

USING PATTERNS TO EXPLAIN INFERENCES IN *ALCHI*

XI DENG AND VOLKER HAARSLEV AND NEMATOLLAH SHIRI

*Department of Computer Science and Software Engineering
Concordia University, Montreal, Canada*

With the increasing number of applications of Description Logics (DLs), unsatisfiable concepts and inconsistent knowledge bases become quite common, especially when the knowledge bases are large and complex. This makes it challenging, even for experienced knowledge engineers, to identify and resolve these unsatisfiabilities and inconsistencies manually. It is thus crucial to provide services to explain how and why a result is derived. Motivated by the possibility of applying resolution technique in first-order logic to construct explanations for Description Logics, we present an algorithm that uses patterns to generate explanations for unsatisfiability and inconsistency reasoning in *ALCHI*, obtained by extending our previous work on *ALC*. The use of resolution proofs to provide explanations for DL reasoners is due to their focus which, through literals involved in the process, contributes directly to the contradiction, hence acting as filters to discard irrelevant information. We also establish the soundness and completeness of the algorithm. The proposed solution approach is independent of the underlying DL reasoners, which suggests its potential application for any DL framework.

Key words: Description Logics, the Semantic Web, explanations, resolution.

1. INTRODUCTION

In recent years, Description Logics (DLs) have found their way into many application domains, including domain modeling, software engineering, configuration, and the Semantic Web (Baader and Nutt 2003). For example, DLs have deeply influenced the design and standardization of the Web Ontology Language OWL. On the other hand, the acceptance of OWL as a Web standard has resulted in the widespread use of DL ontologies on the Web. As more and more applications emerge with increasing large and complexity, unsatisfiability and inconsistency are more usual to encounter. For example, the DICE (Diagnoses for Intensive Care Evaluation) terminology (Haase and Stojanovic 2005) contains more than 2400 concepts, out of which about 750 concepts were unsatisfiable due to migration from other terminological systems. Unsatisfiability and inconsistency may also arise due to unintentional design defects or changes in the ontology evolution process. Although these problems need immediate attention, existing DL reasoners, such as Racer (Haarslev and Möller 2001) and FaCT (Horrocks 1998), do not provide explanation services; they merely provide “Yes/No” answer to a satisfiability or consistency query without giving any information or reasons. In addition to such primitive answers, it is also desirable that DL reasoners provide explanations for these answers and identify the sources of inconsistencies to further assist knowledge engineers and ontology developers to improve the quality of the knowledge base. It is therefore crucial to provide explanation services as a useful feature and facility for DL reasoners. This paper reports our research in this direction.

In general, there are two approaches to build an explanation system. One way is to construct a system specially designed to provide explanations, as suggested in (Wick and Thompson 1992). Several proposals to support for explanations in theorem proving systems follow this idea (Huang 1994; Meier 2000). Another solution approach is to extend an existing reasoner to include an explanation module, which essentially traces the internal reasoning procedure to produce explanations. Most research on explanations in logic programming and deductive databases follow this approach (Byrd 1980; Shmueli and Tsur 1990). In (Deng et al. 2005; Deng et al. 2006), we proposed a framework for constructing explanations for the DL language *ALC* using resolution proofs, which follows the basic ideas of the first

approach, although the necessary interaction with the DL reasoner is also considered. The DL language \mathcal{ALC} includes constructors such as conjunctions, disjunctions, existential and universal restrictions. In this paper, we extend the language of consideration from \mathcal{ALC} to \mathcal{ALCHI} , by additionally allowing constructors for role hierarchy and inverse role. We investigate unsatisfiability and inconsistency patterns in \mathcal{ALCHI} to further improve efficiency of generating explanations. Based on these patterns, we propose an algorithm and prove its soundness and completeness.

There are three main advantages of our approach. First, the resolution technique can generate explanations at a fine-grained, logical level in contrast to merely debugging the knowledge base. Besides, compared to the previous approaches of using natural deduction proofs to explain reasoning, we use resolution technique which is more focused, since all the literals involved in a proof contribute directly to the solution. Below is a variant of the example in (Huang 1996) illustrating a natural deduction proof style explanation. A number shown in the second column at each row indicates the hypotheses that derivation depends on. The conclusion of the inference is shown in column 3. The last column indicates the inference rule and/or hypotheses used in a row.

No	Hyp.	Conclusion	Reason
1.	1	A	(Hypothesis)
2.	2	$A \sqsubseteq B$	(Hypothesis)
3.	3	$\neg B$	(Hypothesis)
4.	2	$\neg A \vee B$	(Tautology 2)
5.	5	$\neg A$	(Hypothesis)
6.	1, 5	\perp	($\neg E$ 1, 5)
7.	7	B	(Hypothesis)
8.	3,7	\perp	($\neg E$ 3, 7)
9.	1, 2, 3	\perp	(Case analysis 4, 6, 8)
10.	1, 2	B	(Indirect 9)

Considering that this problem could be solved in two steps using resolution as shown below, the natural deduction proof is indeed long and tedious.

No	Hyp.	Clause	Reason
1.	1	$\{A\}$	(Hypothesis)
2.	2	$\{\neg A, B\}$	(Hypothesis)
3.	3	$\{\neg B\}$	(Hypothesis)
4.	1, 2	$\{B\}$	(Resolution 2, 3)
5.	3, 4	\perp	(Resolution 3, 4)

A second advantage of our approach is that it is independent of any specific DL reasoners being used. Most implemented DL reasoners use tableau algorithm as their decision procedure. Having a decidable procedure has been one of the main factors for using tableau algorithms. The development of highly efficient optimization techniques has also demonstrated that acceptable performance can be achieved. However, tableau algorithms are designed to render results faster but not necessarily easier for users to comprehend. For example, some DL optimization techniques are adopted to make reasoning more efficient; ease of understanding of reasoning results by general users are not often the concerns of such techniques. Therefore, in order to generate explanations, the internal reasoning procedures may be tailored with performance penalties. In our approach, explanations are constructed based on resolution proofs, and hence no modification of the internal of the DL reasoner is required. This makes our proposed solution applicable to arbitrary DL reasoners rather easily.

The third advantage of our proposal is that it can exploit and benefit from many features of resolution based theorem provers to provide better explanations. For example, there might be more than one reason for the unsatisfiability or inconsistency in a problem description, and it is not sufficient for the explanation service to stop whenever the first source of the contradiction is detected. Since many resolution based theorem provers, such as Otter (WosMcCune and Wos.L 1997), can be configured to provide all the resolution proofs they can find, we can rely on the provers to provide alternative proofs in our approach.

The rest of the paper is organized as follows. Section 2 provides a brief introduction of Description Logics and resolution. Section 3 presents the overview of the explanation framework and the algorithms for generating explanations. Section 4 includes an example to illustrate the procedure. Section 5 discusses the related work in explanations. Section 6 includes concluding remarks and discusses some future work.

2. PRELIMINARIES

In this section, we introduce the syntax, semantics and inference problems in the DL $\mathcal{ALCH}\mathcal{I}$. We also provide basic background information for resolution techniques. For more materials on resolution, the reader is referred to (Robinson and Voronkov 2001).

2.1. Description Logics

Description Logics are a family of concept-based knowledge representation formalisms. They can represent the knowledge of a domain by first defining the relevant concepts of the domain. These concepts are then used to specify properties of the objects and individuals in the domain. Typically a DL language has two parts: *terminology* (TBox) and *assertion* (ABox). The TBox includes intensional knowledge in the form of axioms whereas the ABox contains the extensional knowledge that is specific to elements in the domain, called individuals.

