Non-Standard Reasoning Services for the Debugging of Description Logic Terminologies

Stefan Schlobach
Language and Inference Technology, ILLC
Universiteit van Amsterdam, Netherlands
schlobac@science.uva.nl

Ronald Cornet
Academic Medical Center
Universiteit van Amsterdam, Netherlands
R.Cornet@amc.uva.nl

Abstract

Current Description Logic reasoning systems provide only limited support for debugging logically erroneous knowledge bases. In this paper we propose new non-standard reasoning services which we designed and implemented to pinpoint logical contradictions when developing the medical terminology DICE. We provide complete algorithms for unfoldable ACC-TBoxes based on minimisation of axioms using Boolean methods for *minimal unsatisfiability-presening sub-TBoxes,* and an incomplete bottom-up method for *generalised incoherence-preserving terminologies.*

1 Introduction

Our work was motivated by the development of the DICE[1] terminology. DICE implements frame-based definitions of diagnostic information for the unambiguous and unified classification of patients in Intensive Care medicine. The representation of DICE is currently being migrated to an expressive Description Logic (henceforth DL) to facilitate logical inferences. Figure 1 shows an extract of the DICE terminology. In [Cornet and Abu-Hanna, 2002] the authors describe the migration process in more detail. The resulting DL terminology (usually called a "TBox") contains axioms such as the following, where classes (like BODYPART) are translated as concepts, and slots (like REGION) as roles:

$$Brain \doteq CentralNervousSystem \sqcap \exists systempart.NervousSystem \sqcap BodyPart \sqcap \exists region.HeadAndNeck \sqcap \forall region.HeadAndNeck$$
$$CentralNervousSystem \doteq NervousSystem$$

Developing a coherent terminology is a time-consuming and error-prone process. DICE defines more than 2400 concepts and uses 45 relations. To illustrate some of the problems, take the definition of a "brain" which is incorrectly specified, among others, as a "central nervous-system" and "body-part" located in the head. This definition is contradictory as nervous-systems and body-parts are declared disjoint in DICE. Fortunately, current Description Logic reasoners, such as RACER [Haarslev and Moller, 2001] or FaCT [Ilorrocks, 1998], can detect this type of inconsistency and the knowledge engineer can identify the cause of the problem. Unfortunately, many other concepts are defined based on the erroneous definition of "brain" forcing each of them to be erroneous as well. In practice, DL reasoners provide lists of hundreds of unsatisfiable concepts for the DICE TBox and the debugging remains a jigsaw to be solved by human experts, with little additional explanation to support this process.

By *debugging* we understand the identification and elimination of modelling errors when detecting logical contradictions in a knowledge base. Debugging requires an *explanation* for the logical incorrectness and, as a second step, its *correction.* In this paper we will focus on the former as the latter requires an understanding of the meaning of represented concepts. We define a number of new non-standard reasoning services to explain incoherences through *pinpointing.*

Our experience with debugging DICE provides some hands-on examples for the problem at hand: take the contradictory definition of brains in the DICE anatomy specification. What information is useful for correcting the knowledge base? First, we have to identify the precise position of errors within a TBox; that is, we need a procedure to single out the axioms causing the contradiction. The axioms for *Brain* and *CentralNenvusSystem* form such a minimal incoherent subset of the DICE terminology. Formally, we introduce *minimal unsatisfiability-presening sub-TBoxes* (abbreviated MUPS) and *minimal incoherence-presening sub-TBoxes* (MIPS) as the smallest subsets of axioms of an incoherent terminology preserving unsatisfiability of a particular, respectively of at least one unsatisfiable concept. Secondly, we highlight the elements of these definitions containing the faulty specification. An axiom, e.g., $Brain \doteq BodyPart \sqcap NervousSystem$ points to the core of the erroneously modelled knowledge. For this purpose we define *generalised incoherence-preserving terminologies* (GIT) as sets of incoherent axioms, which are syntactically related to the original axioms, more general and have minimal structural complexity.

The remainder of this paper is organised as follows: Section 2 introduces the relevant DL concepts to make the paper self-contained. In Section 3 we introduce the new reasoning services and provide algorithms in Section 4. The paper finishes with a discussion of the results we obtained with our new methods on the terminology DICE.

[1]DICE stands for "Diagnoses for Intensive Care Evaluation". The development of the DICE terminology has been supported by the NICE foundation.

CLASS	SUPERCLASS	SLOT	SLOT-VALUE
BRAIN	BODYPART CENTRAL NERVOUS SYSTEM	REGION SYSTEM PART	HEAD AND NECK NERVOUS SYSTEM
CENTRAL NERVOUS SYSTEM	NERVOUS SYSTEM		

Figure 1: An extract from the DICE terminology (frame-based).

