $int(g')(x)$ to $int(g)(x)$. We have that (i) $A \sqcap C_1 \sqcap \ldots \sqcap C_n \sqsubseteq_{\mathcal{T}} int(g')(x)$ follows from $(A \sqcap C \sqcap C_1 \sqcap \ldots \sqcap C_n)^{\mathcal{I}} = (A \sqcap C_1 \sqcap \ldots \sqcap C_n)^{\mathcal{I}}$, for all models $\mathcal{I}$ of $\mathcal{T}$, since $A \equiv C \in \mathcal{T}_U$ and (ii) $int(g')(x) \sqsubseteq_{\mathcal{T}} \neg D_1 \sqcup \ldots \sqcup \neg D_m$ 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 $\mathsf{R}_{\sqcup}$ 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. [Horrocks, 2003]). *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 [Horrocks, 2003]. 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.

[Colucci *et al.*, 2004; Elsenbroich *et al.*, 2006] define abductive reasoning tasks in DLs. Among these, we consider

$$\mathsf{C}_{U1}(\lambda)\quad \frac{(x : A)^{\kappa} \in L(g) \quad A \equiv C \in \mathcal{T}_U (\text{or } \mathcal{T}'_U) \quad L(g') = L(g) \cup \{(x : C)^{\lambda}\}}{int(g) := int(g')}$$

$$\mathsf{C}_{U2}(\lambda)\quad \frac{(x : \neg A)^{\kappa} \in L(g) \quad (A \equiv C)^{\lambda} \in \mathcal{T}_U (\text{or } \mathcal{T}'_U) \quad L(g') = L(g) \cup \{(x : \doteq C)^{\lambda}\}}{int(g) := int(g')}$$

$$\mathsf{C}_{U3}(\lambda)\quad \frac{(x : A)^{\kappa} \in L(g) \quad (A \sqsubseteq C)^{\lambda} \in \mathcal{T}_U (\text{or } \mathcal{T}'_U) \quad L(g') = L(g) \cup \{(x : C)^{\lambda}\}}{int(g) := int(g')}$$

Figure 7: Lazy unfolding interpolant calculation rules

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 $\mathsf{subSol}(\mathcal{P})$ 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 $\mathcal{T}_A, \mathcal{T}_B \in \mathsf{subSol}(\mathcal{P})$. 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 $\mathcal{T}_B$

A thorough study of abduction in DLs is given in the master's thesis by Klarman [2008], in which the author solves the more generic problem of ABox abduction using both tableau and resolution, and proposes novel techniques. A minimality criteria similar to ours is also formulated. Since we do not consider ABoxes, our approach requires less involved techniques.

7 Outlook

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 [Goré and Nguyen, 2007] for inverse roles. Techniques like algebraic reasoning seem promising in the case of qualified cardinality restrictions [Haarslev *et al.*, 2001]. In the presence of nominals, the logic loses the interpolation property and requires additional constructs in the concept language to regain this property [Ten Cate *et al.*, 2006]. In future work we plan to extend our techniques towards such expressions Description Logics.

The high computational complexity of these DLs might make one question the practicality of such techniques. However, the schema of a database in contrast to its instance is not exposed to many changes. Hence the explicit definitions can be computed offline as SQL views once the ontology has been developed.

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 [Calvanese *et al.*, 2008]. 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., Craig [1957]).

Another direction we plan to pursue is a case study of real world ontologies to check how many queries of interest are definable in terms of a given set of DBox predicates and how hard it is to amend such ontologies to make queries definable.

Acknowledgements

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.

References

[Abiteboul and Duschka, 1998] S. Abiteboul and O. M. Duschka. Complexity of answering queries using materialized views. In *Proc. PODS*, pages 254–263, 1998.

[Baader *et al.*, 2007] F. Baader, D. Calvanese, D. L. McGuinness, D Nardi, and P. F. Patel-Schneider, editors. *Description Logic Handbook, 2nd edition*. Cambridge Univ. Press, 2007.

[Beth, 1953] E. W. Beth. On Padoa's methods in the theory of definitions. *Koninklijke Nederlandse Akademie van Wetenschappen, Proceedings*, 56:330–339, 1953.

[Calvanese *et al.*, 2008] D. Calvanese, G. De Giacomo, and M. Lenzerini. Conjunctive query containment and answering under description logic constraints. *ACM Transactions on Computational Logic*, 9(3), 2008.

[Colucci *et al.*, 2004] S. Colucci, T. Di Noia, E. Di Sciascio, F. M. Donini, and M. Mongiello. A uniform tableaux-based approach to concept abduction and contraction in $\mathcal{ALN}$. In *Proc. of DL'04*, page 104, 2004.

[Craig, 1957] William Craig. Three uses of the herbrand-gentzen theorem in relating model theory and proof theory. *The Journal of Symbolic Logic*, 22(3):269–285, 1957.

[Elsenbroich *et al.*, 2006] C. Elsenbroich, O. Kutz, and U. Sattler. A case for abductive reasoning over ontologies. In *OWL: Experiences and Directions*, pages 10–11, 2006.

[Etzioni *et al.*, 1997] O. Etzioni, K. Golden, and D. S. Weld. Sound and efficient closed-world reasoning for planning. *Artificial Intelligence*, 89(1–2):113–148, 1997.

[Fitting, 1983] M. Fitting. *Proof Methods for Modal and Intuitionistic Logics*, vol. 169 of *Synthese Library*. 1983.

[Goré and Nguyen, 2007] R. Goré and L. A. Nguyen. Exptime tableaux with global caching for description logics with transitive roles, inverse roles and role hierarchies. In *TABLEAUX '07*, pages 133–148, 2007.

[Haarslev *et al.*, 2001] V. Haarslev, M. Timmann, and R. Möller. Combining tableaux and algebraic methods for reasoning with qualified number restrictions. In *Proc. of DL'01*. 2001.

[Horrocks, 2003] I. Horrocks. Implementation and optimization techniques. In Baader et al. [2007], pages 306–346.

[Klarman, 2008] S. Klarman. Abox abduction in description logic. Master's thesis, Universiteit van Amsterdam, 2008.

[Marx, 2007] M. Marx. Queries determined by views: pack your views. In *Proc. PODS*, pages 23–30, 2007.

[Nash *et al.*, 2007] A. Nash, L. Segoufin, and V. Vianu. Determinacy and rewriting of conjunctive queries using views: A progress report. In *Proc. ICDT*, pages 59–73, 2007.

[Rautenberg, 1983] W. Rautenberg. Modal tableau calculi and interpolation. *Journal of Philosophical Logic*, 12(4):403–423, 1983.

[Schlobach, 2004] Stefan Schlobach. Explaining subsumption by optimal interpolation. In Proc. *JELIA'04*, pages 413–425. 2004.

[Segoufin and Vianu, 2005] Luc Segoufin and Victor Vianu. Views and queries: determinacy and rewriting. In *Proc. PODS*, pages 49–60, 2005.

[Tarski, 1956] A. Tarski. Some methodological investigations on the definability of concepts. In *Logic, Semantics and Metamathematics*, pages 296–319. Clarendon, 1956.

[Ten Cate *et al.*, 2006] B. ten Cate, W. Conradie, M. Marx, and Y. Venema. Definitorially complete description logics. In Proc. of *KR'06*, pages 79–89. 2006.