The language \mathcal{AL} (for Attribute Language) has been introduced as a minimal language of interest in DL. Other languages of this family are various extensions of \mathcal{AL} . In \mathcal{AL} , basic descriptions are *atomic concepts*, designated by unary predicates, and *atomic roles*, designated by binary predicates to express relationships between individuals. Arbitrary concept descriptions such as C and D are defined recursively from atomic concepts and roles using the DL constructors according to the following syntax rules:

$C, D \rightarrow A$	(atomic concept)
\top	(universal concept)
\perp	(bottom concept)
$\neg A$	(atomic negation)
$C \sqcap D$	(intersection)
$\forall R.C$	(value restriction)
$\exists R.\top$	(limited existential quantification)

For example, suppose that **Person** and **Female** are atomic concepts. Then **Person** \sqcap **Female** and **Person** \sqcap \neg **Female** are concepts describing people who are female and people who are not. In addition, if we suppose that **hasSpouse** is an atomic role, then the concept description **Person** \sqcap \exists **hasSpouse**. \top denotes people who have a spouse.

Axioms express how concepts and roles are related to each other. Generally, an axiom is a statement of the form $C \sqsubseteq D$, read as “concept C is subsumed by concept D ”, or $C \equiv D$,

indicating that $C \sqsubseteq D$ and $D \sqsubseteq C$, where C and D are concept descriptions. For instance, by the axiom:

$$\text{Married} \sqsubseteq \text{Person} \sqcap \exists \text{hasSpouse}.\text{Person}$$

we assert that **Married** are people who have spouses.

A TBox is a set of axioms by which we can introduce atomic concepts and concept descriptions. In addition, we can also introduce individuals and assert properties of these individuals in an ABox. For example, suppose **JANE** and **JACK** are individual names, then **Married(JANE)** means that Jane is married, and **hasSpouse(JANE, JACK)** means that JANE is the spouse of JACK.

More expressive language can be obtained if more constructors are added into \mathcal{AL} . For example, \mathcal{ALCHI} is the extension of \mathcal{AL} by allowing negation of arbitrary concepts (indicated by \bar{C} for *Complement*), role hierarchy (indicated by \mathcal{H}) and inverse role (indicated by $\bar{\mathcal{I}}$). With these additional constructors, we can describe that if people have spouses then they have relatives by using role hierarchy: **hasSpouse** \sqsubseteq **hasRelative**. We can also define that the inverse of the role **hasChild** yields the role **hasParent**.

An interpretation \mathcal{I} defines the formal semantics of concepts, roles, and individuals. It consists of a non-empty set $\Delta^{\mathcal{I}}$, called the domain. The interpretation function \mathcal{I} maps every atomic concept A to a subset $A^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, and maps every atomic role R to a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. In addition, \mathcal{I} maps each individual name a to an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. An interpretation \mathcal{I} satisfies $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. It satisfies $C \equiv D$ if $C^{\mathcal{I}} = D^{\mathcal{I}}$. Similarly, it satisfies $R \sqsubseteq S$ if $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$. The interpretation \mathcal{I} is extended to concept descriptions, as shown in Table 1.

Constructor	Semantics
\top	$\Delta^{\mathcal{I}}$
\perp	\emptyset
A	$A^{\mathcal{I}}$
\bar{C}	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
$\forall R.C$	$\{a \in \Delta^{\mathcal{I}} \mid \forall b(a, b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\}$
$\exists R.C$	$\{a \in \Delta^{\mathcal{I}} \mid \exists b(a, b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}$
R^-	$\{(a, b) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (b, a) \in R^{\mathcal{I}}\}$

Axiom	Semantics
$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
$C \equiv D$	$C^{\mathcal{I}} = D^{\mathcal{I}}$
$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$
$a : C$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
$(a, b) : R$	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$

TABLE 1. Interpretation of DL constructors and axioms in \mathcal{ALCHI} .

The basic inference services in TBoxes include satisfiability, subsumption, equivalence, and disjointness. A concept C in a TBox \mathcal{T} is said to be *satisfiable* w.r.t \mathcal{T} if there exists a model of \mathcal{T} (that is an interpretation \mathcal{I} that satisfies the axioms of \mathcal{T}), such that $C^{\mathcal{I}}$ is nonempty. The other three inference services can be reduced to (un)satisfiability. Another important reasoning service in TBoxes is to check whether a TBox \mathcal{T} is *consistent*, i.e., whether there exists a model for \mathcal{T} . The basic reasoning tasks in ABoxes include instance

checking, realization, and retrieval. The instance check verifies if a given individual is an instance of a specified concept. The realization finds the most specific concept that an individual is an instance of. The retrieval finds the individuals in the knowledge base that are instances of a given concept. An ABox \mathcal{A} is *consistent* w.r.t a TBox \mathcal{T} , if there is an interpretation that is a model of both \mathcal{A} and \mathcal{T} (that is an interpretation \mathcal{I} that both satisfies the axioms of \mathcal{T} and the assertions of \mathcal{A}). Similar to the inference services in TBoxes, the other three inference services in ABoxes can also be reduced to the consistency problem of ABoxes. The reader is referred to (Baader and Nutt 2003) for details.

2.2. Resolution

We assume that the reader is familiar with standard definitions of first-order logic (FOL) and clausal theorem proving in the standard logic. Resolution is a most widely used proof procedure for theorem proving in FOL. It proceeds by negating the statement to be proved and adding this negated goal to the axioms that are known to be true. It then uses the following inference rules iteratively to derive a contradiction.

Positive factoring:

$$\frac{C \vee A \vee B}{C\sigma \vee A\sigma}$$

where σ is a most general unifier (mgu) of propositions A and B , denoted as $\text{mgu}(A, B)$.

Resolution:

$$\frac{C_1 \vee \dots \vee C_n \vee A \quad D_1 \vee \dots \vee D_n \vee \neg B}{C_1\sigma \vee \dots \vee C_n\sigma \vee D_1\sigma \vee \dots \vee D_n\sigma}$$

Resolution is sound and complete: if a set of clauses is saturated up to redundancy by the inference rules, then it is satisfiable if and only if it does not contain the empty clause.

3. THE EXPLANATION PROCEDURE

In this section, we present our resolution based framework to provide explanations for unsatisfiability and inconsistency queries in $\mathcal{ALCH}\mathcal{I}$. We also propose our explanation procedure and establish its soundness and completeness.

3.1. Resolution Based Framework

The proposed explanation framework consists of three phases, described as follows. The architecture of the system is shown in Figure 1.

1. **Preprocessing:** The explanation system communicates with a DL reasoner, e.g., Racer, which provides the answer to a query, normally in the form of “Yes” or “No” for a concept satisfiability, or an ABox consistency query, or a list of unsatisfiable concepts for TBox consistency checking. If the answer is “No”, this means a concept is unsatisfiable or a TBox/ABox is inconsistent. Then the original TBox/ABox will be translated into FOL formulas or clauses by the translation component.
2. **Rendering resolution proofs:** A resolution based automated theorem prover is applied to the FOL formulas or clauses resulted from step 1 in order to generate resolution proofs. It is important to note that since a DL reasoner is used first to get the result of the query, the unsatisfiability of the concept or the inconsistency of the TBox/ABox is already known.

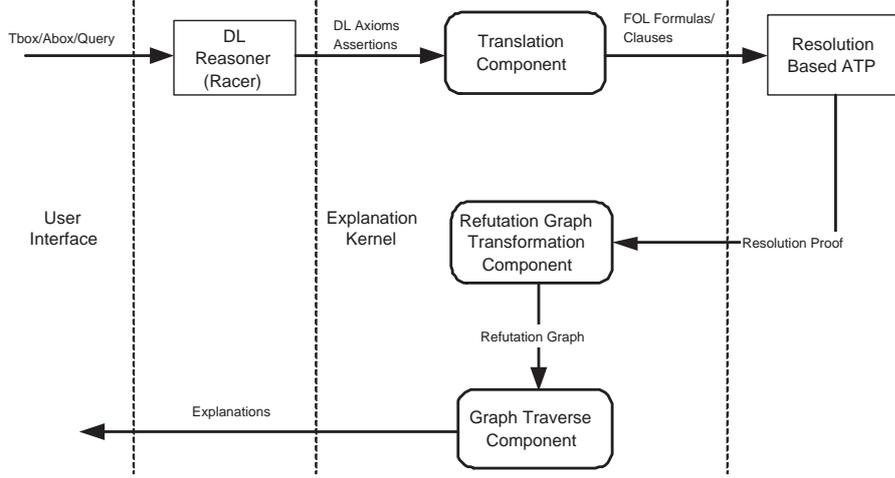


FIGURE 1. The explanation system architecture.