2 Description Logics

We shall not give a formal introduction into Description Logics here, but point to the new handbook [Baader *et at.,* 2003]. Briefly, DLs are set description languages with concepts (usually denoted by capital letters), interpreted as subsets of a domain, and roles which are binary relations, which we denote by small letters. In a terminological component T (called TBox) the interpretations of concepts can be restricted to the *models* of *T* by defining *axioms* of the form $C \sqsubseteq D$.[2] Based on this formal model-theoretic semantics, a TBox can be checked for *incoherence,* i.e., whether there are *unsatisfiable* concepts: concepts which are necessarily interpreted as the empty set in all models of the TBox. Other standard reasoning services include *subsumption* of two concepts *C* and *D* (a subset relation w.r.t. all models of T). Subsumption without reference to a TBox is called *concept subsumption* and we write $C \sqsubseteq D$. Recently, a number of non-standard reasoning services have been defined, such as the *least common subsumer* of concepts or the minimal *rewriting* of concepts (see [Kiisters, 2001] for an overview).

ACC is a simple yet relatively expressive DL with conjunction $(C \sqcap D)$, disjunction $(C \sqcup D)$, negation $(\neg C)$ and universal $(\forall r.C)$ and existential quantification $(\exists r.C)$. A TBox is called *unfoldable* if the left-hand sides of the axioms (the defined concepts) are atomic, and if the right-hand sides (the definitions) contain no direct or indirect reference to the defined concept [Nebel, 1990].

3 Explaining Logical Incoherences

In this section we study ways *of explaining incoherences* in DL terminologies. We propose to simplify a terminology *T* in order to reduce the available information to the root of the incoherence. More concretely we first exclude axioms which are irrelevant to the incoherence and then provide simplified definitions highlighting the exact position of a contradiction within the axioms of this reduced TBox. We will call the former *axiom pinpointing,* the latter *concept pinpointing.*

In this section we will formally introduce axiom and concept pinpointing for a general TBox without restrictions on the underlying representation language. In Section 4 we will give algorithms for the case of unfoldable *ACC-TBoxes.*

3.1 Axiom Pinpointing

Axiom pinpointing means identifying debugging-relevant axioms, where an axiom is *relevant* if a contradictory TBox becomes coherent once the axiom is removed or if, at least, a particular, previously unsatisfiable concept turns satisfiable.

[2] We will not consider assertional components in this paper.

Consider the following (incoherent) TBox 71, where *A,B* and *C* are primitive and *A*\,..., *A7* defined concept names:

$$\begin{aligned} A_1 &\sqsubseteq A_2 \sqcap A_3 \\ A_2 &\sqsubseteq A_5 \\ A_3 &\sqsubseteq \exists s.\neg B \\ A_4 &\sqsubseteq A \\ A_5 &\sqsubseteq \neg A \\ A_6 &\sqsubseteq B \\ A_7 &\sqsubseteq C \end{aligned}$$

The set of unsatisfiable concept names as returned by a DL reasoner is $\{A_1,A_3,A_6,A_7\}$. Although this is still of manageable size, it hides crucial information, e.g., that unsatisfiability of A_1 depends on unsatisfiability of A_3, which is incoherent because of the contradictions between A_4 and A_5. We will use this example to explain our debugging methods.

Minimal unsatisfiability-preserv ing sub-TBoxes (MUPS)
Unsatisfiability-prcserving sub-TBoxes of a TBox *T* and an unsatisfiable concept *A* are subsets of *T* in which *A* is unsatisfiable. In general there are several of these sub-TBoxes and we select the minimal ones, i.e., those containing only axioms that are necessary to preserve unsatisfiability.

Definition 3.1 Let *A* be a concept which is unsatisfiable in a TBox *T.* A set $T' \subseteq T$ is a *minimal unsatisfiability-preserving sub-TBox (MUPS)* of *T* if *A* is unsatisfiable in 7~\ and *A* is satisfiable in every sub-TBox $T'' \subset V$•

We will abbreviate the set of MUPS of *T* and *A* by *mups(T, A).* MUPS for our example TBox T_1 and its unsatisfiable concepts are:

$$\begin{aligned} mups(\mathcal{T}_1, A_1) &= \{\{ax_1, ax_2\}, \{ax_1, ax_3, ax_4, ax_5\}\} \\ &= \{\{ax_3, ax_4, ax_5\}\} \\ mups(\mathcal{T}_1, A_6) &= \{\{ax_1, ax_2, ax_4, ax_6\}, \\ &\quad \{ax_1, ax_3, ax_4, ax_5, ax_6\}\} \\ mups(\mathcal{T}_1, A_7) &= \{\{ax_4, ax_7\}\} \end{aligned}$$

MUPS arc useful for relating unsatisfiability to sets of axioms but we will also use them in Section 4 to calculate MIPS.