The complexity of the resolution based decision procedure of the guarded fragment of FOL is double exponential in the worst case (Ganzinger and de Nivelle 1999). Since \mathcal{ALCHI} can be embedded into the guarded fragment, its upper bound of computational complexity falls into the same category.

3. Explaining: The resolution proof is used by the explanation kernel to construct explanations for better human understanding. Our approach transforms the proof into its corresponding refutation graph (Eisinger 1991), which is a more abstract representation for the refutation proof. A refutation graph is a graph whose nodes are literals (grouped together in clauses) and its edges connect complementary literals. As shown in (Eisinger 1991), for each resolution proof, there is a minimal refutation graph, but one refutation graph can represent multiple different sequences of resolution steps. Hence we can dynamically search to find one of the best sequences as explanations. Finally, the clauses involved in each resolution step are referred back to the contributing DL axioms/assertions or FOL formulas and possibly transformed into natural language explanations later.

3.2. Translation Between DL And FOL

The translation between DL and FOL is based on the semantics of DL. For \mathcal{ALCHI} , concepts can be translated into \mathcal{L}^2 , which is a first order predicate logic over unary and binary predicates with two variables, say x, y . Table 2 shows this translation from \mathcal{ALCHI} into \mathcal{L}^2 . An atomic concept A is translated into a predicate logic formula $\phi_A(x)$ with one free variable x such that for every interpretation \mathcal{I} , the set of elements of $\Delta^{\mathcal{I}}$ satisfying $\phi_A(x)$ is exactly $A^{\mathcal{I}}$. Similarly, a role name R is translated into a binary predicate $\phi_R(x, y)$. An individual name a is translated into the constant a .

In case of TBox inconsistency queries, there are two possible scenarios: a set of unsatisfiable concepts or an inconsistent TBox, i.e., the top concept is inconsistent thus causing all the concepts in the TBox to be unsatisfiable. In case of ABoxes inconsistency queries, there are two possible scenarios: an individual is asserted as an instance of an unsatisfiable concept

DL Constructor	FOL Formula
A	$\phi_A(x)$
$\neg C$	$\neg\phi_C(x)$
$C \sqcap D$	$\phi_C(x) \wedge \phi_D(x)$
$C \sqcup D$	$\phi_C(x) \vee \phi_D(x)$
$\exists R.C$	$\exists y(\phi_R(x, y) \wedge \phi_C(y))$
$\forall R.C$	$\forall y(\phi_R(x, y) \rightarrow \phi_C(y))$
R, R^-	$\forall x, y(\phi_R(x, y) \leftrightarrow \phi_{R^-}(y, x))$

DL Axiom	FOL Formula
$C \sqsubseteq D$	$\forall x(\phi_C(x) \rightarrow \phi_D(x))$
$C \equiv D$	$\forall x(\phi_C(x) \leftrightarrow \phi_D(x))$
$R \sqsubseteq S$	$\forall x, y(\phi_R(x, y) \rightarrow \phi_S(x, y))$
$a : A$	$\phi_A(a)$
$(a, b) : R$	$\phi_R(a, b)$

 TABLE 2. Translation from $\mathcal{ALCH}\mathcal{I}$ into \mathcal{L}^2 .

or it is asserted to belong to an inconsistent concept description. Consequently, these cases are distinguished in the translation.

Definition 1. Let \mathcal{T} be a TBox, and C be an unsatisfiable concept in \mathcal{T} . An *unsatisfiability translation function* Γ w.r.t. C translates $\mathcal{T} \cup \{C\}$ into a set of FOL formulas.

For example, suppose $\mathcal{T} = \{\text{Person} \sqsubseteq \text{Man} \sqcap \neg\text{Man}\}$. The concept **Person** is unsatisfiable, so **Person** and the axiom in \mathcal{T} form the input set of Γ .

Definition 2. Let \mathcal{T} be a TBox and \mathcal{A} be an ABox (either \mathcal{T} or \mathcal{A} can be empty). An *inconsistency translation function* Λ translates $\mathcal{T} \cup \mathcal{A}$ into a set of FOL formulas.

For example, suppose TBox \mathcal{T} is $\{\text{Person} \sqsubseteq \perp, \top \sqsubseteq \text{Person}\}$. Here, \mathcal{T} is inconsistent, so the two axioms form the input set of Λ .

A straightforward translation as shown in Table 2 would easily cause the exponential blow up of the number of the clauses. Consider the axiom $E \sqsubseteq F$, where E and F are complex concept descriptions. If n and m are the numbers of clauses generated by E and F respectively, then the above formula generates $n \times m$ clauses. The reason for the exponential explosion is duplication of subformulae obtained by the exhaustive application of the distributive law. In order to avoid such blow-ups, we adopt the standard translation augmented by *structural transformation* as in (Hustadt et al. 2005). Roughly speaking, the structural transformation is a kind of conjunctive normal form (CNF) transformation of first-order predicate logic formulae by replacing the subformulae with some new predicates and adding suitable definitions for these predicates. In the previous example, if we replace F by a fresh concept,

say C , then the above axiom transforms into two: $E \sqsubseteq C$ and $C \sqsubseteq F$. The number of clauses generated by these two axioms is $n+m$. Besides, the structural transformation also helps preserve original structures of DL axioms after their corresponding first-order logic formulae are transformed into conjunctive normal forms. For instance, consider the axiom $\forall R.A \sqsubseteq \exists S.B$. Without the transformation, the subsumee and subsumer of this axiom are distributed into four clauses, $\{R(x, f_1(x)), S(x, f_2(x))\}$, $\{R(x, f_1(x)), B(x, f_2(x))\}$, $\{S(x, f_2(x)), \neg A(f_1(x))\}$, and $\{\neg A(f_1(x)), B(f_2(x))\}$, making it difficult to generate comprehensible explanations.

The structural transformation can be formally described as follows. More details can be found in (Nonnengart et al. 1998). Since the basic idea is to replace non-atomic concepts, which are subformula of the axioms, with new concept names, the notion of formula positions are used to define the transformation. For example, let φ be a formula and $\phi = \varphi|_\pi$ be a subformula of φ at position π that we want to replace. The transformation will replace ϕ with a predicate new to φ , say R .

Definition 3. A position is a word over the natural numbers. The set $pos(\varphi)$ of positions of a given formula is defined as follows:

- the empty word $\varepsilon \in pos(\varphi)$
- for $1 \leq i \leq n$, $i.p \in pos(\varphi)$ if $\varphi = \varphi_1 \circ \dots \circ \varphi_n$ and $p \in pos(\varphi_i)$ where \circ is a first-order operator. If $p \in pos(\varphi)$, $\varphi|_{i.p} = \varphi_i|_p$ where $\varphi = \varphi_1 \circ \dots \circ \varphi_n$. We write $\varphi[\phi]_p$ for $\varphi|_p = \phi$. With $\varphi[p/\phi]$ where $p \in pos(\varphi)$ we denote the formula obtained by replacing $\varphi|_p$ with ϕ at position p in φ . The polarity of a formula occurring at position in a formula is denoted by $Pol(\varphi, \pi)$ and defined as: $Pol(\varphi, \varepsilon) = 1$; $Pol(\varphi, \pi.i) = Pol(\varphi, \pi)$ if $\varphi|_\pi$ is a conjunction, disjunction, formula starting with a quantifier or an implication with $i = 2$; $Pol(\varphi, \pi.i) = -Pol(\varphi, \pi)$ if $\varphi|_\pi$ is a formula starting with a negation symbol or an implication with $i = 1$ and, $Pol(\varphi, \pi.i) = Pol(\varphi, \pi)$ if $\varphi|_\pi$ is an equivalence.

Definition 4. Let φ be a formula and $\phi = \varphi|_\pi$ be a subformula of φ at position π . Let x_1, \dots, x_n be the free variables in φ and let R be a new predicate. Then the formula

$$\varphi[\pi/R(x_1, \dots, x_n)] \wedge Def_\pi^\varphi$$

is a structural transformation of φ at position ϕ . The formula Def_π^φ is a polarity dependent definition of the new predicate R :

$$Def_\pi^\varphi = \begin{cases} \forall x_1, \dots, x_n [R(x_1, \dots, x_n) \rightarrow \phi] & \text{if } Pol(\varphi, \pi) = 1 \\ \forall x_1, \dots, x_n [\phi \rightarrow R(x_1, \dots, x_n)] & \text{if } Pol(\varphi, \pi) = -1 \end{cases}$$

There are six types of clauses in \mathcal{ALCHI} after normalization:¹

1. $\bigvee X_i$
2. $\bigvee X_i \vee R(x, f(x))$
3. $\bigvee X_i \vee Y$
4. $\bigvee X_i \vee \neg R(x, y) \vee Z$
5. $\neg R(x, y) \vee (R^-)(y, x)$
6. $\neg R(x, y) \vee S(x, y)$

where $X_i \in \{C_i(x), \neg C_i(x)\}$, $Y \in \{D(f(x)), \neg D(f(x))\}$, and $Z \in \{D(y), \neg D(y)\}$.

¹We only consider TBoxes here. ABoxes cases have the similar syntactic properties.

Specifically, clause type (1) is translated from axioms $C_i \sqsubseteq C_j$, with both C_i and C_j being complex concepts. Types (2) and (3) are translated from axioms $C \sqsubseteq \exists R.D$ or $\forall R.C \sqsubseteq D$. Type (4) is translated from axioms $C \sqsubseteq \forall R.D$ or $\exists R.C \sqsubseteq D$. Type (5) is the result of translating from the definition of inverse role. Type (6) is translated from role hierarchy $R \sqsubseteq S$.

The following theorem is the basis of the correctness of the translation.

Theorem 1. Let \mathcal{T} be a consistent TBox in $\mathcal{ALCH}\mathcal{I}$ and C be a named concept in \mathcal{T} . Then C is unsatisfiable if and only if the empty clause is derived under resolution given Γ w.r.t. C .

Proof. As proved in (Hustadt et al. 2005), the structural transformation does not affect satisfiability. Let $\Theta(\mathcal{T})$ and $\Theta(C(\mathbf{a}))$ be the resulting set of FOL formulae of \mathcal{T} and $C(\mathbf{a})$ after the translation, where \mathbf{a} is a newly introduced individual. As \mathcal{T} is given to be consistent, $\Theta(\mathcal{T})$ is also consistent. Since C is unsatisfiable, C does not admit any instance, i.e., $C(\mathbf{a})$ is inconsistent. Hence $\Theta(\mathcal{T}) \cup \Theta(C(\mathbf{a}))$ is inconsistent. Due to completeness of refutation resolution, the empty clause can be derived.

On the other hand, if the empty clause is derived from $\Theta(\mathcal{T}) \cup \Theta(C(\mathbf{a}))$, then $\Theta(\mathcal{T}) \cup \Theta(C(\mathbf{a}))$ is inconsistent. Since \mathcal{T} is given to be consistent, C must be unsatisfiable. ■

The following result can also be obtained due to the consistency preserving property of the translation.

Theorem 2. Let \mathcal{T} be a TBox and \mathcal{A} be an ABox (either \mathcal{T} or \mathcal{A} can be empty). Then $\mathcal{T} \cup \mathcal{A}$ is inconsistent if and only if the empty clause is derived by resolution, given $\Lambda(\mathcal{T} \cup \mathcal{A})$.

Proof. As proved in (Hustadt et al. 2005), the structural transformation does not affect satisfiability. If $\mathcal{T} \cup \mathcal{A}$ is inconsistent, then $\Lambda(\mathcal{T} \cup \mathcal{A})$ is inconsistent, which means the empty clause can be derived. On the other hand, if the empty clause is derived from $\Lambda(\mathcal{T} \cup \mathcal{A})$, then it is inconsistent. Consequently, $\mathcal{T} \cup \mathcal{A}$ is inconsistent. ■

3.3. Resolution Proofs

After the translation step, a resolution based theorem prover is invoked to generate resolution proofs. To illustrate this procedure, we consider the following example of a TBox.

1. $A1 \sqsubseteq A2 \sqcap \neg A \sqcap A3$
2. $A2 \sqsubseteq A \sqcap A4$

The concept $A1$ is unsatisfiable. After the translation, the resulting clauses are:

$$\begin{array}{ll}
 \mathbf{C}_0 = \{A1(c)\} & \\
 \mathbf{C}_{11} = \{\neg A1(x), A2(x)\} & \\
 \mathbf{C}_{12} = \{\neg A1(x), \neg A(x)\} & \mathbf{C}_{13} = \{\neg A1(x), A3(x)\} \\
 \mathbf{C}_{21} = \{\neg A2(x), A(x)\} & \mathbf{C}_{22} = \{\neg A2(x), A4(x)\}
 \end{array}$$

Below is one of the possible resolution proofs. The numbers shown in the second column of each row indicate the hypotheses it depends on. The inference rule that justifies a row is given after the conclusion formula, followed by the premise rows.

No	Hyp.	Clause	Reason
1.	1	$\{A1(c)\}$	(Hypothesis)
2.	2	$\{\neg A1(x), A2(x)\}$	(Hypothesis)
3.	3	$\{\neg A1(x), \neg A(x)\}$	(Hypothesis)
4.	4	$\{\neg A2(x), A(x)\}$	(Hypothesis)
5.	1, 2	$\{A2(x)\}$	(Resolution 1, 2)
6.	1, 3	$\neg A(x)$	(Resolution 1, 3)
7.	5, 6, 4	\perp	(Resolution 5, 6, 4)

Since the resolution technique operates on clauses, which are on a finer-grained level than the original DL axioms, it can determine not only which axioms are relevant but also which parts of the asserted axioms are relevant for the particular unsatisfiability or inconsistency problem. In this example, the conjunct $A3$ in Axiom 1 and $A4$ in Axiom 2 are irrelevant for the unsatisfiability of A , hence its corresponding clauses do not appear in the resolution proof. In order to reflect this characteristics in the explanation, we define the minimal DL axiom counterparts of clauses as the *clausal axioms*, shown as follows.

Definition 5. A *clausal axiom* is inductively defined as follows, where C_i , C_j and C_k are complex concepts, D is an atomic concept, and \otimes stands for \sqcap or \sqcup .

- if the original DL axiom of a clause $\bigvee X_i \vee D$ is $C_i \sqsubseteq D \sqcap C_j$, then its clausal axiom is $C_i \sqsubseteq D$;
- if the original DL axiom of a clause $\bigvee X_i \vee \neg D$ is $D \sqcup C_i \sqsubseteq C_j$, then its clausal axiom is $D \sqsubseteq C_j$;
- if the original DL axiom of a clause $\bigvee X_i \vee R(x, f(x))$ is $C_i \sqsubseteq \exists R.C_j \otimes C_k$ (or $\forall R.C_i \otimes C_k \sqsubseteq C_j$), then its clausal axiom is $C_i \sqsubseteq \exists R.\top \otimes C_k$ (or $\forall R.\top \otimes C_k \sqsubseteq C_j$).

3.4. Refutation Graphs

The generated resolution proof can be further transformed into its refutation graph representation to construct explanations (Deng et al. 2005). A refutation graph is based on a set of *literal nodes*, which are nodes labeled with literals. These literal nodes are grouped together to *clause nodes*, which represent multisets of literals (different literal nodes may be labeled with the same literal), i.e., clauses. Usually, literal nodes are represented as small boxes labeled with their literals. Adjacent boxes denote a clause, i.e., the conjunction of the literals in the boxes. By traversing the graph, a best way to read the proof can be found.

We will use the following definitions of refutation graphs in this paper. Please see (Eisinger 1991) for more details.