Minimal incoherence-preserving sub-TBoxes (MIPS)
MIPS are the smallest subsets of an original TBox preserving unsatisfiability of at least one atomic concept.

Definition 3.2 Let *T* be an incoherent TBox. A TBox $T' \subseteq T$ is a *minimal incoherence-preserving sub-TBox (MIPS) of T* if T' is incoherent, and every sub-TBox $T'' \subset T'$ is coherent.

We will abbreviate the set of MIPS of *T* by *mips{T).* For 71 we get three MIPS:

$$mips(\mathcal{T}_1) = \{\{ax_1, ax_2\}, \{ax_3, ax_4, ax_5\}, \{ax_4, ax_7\}\}$$

It can easily be checked that each of the three incoherent TBoxes in *mips(Ti)* is indeed a MIPS as taking away a single axiom renders each of the three coherent. The first one signifies, for example, that the first two axioms are already contradictory without reference to any other axiom, which suggests a modelling error already in these two axioms.

Cores

Minimal incoherence-preserving sub-TBoxes identify smallest sets of TBox axioms causing the original TBox to be incoherent. In terminologies such as DICE, which are created through migration from other representation formalisms, there are several such sub-TBoxes, each corresponding to a particular contradictory terminology. *Cores* are now sets of axioms occurring in several of these incoherent TBoxes. The more MIPS such a core belongs to, the more likely its axioms will be the cause of contradictions.

Definition 3.3 Let *T* be a TBox. A non-empty intersection of *n* different MIPS in *rnips(T)* (with $n \geq 1$) is called a *MIPS-core ofarity n* (or simply *n-ary core)* for T.

livery set containing precisely one MIPS is, at least, a 1-ary core. The most interesting cores of a TBox, T, are those with axioms that are present in as many MIPS of *T* as possible, i.e., having maximal arity. On the other hand, the size of a core is also significant, as a bigger size points to clusters of axioms causing contradictions in combination only.

In our running example, axiom ax_4 occurs both in $\{ax_3, ax_4, ax_5\}$ and $\{ax_4, ax_7\}$, which makes $\{ax_4\}$ a core of arity 2 for T_1, which is the core of maximal arity in this example.

3.2 Concept Pinpointing

The next step in the debugging process is to simplify the definitions in order to obtain more concise descriptions of errors within an incoherent TBox.

Generalised Terminologies

Incoherence of a TBox can be regarded as an overspecification of one or more concepts in the relevant definitions. *Generalised terminologies* are terminologies where some of the definitions have been generalised.[3] Furthermore, we require generalised definitions to be *syntactically related* to the original axioms. For different representation languages and types of knowledge bases we will have to formally specify what we consider to be syntactically related concepts. For the time being let us keep the definition abstract and assume that there is such a relation *rel (C,* C') denoting that two concepts *C* and C' are related. Formally, a concept C" is then a *syntactic generalisation* of a concept *C* if *rel{C, C)* and $C \sqsubseteq C'$ (independent of T). Note that we have to take care of the case when *C* is unsatisfiable w.r.t. T. Then *C* is equivalent to $\bot$, and everything is more general. Therefore we generalise *C* without referencing to the original terminology T, i.e., we check for simple concept subsumption, only.

Now, generalised incoherence-preserving terminologies (GITs) are TBoxes where the defining concepts of the axioms are maximally generalised without losing incoherence.

Definition 3.4 L$\mathcal{T} = \{C_1 \dot{\sqsubseteq} D_1, \ldots, C_n \dot{\sqsubseteq} D_n\}$ be an incoherent TBox. An **incoherent TBox** $\mathcal{T}' = \{C_1 \dot{\sqsubseteq} D_1', \ldots, C_n \dot{\sqsubseteq} D_n'\}$ is a *generalised incoherence-preserving terminology (GIT)* of T if, and only if,

- each D_1 is a syntactic generalisation of D_i $(1 \leq i \leq n)$,
- every TBox T" = $\{C_1 \dot{\sqsubseteq} D_1'', \ldots, C_n \dot{\sqsubseteq} D_n''\}$ with a syntactic generalisation D'' of D_t where $D_i \dot{\sqsubseteq} D_i$ and $D_i'' \not\sqsubseteq D_i'$ (for some $1 \leq i \leq n$) is coherent.

We abbreviate the set of generalised incoherence-preserving terminologies of a TBox *T* by *git(T).* Note that the set of GITs of a TBox *T* is equivalent to the union of GITs for the MIPS of T. Take a simple syntactic relation relating a concept with the syntactic sub-concepts of its unfolded version with the same polarity and quantifier depth.[4] As this particular definition of a syntactic relation depends on unfolding the related concepts, all but one axiom per GIT can be trivially generalised to $A_i \dot{\sqsubseteq} \top$. Of all possible GITs we conjecture that the simplest ones, e.g., those with *syntactically minimal* generalisations, are most likely to be useful for the identification of errors. In our experiments we use two alternative formalisations, the first with respect to minimal size of axioms, the second with respect to the number of concept names occurring in the GIT. Three minimal sized GITs exist for our example TBox 71, where we only show the non-trivial axioms:

$$git(\mathcal{T}_1) = \{\{A_1 \dot{\sqsubseteq} \neg A \sqcap A\}, \{A_3 \dot{\sqsubseteq} \forall s.B \sqcap \exists s.\neg B\}, \{A_7 \dot{\sqsubseteq} \forall s.B \sqcap \exists s.\neg B\}\}.$$

4 Debugging Unfoldable ALC-TBoxes

Practical experience has shown that applying our methods on a simplified version of DICE can already provide valuable debugging information. We will therefore only provide algorithms for unfoldable ,ALC-TBoxes [Nebel, 1990] as this significantly improves both the computational properties and the readability of the algorithm.

4.1 Algorithms for Axiom Pointing

The calculation of MIPS depends on the MUPS only, and we will provide an algorithm to calculate these minimal unsatisfiability-preserving sub-TBoxes based on Boolean minimisation of terminological axioms needed to close a standard tableau ([Baader *et al.,* 2003] Chapter 2).

Usually, unsatisfiability of a concept is detected with a fully saturated tableau (expanded with rules similar to those in Figure 2) where all branches contain a contradiction (or close, as we say). The information which axioms are relevant for the closure is contained in a simple label which is added to each formula in a branch. A *labelled formula* has the form $(a : C)^x$ where *a* is an individual name, *C* a concept and *x* a set of axioms, which we will refer to as *label. A* labelled branch is a set of labelled formulas and a tableau is a set of labelled branches. A formula can occur with different labels on the same branch. A branch is closed if it contains a clash, i.e. if there is at least one pair of formulas with contradictory atoms on the same individual. The notions of open branch and closed and open tableau are defined as usual and do not depend on the labels. We will always assume that any

[3] To simplify matters we generalise the right-hand side of axioms only as we are currently working with unfoldable TBoxes anyway.

[4] Thc polarity of a concept name *A* specifies whether *A* occurs within odd or even numbers of negations, the quantifier depth is the sequence of roles over which the concept is quantified. Such a relation will formally be defined for unfoldable *ACC-TBoxes* in Definition 4.3.

($\sqcap$):	**if**	$(a : C_1 \sqcap C_2)^{label} \in B$, but not both $(a : C_1)^{label} \in B$ and $(a : C_2)^{label} \in B$
	then	$B' := B \cup \{(a : C_1)^{label}, (a : C_2)^{label}\}$.
($\sqcup$):	**if**	$(a : C_1 \sqcup C_2)^{label} \in B$, but neither $(a : C_1)^{label} \in B$ nor $(a : C_2)^{label} \in B$
	then	$B' := B \cup \{(a : C_1)^{label}\}$ and $B'' := B \cup \{(a : C_2)^{label}\}$.
(Ax)	**if**	$(a : A)^{label} \in B$ and $(A \dot{\sqsubseteq} C) \in \mathcal{T}$
	then	$B' := B \cup \{(a : C)^{label \cup \{A \dot{\sqsubseteq} C\}}\}$.
($\exists$):	**if**	$(a : \exists R_i.C)^{label} \in B$, $R_i \in N_R$ and all other rules have been applied on all formulas over a, and if $\{(a : \forall R_1.C_1)^{label_1}, \ldots, (a : \forall R_n.C_n)^{label_n}\} \subseteq B$ is the set of universal formulas for a w.r.t. R_i in B,
	then	$B' := \{(b : C)^{label}, (b : C_1)^{label_1 \cup label}, \ldots, (b : C_n)^{label_n \cup label}\}$ where b is a new individual name not occurring in B.