Definition 6. A *refutation graph* is a quadruple $\mathcal{G} = (\mathcal{L}, \mathcal{C}, \mathcal{M}_{\mathcal{L}}, \mathcal{K})$, where \mathcal{L} is a finite set of literal nodes in \mathcal{G} , and \mathcal{C} is a partition of the set of literal nodes, whose members are clause nodes in \mathcal{G} . $\mathcal{M}_{\mathcal{L}}$ is a mapping from \mathcal{L} to a set of literals, which associates with every literal node a literal. The set of links \mathcal{K} is a partition of a subset of \mathcal{L} . All the literal nodes in one link are labeled with literals which are unifiable. There is no pure literal node in a refutation graph, i.e., every literal node belongs to some link in \mathcal{K} .

The refutation graph of the example in Section 3.3 is shown in Figure 2. We extend the above notions to provide explanations for DL reasoning.

Definition 7. A *labeled refutation graph* is a quintuple $\mathcal{G}' = (\mathcal{L}, \mathcal{C}, \mathcal{M}_{\mathcal{L}}, \mathcal{K}, \mathcal{M}_{\mathcal{D}})$, where $\mathcal{L}, \mathcal{C}, \mathcal{M}_{\mathcal{L}}$ and \mathcal{K} are as defined in the refutation graph \mathcal{G} above. And $\mathcal{M}_{\mathcal{D}}$ is a mapping from \mathcal{L} to a FOL formula, which labels the literal nodes with their originating FOL formulas.

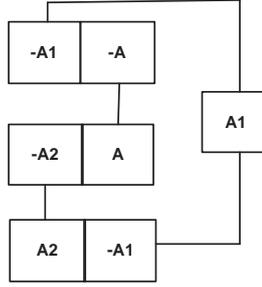


FIGURE 2. The refutation graph.

In our example, one of the mappings in $\mathcal{M}_{\mathcal{D}}$ is as follows:

$$\{\neg A1(x)\} \mapsto \forall x A1(x) \rightarrow A2(x) \wedge \neg A(x) \wedge A3(x)$$

Definition 8. A *traversing path* of the labeled refutation graph involving clause nodes C_1, \dots, C_n is a sequence $(M_1, C_1), (L_2, C_2), (M_2, C_2), (L_3, C_3), (M_3, C_3), \dots, (L_n, C_n)$, where L_i and M_i , for $i = 1, \dots, n$, are literal nodes in C_i and $L_i \neq M_i$. Also for all $i < n$, M_i and L_{i+1} unify, i.e., there is a link between them.

Definition 9. A *bridge* in a refutation graph connects two sets of complementary literal nodes, which are involved in a resolution step. A *traversal ordering* \leq over the bridges. A factoring link (*f-link* for short) connects the literals that participate in a factoring step.

We believe that the quality of explanations largely depends on how the refutation graph is traversed, i.e., on how \leq is defined. And the ordering can be decided according to the unsatisfiability and inconsistency patterns. Generally speaking, an inconsistency pattern (*i-pattern* for short) is a resolving step over the concept names and the role names. It helps decide the *traversal ordering* \leq among the bridges.

Definition 10. The set of ***i-patterns*** over \mathcal{ALCHIT} is defined as one of the following cases:

Pattern 1. $\exists R.\top, \forall R.C_i$, and $\bigcap C_i = \perp$, where $i = 1, \dots, n$.

The simplified version presented in the form of a refutation graph is shown in Figure 3. The dotted line shows the possible intermediate resolving steps between the two literals in bold boxes. In this case, all the bridges connecting $R(x, f(x))$ and $\neg R(x, y)$ have the same traversal order, which is higher than the order of the bridges connecting the literal nodes $C_i(y)$.

Pattern 2. $\exists R.D, \forall R.C_i$, and $D \sqcap \bigcap C_i = \perp$, where $i = 1, \dots, n$.

The simplified refutation graph is shown in Figure 4. Similarly, the bridge connecting $R(x, f(x))$ and $\neg R(x, y)$ has a higher order than bridges connecting $C_i(y)$ and $D(f(x))$.

Pattern 3. $\bigcap C_i = \perp$, where $i = 1, \dots, n$.

In this case, every C_i resides in different clause nodes. All the bridges have the same traversal order. The simplified refutation graph is shown in Figure 5.

Pattern 4. $\bigcup C_i$, where each C_i is an *i-pattern*, for $i = 1, \dots, n$.

All the bridges adjacent to C_i have the same traversal order. The simplified refutation graph is shown in Figure 6.

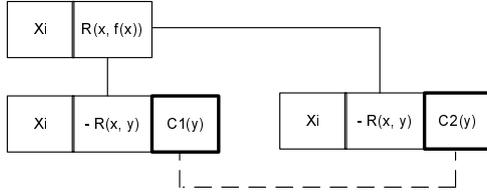


FIGURE 3. Pattern 1.

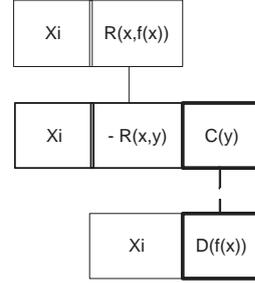


FIGURE 4. Pattern 2.

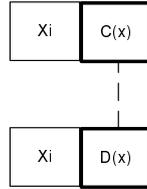


FIGURE 5. Pattern 3.

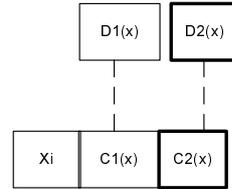


FIGURE 6. Pattern 4.

The ordering of the bridges in the refutation graph is based on the i -patterns. A positive integer is set to be the initial value of each of the bridges in the graph. Then the graph is inspected to match the i -patterns. If a subgraph is matched with pattern 1 or 2, the involved bridges increase the ordering value by 1. Otherwise the ordering value remains unchanged.

3.5. The Algorithm

After the traversal ordering of the bridges in the refutation graph is decided, the *traverse algorithm* is applied to generate explanations. The main idea of our explanation algorithm is based on the refutation graph. It starts from a literal node N_0 and traverses the graph. Among all the bridges adjacent to N_0 , it chooses the one(s) that are not lower than any others w.r.t. \leq . Then it chooses all the literal nodes N_i on the opposite side of the chosen bridges from N_0 . Among each of the clause nodes that N_i resides in, it chooses the yet-to-be-chosen literal nodes. The traversal process ends when all the nodes in the graph have been chosen. After the traversal is completed, the clause nodes involved in each step are translated into an entry in an explanation list (EL) consisting of their clausal axioms in DL. After some clean-up process, such as deleting duplicate lines, the explanation list can be further transformed into natural language style explanations. This algorithm is formally stated in Figure 7. It uses a stack, called SOT , which includes the literal nodes which are yet to be traversed.

Theorem 3. The traversal algorithm is sound, complete and will terminate with an explanation.

Proof. Termination: In each step of the traversal, the number of literal nodes that remain untraversed decreases, since once a literal node is traversed, it will not be traversed again.

 Algorithm **TRAVERSE**
Input: a refutation graph \mathcal{R} **Output:** explanation list EL

```

if the query is an unsatisfiable query
  start from the unsatisfiable concept  $\mathcal{C}$  in  $\mathcal{R}$ 
else start from an arbitrary concept  $\mathcal{C}$  in  $\mathcal{R}$ 
   $SOT$  (Stack of Traversal)  $\leftarrow$  the associated literal node of  $\mathcal{C}$ 
for all the literal nodes  $L_i$  in  $SOT$ 
  put the corresponding clausal axiom of  $L_i$  into the explanation list  $EL$ ;
  for all the bridges which are adjacent to  $L_i$ , with the highest traversal order
  for all the literal nodes  $L_k$  that reside in the same clause node as the
  opposite side of the bridge
     $SOT \leftarrow L_k$ 
  remove the bridge
  remove  $L_i$  from  $SOT$ 
return the explanation list  $EL$ 

```

end of **TRAVERSE**

 FIGURE 7. Traversal algorithm.

As the number of literal nodes in a refutation graph is finite, the traversal algorithm will terminate. More precisely, since the traversal step is applied once for each literal node, i.e., the number of the traversal steps is linear in the number of the literal nodes.