Figure 2: Tableau Rules for $\mathcal{ALC}$-Satisfiability w.r.t. a TBox $\mathcal{T}$ (with Labels)

formula is in *negation normal form* (nnf) and newly created formulas are immediately transformed. We usually omit the prefix "labelled".

To calculate a minimal unsatisfiability-preserving TBox for a concept name *A* w.r.t. an unfoldable TBox *T* we construct a tableau from a branch *B* initially containing only $(a : A)^{\emptyset}$ (for a new individual name *a)* by applying the rules in Figure 2 as long as possible. The rules are standard ALC-tableau rules with lazy unfolding, and have to be read as follows: assume that there is a tableau T = {B, $B_1, \ldots, B_n$} with n+1 branches. Application of one of the rules on *B* yields the t aau $T' := \{B', B_1, \ldots, B_n\}$ fthe ($\sqcap$), ($\exists$) and *{Ax)*-rule, $T'' := \{B', B'', B_1, \ldots, B_n\}$ in case of th ($\sqcup$) u l e .

Once no more rules can be applied, we know which atoms are needed to close a saturated branch and can construct a minimisation function for *A* and *T* according to the rules in Figure 3. A propositional formula φ is called a *minimisation function for A and T* if *A* is unsatisfiable in every subset of *T* containing the axioms which are true in an assignment making φ true. In our case axioms are used as propositional variables in φ. As we can identify unsatisfiability of *A* w.r.t. a set *S* of axioms with a closed tableau using only the axioms in *S* for unfolding, branching on a disjunctive rule implies that we need to join the functions of the appropriate sub-branches conjunctively. If an existential rule has been applied, the new branch *B'* might not necessarily be closed on formulas for both individuals. Assume that *B'* closes on the individual *a* but not on *b.* In this case *min-function(a, B,T)* = $\bot$, which means that the related disjunct does not influence the calculation of the minimal incoherent TBox.

Based on the minimisation function *minfunction(a, {(a :* $A)^{\emptyset}\}, \mathcal{T}$) (let us call it φ) which we calculated using the rules in Figure 3 we can now calculate the MUPS for *A* w.r.t. T. The idea is to use prime implicants of φ. A prime implicant $ax_1 \wedge \ldots \wedge ax_n$ is the smallest conjunction of literals[5] implying φ [Quine, 1952]. As φ is a minimisation function every implicant of φ must be a minimisation function as well and therefore also the prime implicant. But this implies that the concept *A* must be unsatisfiable w.r.t. the set of axioms $\{ax_1, \ldots, ax_n\}$. As $ax_1 \wedge \ldots \wedge ax_n$ is the smallest implicant **we also know that** $\{ax_1, \ldots, ax_n\}$ must be minimal, i.e. a MUPS. Theorem 4.1 captures this result formally.

[5]Note that in our case all literals are non-negated axioms.

Theorem 4.1 *Let A be a concept name, which is unsatisfiable w.r.t. an unfoldable ACC-TBox* T. *The set of prime implicants of the minimisation function minJunction(a,{(a :* $A)^{\emptyset}\}$, T) *is the set mups(T, A) of minimal unsatisfiability-preserving sub-TBoxes of A and* T.

Proof: We first prove the claim that the propositional formula φ := *min function*$(a, \{(a : A)^{\emptyset}\}$, T) is indeed a minimisation function for the MUPS problem w.r.t. an unsatisfiablc concept *A* and a TBox *T.* We show that a tableau starting on a single branch $B := \{(a : A)^{\emptyset}\}$ closes on all branches by unfolding axioms only, that are evaluated as true in an assignment making φ true. This saturated tableau *Tab** is a particular sub-tableau of the original saturated tableau Tab which we used to calculate *min-function*$(a, \{(a : A)^{\emptyset}\}$, T), and it is this connection that we make use of to prove our first claim. Every branch in the new tableau is a subset of a branch occurring in the original one and we define *visible formulas* as those labelled formulas occurring in both tableaux. By induction over the rules applied to saturate *Tab* we can then show that each branch in the original tableau closes on at least one pair of visible formulas. If *A* is unsatisfiable w.r.t. T, the tableau starting with the branch $\{(a : A)^{\emptyset}\}$ closes w.r.t. T. As we have shown that this tableau closes w.r.t. *T* on visible formulas, it follows that *Tab** is closed on all branches, which proves the first claim. By another induction over the application of the rules in Figure 3 we can prove that φ is a *maximal* minimisation function, which means that $\psi \rightarrow \varphi$ for every minimisation function ψ. This proves the first part of the proof; the first claim (and the argument from above) implies that every implicant of a minimisation function identifies an unsatisfiability-preserving TBox, and maximality implies that prime implicants identify the minimal ones.
To show that the conjunction of every MUPS $\{ax_1, \ldots, ax_n\}$ is a prime implicant of *min function*$(a, \{(a : A)^{\emptyset}\}, \mathcal{T})$ **is** trivial $ax_1 \wedge \ldots \wedge ax_n$ is a minimisation function by definition. But as we know that *min function*$(a, \{(a : A)^{\emptyset}\}, \mathcal{T})$ is maximal we know that $ax_1 \wedge \ldots \wedge ax_n \rightarrow$ *min function*$(a, \{(a : A)^{\emptyset}\}, \mathcal{T})$ which implies **that** $ax_1 \wedge \ldots \wedge ax_n$ **must be prime** as otherwise $\{ax_1, \ldots, ax_n\}$ would not be minimal. ■

Satisfiability in *ACC* is PSPACE-complete, and calculating MUPS does not increase the complexity as we can construct the minimisation function in a depth-first way, allowing us to keep only one single branch in memory at a time. However,

if *rule* = (⊓) has been applied to $(a : C_1 \sqcap C_2)^{label}$ and B' is the new branch **return** *min_function*$(a, B', \mathcal{T})$;
if *rule* = (⊔) has been applied to $(a : C_1 \sqcup C_2)^{label}$ and B' and B'' are the new branches
return *min_function*$(a, B', \mathcal{T}) \wedge$ *min_function*$(a, B'', \mathcal{T})$;
if *rule* = (∃) has been applied to $(a : \exists R.C)^{label}$, B' is the new branch and b the new variable
return *min_function*$(a, B', \mathcal{T}) \vee$ *min_function*$(b, B', \mathcal{T})$;
if *rule* = (∧x) has been applied and B' is the new branch **return** *min_function*$(a, B', \mathcal{T})$;
if no further rule can be applied **return:** $\bigvee_{(a:A)^u \in B, (a:\neg A)^v \in B} (\bigwedge_{ax \in x} ax \wedge \bigwedge_{ax \in y} ax)$;

Figure 3: *min_function*$(a, B, \mathcal{T})$: Minimisation-function for the MUPS-problem

we calculate prime implicants of a minimisation function the size of which can be exponential in the number of axioms in the TBox. Therefore, approximation methods have to be considered in practice avoiding the construction of fully saturated tableaux in order to reduce the size of the minimisation functions.

From MUPS we can easily calculate MIPS, but we need an additional operation on sets of TBoxes, called *subset-reduction.* Let M = {$T_1, \ldots, T_m$} be a set of TBoxes. The *subset-reduction* of M is the smallest subset $sr(M) \subseteq M$ such that for all $T \in M$ there is a set $V \in sr(M)$ such that $'V \subseteq T$. A simple algorithm for the calculation of MIPS for T now simply follows from Theorem 4.2, which is a direct consequence of the definitions of MIPS and MUPS.

Theorem 4.2 *Let T be an incoherent TBox with unsatisfiable concepts* $\Delta^{\mathcal{T}}$. *Then,* $mips(\mathcal{T}) = sr(\bigcup_{A \in \Delta^{\mathcal{T}}} mups(\mathcal{T}, A))$

Checking elements of *mips(T)* for cores of maximal arity requires exponentially many checks in the size *of mips(T).* In practice, we therefore apply a bottom-up method searching for maximal cores of increasing size stopping once the arity of the cores is smaller than 2.

4.2 Algorithms for Concept Pinpointing

Calculating GITs depends on the definition of rel(C, *C).* In our case we will define this syntactic relation described in Footnote 4 including quantifier depth and polarity.

Definition 4.3 Let T be an unfoldable „AIC-T13ox. A concept C" is *syntactically related* to a concept *C* (notation: *rcI(C, C'))* if C" $\subseteq qs_{\mathcal{T}}$-(C), where the set *of qualified subconcepts, qsr{C),* is defined inductively:

$$qs_{\mathcal{T}}(C \sqcap D) = \{C', D', C' \sqcap D' | C' \in qs_{\mathcal{T}}(C), D' \in qs_{\mathcal{T}}(D)\}$$
$$qs_{\mathcal{T}}(C \sqcup D) = \{C', D', C' \sqcup D' | C' \in qs_{\mathcal{T}}(C), D' \in qs_{\mathcal{T}}(D)\}$$
$$qs_{\mathcal{T}}(\exists r.C) = qs_{\mathcal{T}}(\forall r.C) - \{\exists r.C', \forall r.C' | C' \in qs_{\mathcal{T}}(C)\}$$
$$qs_{\mathcal{T}}(\neg C) = \{\neg C' \mid C' \in qs_{\mathcal{T}}(C)\}$$
$$qs_{\mathcal{T}}(A) = \begin{cases} \{A\} \cup qs_{\mathcal{T}}(D) & \text{if } A \doteq D \in \mathcal{T} \\ \{A\} & \text{otherwise} \end{cases}$$