Soundness: Since every literal node is visited exactly once, and all the literal nodes in the refutation graph are translated from the axioms that contribute to the unsatisfiability and inconsistency, the traversal algorithm will present these culprits in the explanations.

Completeness: The completeness in our context means that at the end of the algorithm, no literal node is left untraversed. The fact that we cannot reach a blocked situation follows upon the fact that every literal node in the refutation graph has a complementary literal node connected by a link, i.e., every literal node is reachable through other nodes. ■

4. EVALUATION

We have developed a running prototype system based on our proposal and performed some experiments. The initial results are promising. This system consists of three parts: (1) The translation component: which transforms the problem description, i.e., the TBox, the ABox (if any) and the negated unsatisfiable concept (if it is an unsatisfiability problem) into the corresponding first order formulae. (2) The transformation component: which uses the resolution based theorem prover Otter (WosMcCune and Wos.L 1997) in order to obtain the resolution proof, which is then transformed into a refutation graph. (3) The traverse component: which generate the explanations based on the refutation graph. The input theory is in the KRSS syntax (Patel-Schneider and Swartout 1993). All the components are implemented in Java.

We next illustrate the computation of the prototype system, using the following input example KB .

By using Racer, we get to know that *HappyPerson* is unsatisfiable. The KB is aug-

-
1. $Physician \sqsubseteq \exists hasDegree.BS$
 2. $HappyPerson \sqsubseteq Doctor \sqcup \exists hasChild.(PhD \sqcap \neg Poor)$
 3. $\exists hasParent.HappyPerson \sqsubseteq Married$
 4. $MD \sqsubseteq \neg BS$
 5. $Married \sqsubseteq Person \sqcap \exists hasSpouse.Person$
 6. $Doctor \sqsubseteq \forall hasDegree.MD \sqcap Physician$
 7. $PhD \sqcap Married \sqsubseteq Poor$
 8. $hasParent \equiv hasChild^-$
-

mented with $HappyPerson(a)$ where a is a fresh individual. We call the resulting knowledge base KB' . The goal now is to show that KB' is inconsistent.

Axiom 1, 2, 3, 5 and 6 contain non-literal subconcepts. But as the structural transformation does not neither decrease the number of the clauses nor simplify the explanations for 1, 3, 5 and 6, we only show how axiom 2 is converted to FOL formulae based on structural transformation. We introduce a new name Q for this subconcept and obtain

$$HappyPerson \sqsubseteq Doctor \sqcup \exists hasChild.Q$$

$$Q \sqsubseteq PhD \sqcap \neg Poor$$

The set of clauses in KB' after normalization is shown as follows.

-
- 1.1. $\neg Physician(x) \vee hasDegree(x, f_1(x))$
 - 1.2. $\neg Physician(x) \vee BS(f_1(x))$
 - 2.1. $\neg HappyPerson(x) \vee Doctor(x) \vee hasChild(x, f_2(x))$
 - 2.2. $\neg HappyPerson(x) \vee Doctor(x) \vee Q(f_2(x))$
 - 2.3. $\neg Q(x) \vee PhD(x)$
 - 2.4. $\neg Q(x) \vee \neg Poor(x)$
 - 3.1. $\neg hasParent(x, y) \vee \neg HappyPerson(y) \vee Married(x)$
 - 4.1. $\neg MD(x) \vee \neg BS(x)$
 - 5.1. $\neg Married(x) \vee Person(x)$
 - 5.2. $\neg Married(x) \vee hasSpouse(x, f_3(x))$
 - 5.3. $\neg Married(x) \vee Person(f_3(x))$
 - 6.1. $\neg Doctor(x) \vee \neg hasDegree(x, y) \vee MD(y)$
 - 6.2. $\neg Doctor(x) \vee Physician(x)$
 - 7.1. $\neg PhD(x) \vee \neg Married(x) \vee Poor(x)$
 - 8.1. $\neg hasChild(x, y) \vee hasParent(y, x)$
 - 8.2. $\neg hasParent(y, x) \vee hasChild(x, y)$
 9. $HappyPerson(a)$
-

By applying hyper resolution to this set of clauses and converting the resolution proof to its refutation graph in our prototype system, we obtain the graph shown in Figure 8. From the resolution as well the refutation graph, we know that axiom 5 does not contribute to the unsatisfiability of $HappyPerson$ since none of its clauses is in the resolution proof.

By applying the algorithm to explain unsatisfiable concepts, our prototype system produce the following traversal order, with the traversed literal node followed by its clausal axiom:

8. $HappyPerson(a)$ [**HappyPerson(a)**]
- \hookrightarrow 2.1. $\neg HappyPerson(x) \vee Doctor(x) \vee hasChild(x, f_2(x))$
[HappyPerson \sqsubseteq Doctor \sqcup $\exists hasChild.$ \top]
- \hookrightarrow 6.2. $\neg Doctor(x) \vee Physician(x)$

$$\begin{aligned}
 & \text{[Doctor} \sqsubseteq \text{Physician]} \\
 & \hookrightarrow \mathbf{1.2.} \neg\text{Physician}(x) \vee \text{BS}(f_1(x)) \text{ [Physician} \sqsubseteq \exists\text{hasDegree.BS]} \\
 & \quad \hookrightarrow \mathbf{4.1.} \neg\text{MD}(x) \vee \neg\text{BS}(x) \text{ [MD} \sqsubseteq \neg\text{BS]} \\
 & \quad \quad \mathbf{1.1.} \neg\text{Physician}(x) \vee \text{hasDegree}(x, f_1(x)) \\
 & \quad \quad \text{[Physician} \sqsubseteq \exists\text{hasDegree.}\top\text{]} \\
 & \hookrightarrow \mathbf{6.1.} \neg\text{Doctor}(x) \vee \neg\text{hasDegree}(x, y) \vee \text{MD}(y) \\
 & \quad \text{[Doctor} \sqsubseteq \forall\text{hasDegree.MD]} \\
 \mathbf{2.2.} & \neg\text{HappyPerson}(x) \vee \text{Doctor}(x) \vee Q(f_2(x)) \\
 & \text{[HappyPerson} \sqsubseteq \text{Doctor} \sqcup \exists\text{hasChild.Q]} \\
 & \hookrightarrow \mathbf{2.3.} \neg Q(x) \vee \text{PhD}(x) \text{ [Q} \sqsubseteq \text{PhD} \sqcap \neg\text{Poor]} \\
 & \quad \mathbf{2.4.} \neg Q(x) \vee \neg\text{Poor}(x) \text{ [Q} \sqsubseteq \text{PhD} \sqcap \neg\text{Poor]} \\
 \mathbf{3.1.} & \neg\text{hasParent}(x, y) \vee \neg\text{HappyPerson}(y) \vee \text{Married}(x) \\
 & \text{[\exists\text{hasParent.HappyPerson} \sqsubseteq \text{Married]} \\
 & \hookrightarrow \mathbf{8.1.} \neg\text{hasChild}(x, y) \vee \text{hasParent}(y, x) \\
 & \quad \text{[hasParent} \equiv \text{hasChild}^\neg\text{]} \\
 & \hookrightarrow \mathbf{7.1.} \neg\text{PhD}(x) \vee \neg\text{Married}(x) \vee \text{Poor}(x) \\
 & \quad \text{[PhD} \sqcap \text{Married} \sqsubseteq \text{Poor]}
 \end{aligned}$$

The traversal steps are further organized and represented by the system as the following explanations:

$$\begin{aligned}
 & \text{HappyPerson}(a) \\
 & \hookrightarrow \text{HappyPerson} \sqsubseteq \text{Doctor} \sqcup \exists\text{hasChild.Q} \\
 & \quad \hookrightarrow \text{Doctor} \sqsubseteq \text{Physician} \\
 & \quad \quad \hookrightarrow \text{Physician} \sqsubseteq \exists\text{hasDegree.BS} \\
 & \quad \hookrightarrow \text{Doctor} \sqsubseteq \forall\text{hasDegree.MD} \\
 & \quad \quad \hookrightarrow \text{MD} \sqsubseteq \neg\text{BS} \\
 & \hookrightarrow \text{HappyPerson} \sqsubseteq \text{Doctor} \sqcup \exists\text{hasChild.Q} \\
 & \quad \hookrightarrow \text{Q} \sqsubseteq \text{PhD} \sqcap \neg\text{Poor} \\
 & \hookrightarrow \exists\text{hasParent.HappyPerson} \sqsubseteq \text{Married} \\
 & \quad \hookrightarrow \text{hasParent} \equiv \text{hasChild}^\neg \\
 & \quad \hookrightarrow \text{PhD} \sqcap \text{Married} \sqsubseteq \text{Poor}
 \end{aligned}$$

The explanation reads as follow:

If a is a HappyPerson, then it can either be a Doctor or has a child which is Q (PhD and not Poor). First, if it is a Doctor, then all its degree is MD and it is a Physician. Every Physician has a BS degree, however, BS is disjoint with MD. So there is a conflict within the branch of Doctor. Secondly, if a has a child which is a PhD and not poor, since every child of a HappyPerson is Married and every married PhD is poor, then a 's child must be poor, which is a contradiction. So a cannot be a HappyPerson.

5. RELATED WORK

There have been several proposals to provide explanations for DL reasoning. The earliest work is (McGuinness 1996), which provides an explanation facility for subsumption and non subsumption reasoning in CLASSIC (Brachman et al. 1990). CLASSIC is a family of knowledge representation systems based on DLs. It allows constructors such as universal quantification, conjunction and restricted number restrictions. Since disjunction is excluded in CLASSIC, explanations are generated based on structural subsumption comparisons. Lengthy explanations are decomposed into smaller steps and a single step explanation is followed by more detailed explanations. Being one of the first DL systems to obtain an

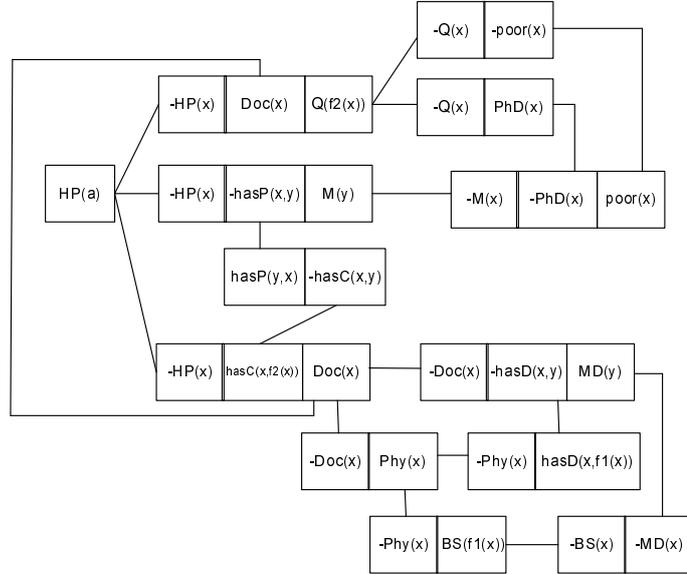


FIGURE 8. The refutation graph of the example using abbreviated predicate names

explanation facility, CLASSIC generates explanations in a proof-theoretic framework. However, since the expressive power of CLASSIC is limited, this approach is not applicable to the DL language \mathcal{ALC} .

This work is extended in (Borgida et al. 1999) by using sequent rules to explain subsumption in \mathcal{ALC} . The sequent rules are modified to imitate the behavior of tableau calculus as well as of human reasoning. For example, in order to avoid confusion, sequent rules are defined in such a way that formulas can never be shifted from antecedents to consequents or vice versa. This approach is based on the observation that sequent calculi are a natural way of explaining subsumption as they axiomatize entailment. It aims to provide explanations for subsumption but not unsatisfiability.

In contrast to these works, (Schlobach and Cornet 2003) provides algorithms to pinpoint unsatisfiable concepts and related axioms. This approach first excludes axioms that are irrelevant to the inconsistency, and then provide simplified definitions which highlight the exact position of the contradiction. This work is extended in (Parsia et al. 2005b; Parsia et al. 2005a) to debug OWL ontologies using both glass box and black box. Glass box relies on information from internals of the reasoners. It traces back to the last clash to give the sources of inconsistency. One drawback of this solution is that showing clashes alone is not specific enough to point out the source of unsatisfiability. In addition, glass box needs to extend or to alter the reasoner, e.g., modify normalization/absorption, to keep track of the source axioms, thus efficiency of the reasoner is affected. Black box approach uses reasoners as oracles and exploits the asserted structure of the ontology to help isolate the source of problems. This method is only used for detecting dependencies between unsatisfiable concepts.

On the other hand, the insufficiency of explanations in automated theorem proving (ATP) has been one of the problems that impede its usability. Therefore researchers in ATP have made efforts to transform machine-generated proofs into human understandable proofs. Most existing explanation facilities in resolution based ATP transform the proofs into natural lan-

guage style explanations. For example, (Felty and Miller 1987) uses the sequential variant of natural deduction to present proofs, then further transforms them into natural language explanations. PROVERB (Huang 1994; Huang and Fiedler 1996) uses a reconstructive approach to generate proofs which are logically equivalent to the original natural deduction proofs, but at a higher level of abstraction. These explanation facilities are specifically designed to solve problems in theorem proving, particularly mathematical theorems. The above approaches focus more on deriving conclusions using theorems, lemmas and axioms, which are not in general suitable for indirect proofs. The closest work to our approach that we are aware of is the TRAMP system (Meier 2000), which extends PROVERB by using refutation graphs to represent assertion applications. However, TRAMP uses refutation graphs merely as a form of presentation. In our approach, we exploit graph features for explanations as well as for inconsistency detection.

Another interesting piece of work is (Horrocks and Voronkov 2006). The authors claim that first-order theorem provers are not efficient for reasoning with ontologies based on DL compared to specialized DL reasoners. However, a carefully adapted first-order theorem prover can be used to reason with large ontologies of more expressive ontology languages, which DL reasoners cannot handle. Their experiments show that the performance of a theorem prover can be greatly improved for query answering and consistency checking, but it is not very effective at proving satisfiability. They also discuss the need of providing explanations for both query answering and inconsistencies, and suggest that the proof format in which each inference is displayed separately is easier to read and understand. Their work sheds the light on the usage of resolution proof for explanations in the context of DL from a practical point of view. Moreover, if a first-order theorem prover is used to solve the problems when DL reasoners fail, our approach can still be applied to provide an explanation.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a framework which uses resolution proofs to provide explanations for DL reasoning, and demonstrated its applicability. We developed a sound and complete algorithm for explaining unsatisfiability and inconsistency queries w.r.t TBoxes/ABoxes in \mathcal{ALCHI} . We identified patterns of unsatisfiability and inconsistency and used them to improve the quality of explanations. This method is extensible in that newly found frequent patterns can be added in the future, if desired. We have developed a running prototype system which implements our algorithm. The explanations provided in this paper are generated by this system. A main advantage of our resolution based solution is that it is independent of a particular DL reasoner used. This makes it applicable to any DL reasoner, since it will not require changes in the internal details of such reasoners which often employ sophisticated optimization techniques and in general may result in producing answers that are not comprehensible for human users.

This research raises the following issues for future work.