Moreover, $qs_{\mathcal{T}}(C)$ also contair⊥and ⊤ for all concepts *C*.

Note that we include terminological information from unfolded TBox axioms in the definition of syntactic relatedness. This choice allows us to create smaller and more concise GITs but is non-essential. As the TBox is finite and cycle-free, and as the concept size decreases in every recursion step, $qs_{\mathcal{T}}(C)$ is finite. Therefore there is only a finite set of concepts related to any axiom $A \doteq C$. A simple algorithm to calculate GITs for an incoherent unfoldable ALC-TBox *T* is therefore:

For all MIPS $M \in mips\{T)$ (with |rmp*(T)| = m)

1. Let *T'* be the TBox where all axioms $A \doteq C \notin M$ in *T* have been replaced by $A \doteq \top$.
2. Calculate different TBoxes T" where an arbitrary number of axioms $A_i \doteq C_i \in M$ have been replaced in T' by generalised axioms $A_i \doteq C_i'$, i.e., where $C_i' \in qs_{\mathcal{T}}(C_i)$ and $C_i \sqsubseteq C_i'$ for some $i \in \{1, \ldots, m\}$
3. Return those incoherent TBoxes T" with minimal syntactic generalisations according to the subsumption relation, and the size or number of concept names.

The algorithm described above is a naive algorithm which might not terminate in reasonable time on large terminologies with complex definitions as the number of syntactically related concepts for a given concept *C* is exponential in the size of *C.* A more efficient algorithm to calculate GITs is based on the fact that the syntactically related concepts can be created ordered by size starting with atomic concepts. To find GITs of minimal size we therefore apply a bottom-up strategy checking more and more complex terminologies for coherence in Step 2 stopping once we find an incoherent one.

5 Evaluation

The algorithms for axiom pinpointing have been implemented in JAVA using RACER, which provides the set of unsatisfiable concepts. We evaluated the methods on both the anatomy fragment of DICE and the full DICE terminology. The fragment defines 529 concept names, 76 of which were unsatisfiable at first.*' There arc 5 MIPS containing 2 axioms each, but no axiom occurs in all MIPS. However, there is a core of arity 3 and size 1: the axiom defining "central nervous system" as a "nervous system". Note that the definition of "nervous system" is *not* contradictory but that its *use* is erroneous, e.g., the concept *brain* should be defined as a "part of" and not as a subconcept of the concept *nervous system.* For each of the 5 MIPS minimal size GITs have been calculated, and they point to the exact position of the logical incorrectness. For the MIPS related to the error described

[6]Thc high number of unsatisfiable concepts is due to the fact that the DL terminology for DICE has been created by migration from a frame-based terminological system. In order to make the semantics as explicit as possible a very restrictive translation has been chosen to highlight as many ambiguities as possible. Moreover, many concepts were defined as sub-concepts of unsatisfiable concepts. See [Cornet and Abu-Hanna, 2002] for details.

in the introduction the minimal size GIT is simply *Brain* $\sqsubseteq$ *NervousSystem* $\sqcap$ *BodyPart* which, with the information that *BodyPart* and *NervousSystem* are disjoint concepts, identifies the erroneous specification.

The full DICE terminology defines more than 2400 concepts, of which more than 750 were unsatisfiablc given the chosen migration method. Our implemented algorithms found the MUPS for all but 7 unsatisfiable concepts. In these seven cases the algorithm failed on the calculation of the prime implicants as we use a naive method in our current implementation. Based on the correctly calculated MUPS we approximated the set of MIPS which contained more than 350 TBoxes. We implemented an iterative approach, calculating the core *Max* of size 1 with maximal arity for *mips(Tdicv),* then the same for the remaining TBoxes in *mips(Tdice)\Max* and so on. This way we identified 10 cores of size 1 with an arity of 25 to 10, covering almost half of the errors. For some of these cores we calculated GITs (by hand) which lead to the identification of a number of modelling errors. Even though the developed reasoning services are not yet fully integrated into the knowledge modelling environment, they are useful to pinpoint to the core of the logical contradiction, which is then investigated and eliminated by a domain expert.