1. The underlying DL language is restricted to \mathcal{ALCHI} and extensions to more expressive DL languages are needed. Since most resolution based automated theorem provers use the equality predicate to express number restrictions, it might become a problem for explanations in our context. As the number of literals grows exponentially with the actual numbers, FOL formulas or clauses will become cumbersome, which may degrade the performance. One possible solution is to shift to DL reasoners to handle number restrictions. Besides, more sophisticated translations and resolution technique should be also used. Theoretically there is a resolution decision procedure based on the use of a

- particular selection function for *SHIQ*, which is *ALCHI* extended to include number restrictions and transitive roles. It can decide satisfiability in ExpTime. Practically, when a relevant-only translation is used, as shown in (Tsarkov and Horrocks 2003), the performance of Vampire, a FOL theorem prover, is comparable with that of FaCT++ (Tsarkov and Horrocks 2006), especially in the context of positive (unsatisfiability) test.
2. Currently the explanation is restricted to unsatisfiability and inconsistency queries. Although we believe explanations for this kind of queries are more often needed by users, providing explanations for subsumption, satisfiability and non-subsumption queries is also necessary. As pointed out in (McGuinness and Borgida 1995), an explanation can be generated by returning a model or a counter-example. More work needs to be done to identify most suitable model or counter-example as an explanation.
 3. So far we have focused on explaining semantic defects in the knowledge base. When the problem is identified and understood, the next step to take is to resolve it. Usually some axioms or parts of the axioms should be removed and often there are a number of possible alternative remedies that can correct the defect. It would be interesting to develop methods to select a remedy that can mostly preserve the desirable properties of the original knowledge base.

ACKNOWLEDGMENTS

This work was supported in part by Natural Sciences and Engineering Research Council (NSERC) of Canada, by Genome Quebec, and by the ENCS Faculty, Concordia University. We also thank anonymous reviewers for their helpful comments.

REFERENCES

- BAADER, F. AND W. NUTT. 2003. Basic description logic. *In* The Description Logic Handbook: Theory, Implementation, and Applications. *Edited by* F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider. Cambridge University Press. 5–44.
- BORGIDA, A., E. FRANCONI, I. HORROCKS, D. L. MCGUINNESS, AND P. F. PATEL-SCHNEIDER. 1999. Explaining *ALC* subsumption. 37–40.
- BRACHMAN, R. J., D. L. MCGUINNESS, P. F. PATEL-SCHNEIDER, AND L. A. RESNICK. 1990. Living with CLASSIC: when and how to use a KL-ONE-like language. *In* Principles of semantic networks. *Edited by* J. Sowa. Morgan Kaufmann. San Mateo, US. 401–456.
- BYRD, L.. 1980. Understanding the control flow of prolog programs, 127–138.
- DENG, X., V. HAARSLEV, AND N. SHIRI. 2005. A framework for explaining reasoning in description logics. *In* Proceedings of the AAAI Fall Symposium on Explanation-aware Computing. AAAI Press. 189–204.
- DENG, X., V. HAARSLEV, AND N. SHIRI. 2006. Resolution based explanations for reasoning in the description logic *ALC*. *In* Proceedings of the Canadian Semantic Web Working Symposium. Springer. 55–61.
- EISINGER, N.. 1991. Completeness, Confluence, and Related Properties of Clause Graph Resolution. Morgan Kaufmann Publishers Inc.
- FELTY, A. AND D. MILLER. 1987. Proof Explanation and Revision. Technical Report MS-CIS-88-17. University of Pennsylvania.
- GANZINGER, H. AND DE H. NIVELLE. 1999. A superposition decision procedure for the guarded fragment with equality. *In* Proceedings of the 14th Annual IEEE Symposium on Logic in Computer Science (LICS '99). IEEE Computer Society. 295.

- HAARSLEV, V. AND R. MÖLLER. 2001. Racer system description. *In* Proceedings of International Joint Conference on Automated Reasoning (IJCAR 2001). *Edited by* T. N. R. Gori, A. Leitsch. Springer-Verlag. 701–705.
- HAASE, P. AND L. STOJANOVIC. 2005. Consistent evolution of owl ontologies. *In* Proceedings of the Second European Semantic Web Conference. *Edited by* A. Gmez-Prez and J. Euzenat. Springer. Heraklion, Greece. 182–197.
- HORROCKS, I.. 1998. The FaCT system. *In* Proceedings of the 2nd International Conference on Analytic Tableaux and Related Methods (TABLEAUX'98). *Edited by* H. de Swart. Springer-Verlag. 307–312.
- HORROCKS, I. AND A. VORONKOV. 2006. Reasoning support for expressive ontology languages using a theorem prover. *In* Proceedings of the Fourth International Symposium on Foundations of Information and Knowledge Systems (FoIKS). Springer. 201–218.
- HUANG, X.. 1994. Reconstructing proofs at the assertion level. *In* Proceedings of the 12th International Conference on Automated Deduction (CADE-12). *Edited by* A. Bundy. 738–752.
- HUANG, X.. 1996. Translating machine-generated resolution proofs into nd-proofs at the assertion level. *In* Proceedings of the 4th Pacific Rim International Conference on Artificial Intelligence: Topics in Artificial Intelligence. *Edited by* N. Y. Foo and R. Goebel. Springer. 399–410.
- HUANG, X. AND A. FIEDLER. 1996. Presenting machine-found proofs. *In* Proceedings of the 13th International Conference on Automated Deduction (CADE-13). *Edited by* M. A. McRobbie and J. K. Slaney. Springer. New Brunswick, NJ, USA. 221–225.
- HUSTADT, U., B. MOTIK, AND U. SATTLER. 2005. Data complexity of reasoning in very expressive description logics. *In* Proceedings of Nineteenth International Joint Conference on Artificial Intelligence (IJCAI 2005). 466–471.
- MCGUINNESS, D.. 1996. Explaining Reasoning in Description Logics. Ph.D. thesis. Rutgers University. New Brunswick, New Jersey.
- MCGUINNESS, D. L. AND A. BORGIDA. 1995. Explaining subsumption in description logics. *In* Proceedings of the tenth International Joint Conference on Artificial Intelligence (IJCAI'95). 816–821.
- MEIER, A.. 2000. TRAMP: Transformation of Machine-Found Proofs into Natural Deduction Proofs at the Assertion Level. *In* Proceedings of the 17th Conference on Automated Deduction (CADE-17). *Edited by* D. McAllester. Springer Verlag, Berlin, Germany. Pittsburgh, USA. 460–464.
- NONNENGART, A., G. ROCK, AND C. WEIDENBACH. 1998. On Generating Small Clause Normal Forms. *In* Proceedings of the 15th International Conference on Automated Deduction. *Edited by* C. Kirchner and H. Kirchner. Springer-Verlag. 397–411.
- PARSIA, B., A. KALYANPUR, AND E. SIRIN. 2005a. Black box techniques for debugging unsatisfiable concepts. *In* Proceedings of 2005 International Workshop on Description Logics (DL2005).
- PARSIA, B., E. SIRIN, AND A. KALYANPUR. 2005b. Debugging owl ontologies. *In* Proceedings of the 14th International World Wide Web Conference (WWW 2005). ACM Press. 633–640.
- PATEL-SCHNEIDER, P. F. AND B. SWARTOUT. 1993. Description-Logic Knowledge Representation System Specification.
- ROBINSON, J. A. AND A. VORONKOV (eds.). 2001. Handbook of Automated Reasoning (in 2 volumes). Elsevier and MIT Press.
- SCHLOBACH, S. AND R. CORNET. 2003. Non-standard reasoning services for the debugging of description logic terminologies. *In* Proceedings of the eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03). Morgan Kaufmann. 355–362.
- SHMUELI, O. AND S. TSUR. 1990. Logical diagnosis of ldl programs. *In* Logic Programming: Proceedings of the Seventh International Conference. *Edited by* D. H. D. Warren and P. Szeredi. MIT Press. Cambridge, MA. 112–129.
- TSARKOV, D. AND I. HORROCKS. 2003. DL reasoner vs. first-order prover. 152–159.

- TSARKOV, D. AND I. HORROCKS. 2006. FaCT++ description logic reasoner: System description. *In* Proceedings of the International Joint Conference on Automated Reasoning (IJCAR 2006). Springer. Seattle, Washington, USA. 292–297.
- WICK, M. R. AND W. B. THOMPSON. 1992. Reconstructive expert system explanation, *Artificial Intelligence* **54(1-2)**: 33.
- WOSMcCUNE, M. AND WOS.L. 1997. Otter - the cade-13 competition incarnations, *Journal of Automated Reasoning* **18(2)**: 211.