Although the theoretical complexity of the MUPS problem is exponential in the size of the TBox the calculation of most MUPS for DICE was not problematic due to the relatively simple structure of the tableaux for the unsatisfiability proofs. Runtime analysis suggests that most CPU-time is spent by RACER identifying the unsatisfiable concepts, whereas computing MUPS usually takes less than 10 percent of the overall runtime[7].

6 Conclusion & Further Work

Explanation has been a research topic right from the first days of research in Artificial Intelligence (e.g., introducing TMS [Doyle, 1979] or Diagnosis [Reiter, 1987]) and Theorem Proving (recently [Fiedler, 2001]). Despite a significant interest in explanation of DL reasoning recently shown in the DL community (as [DIG, 2002] suggests) relatively little work has been published on the subject. One exception is [McGuinness, 1996] where the author provides explanation for subsumption and non-subsumption. Her approach, based on *explanation as proof fragments,* uses structural subsumption for CLASSIC and has been extended to AlC-tableau reasoning in [Borgida *et al,* 1999]. In contrast to this approach, our non-standard reasoning services for axiom and concept pinpointing focus on the reduction of information, and are independent of particular calculi or implementations.

We have introduced non-standard reasoning services facilitating the debugging of logically incoherent DL terminologies and algorithms to calculate them. Whereas the ideas for axiom pinpointing are more evolved and efficiently implemented, only a relatively naive algorithm for concept pinpointing has been developed as yet. We are currently working on more efficient methods and alternative debugging services, and tests are under way on the DICE terminology.

[7]For the full DICE terminology the overall runtime to calculate MIPS is currently about 40 minutes on a PC with a 1 GHz Pentium.

As our methods are developed for the particular application of the DICE terminology some restrictions apply, most importantly to use *ACC* and to consider unfoldable TBoxes, only. Neither is essential, and we conjecture that it is not too hard to extend our algorithms to find MIPS and GITs both for more expressive languages and for general TBoxes. Both issues will be addressed in future investigations. Further work must also extend the application domain to other terminologies as we believe that the success we have with debugging DICE is not tied to the structure of this particular representation. Of particular interest should be applications related to the Semantic Web effort as our methods are particularly geared to support the debugging of existing terminologies and logical contradictions caused by migration or merging of terminologies.

References

[Baader et a/.,2003] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook.* Cambridge University Press, 2003.

[Borgida *et al,* 1999] A. Borgida, E. Franconi, I. Horrocks, D. McGuinness, and P. Patel-Schneider. Explaining *ACC* subsumption. In *DL-99,* pages 37-40, 1999.

[Cornet and Abu-Hanna, 2002] R. Cornet and A. Abu-Hanna. Evaluation of a frame-based ontology. A formalization-oriented approach. In *Proceedings of MIE2002, Studies in Health Technology & Information,* volume 90, pages 488-93, 2002.

[DIG, 2002] Minutes of the DL Implementation Group-Workshop, 2002. http://dl.kr.org/dig/minutes-012002 . html, visited on January 9, 2003.

[Doyle, 1979] J. Doyle. A truth maintenance system. *Artificial Intelligence,* 12(3):231-272,1979.

[Fiedler, 2001] A. Fiedler. *Prex:* An interactive proof explainer. In R. Gore, A. Leitsch, and T. Nipkow, editors, *IJCAR 2001,* number 2083 in LNAI, pages 416-420,2001.

[Haarslev and Moller, 2001] V. Haarslev and R. Moller. RACER system description. In R. Gore, A. Leitsch, and T. Nipkow, editors, *IJCAR 2001,* number 2083 in LNAI, 2001.

[Horrocks, 1998] I. Horrocks. The FaCT system. In H. de Swart, editor, *Tableaux'98,* number 1397 in LNAI, pages 307-312, 1998.

[Kusters, 2001] R. Kiisters. *Non-Standard Inferences in Description Logics,* volume 2100 *of LNAI.* 2001.

[McGuinness, 1996] Deborah McGuinness. *Explaining Reasoning in Description Logics.* PhD thesis, Department of Computer Science, Rutgers University, 1996.

[Nebel, 1990] B. Nebel. Terminological reasoning is inherently intractable. *AI,* 43:235-249, 1990.

[Quine, 1952] W.V. Quine. The problem of simplifying truth functions. *American Math. Monthly,* 59:521-531,1952.

[Reiter, 1987] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence,* 32:57-95, 1987.