Resolution Based Explanations for Reasoning in the Description Logic $\mathcal{ALC}$

Xi Deng, Volker Haarslev, and Nematollaah Shiri

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

Abstract. With the increasing number of applications of description logics (DLs), unsatisfiable concepts and inconsistent terminologies become quite common, especially when the knowledge bases are large and complex. Even for an experienced knowledge engineer, it can be extremely difficult to identify and resolve the origins of these unsatisfiabilities and inconsistencies. Thus it is crucial to provide services to explain how and why a result is derived. Motivated by the possibilities of applying resolution technique in first-order logic to construct explanations for description logics, we extend our previous work and present an algorithm that generates explanations for unsatisfiability and inconsistency reasoning in the description language $\mathcal{ALC}$. The main advantage of our approach is that it is independent of any specific DL reasoners.

1 Introduction

In recent years, description logics (DLs) have found a wide range of applications in computer science, such as domain modeling, software engineering, configuration, and the semantic web [3]. With increasing complex applications, unsatisfiability and inconsistency become quite common. For example, the DICE (Diagnoses for Intensive Care Evaluation) terminology [17] 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 in the terminology or changes in the process of ontology evolution. However, existing DL reasoners, such as Racer [7] and FaCT [10], do not provide explanation services; they merely provide "Yes/No" answer

to a satisfiability or consistency query with no information about the reasons. In addition to such answers, it is desirable that DL reasoners also provide reasons for their answers and identify the sources of inconsistencies to further help knowledge engineers and ontology developers. It is therefore crucial and also challenging to provide explanation services as a useful feature and facility for DL reasoners.

In our previous work [4], we proposed a framework of constructing explanations for the description logic language $\mathcal{ALC}$ using resolution proofs. The approach works as follows:

1. Firstly, if a DL reasoner provides a negative answer to a satisfiability/ consistency query, i.e., a concept is unsatisfiable or a TBox/ABox is inconsistent, the axioms and assertions in the knowledge base will be translated into first-order formulae.
2. Then a resolution based automated theorem prover (ATP) is used to generate the resolution proof, taking the translated formulae as inputs.
3. At last, the resolution proof is transformed into its corresponding refutation graph [6]. Our algorithm traverses the graph and "reads" the proof to generate explanations. Later, the clauses involved in each traversal step are traced back to the contributing axioms/assertions in the original DL knowledge base.

Our approach has two main advantages. The first is that it is independent of any specific DL reasoners. Most implemented DL reasoners use tableau algorithms as the underlying reasoning calculus. Tableau rules are designed to render the results faster but not necessarily easier for the users to understand. Furthermore, some DL optimization techniques, such as absorption[1], are adopted to make reasoning more efficient, however they may make it more difficult for general users to understand if presented as explanations. In order to give explanations, the internal reasoning procedures should be tailored with performance penalties. Since the explanations are constructed based on resolution proofs in our approach, no modification of the internal of a DL reasoner is needed. This makes it possible to provide explanations for any DL reasoner. The second advantage of our approach of using resolution, compared to natural deduction proofs or tableau proofs, is that it is more focused, as all the literals in the clauses involved in a proof contribute to the proof. In other words, the resolution technique filters and excludes from a proof, the axioms and assertions in the knowledge base that are irrelevant to the query and hence unused in the process.

[1] The basic idea of absorption is to transform a general axiom, e.g., $C \sqsubseteq D$, to the form of a primitive definition $A \sqsubseteq D'$, where A is an atomic concept name and D' is a non-atomic concept.

Since for our explanation, we use a proof that is different from the original proof, a question that may naturally arise at this point is concerned with correctness of the explanation procedure. In this paper, our focus is to study soundness and completeness of our algorithm. In order to guarantee termination of the resolution procedure and hence our explanation technique, we adopt a structural transformation during translation.

The rest of the paper is organized as follows. Sect.2 discusses related work in explanations. In Sect.3, we introduce our explanation algorithm and establish its soundness and completeness. Sect.4 includes an illustrative example. Sect.5 includes concluding remarks and discusses some future work.

2 Related Work

There have been several proposals to provide explanations for DL reasoning. The earliest work is [13] which provides an explanation facility for subsumption and non-subsumption reasoning in CLASSIC [2]. CLASSIC is a family of knowledge representation systems based on description logics. It allows universal quantification, conjunction and restricted number restrictions. Since disjunction is not allowed in CLASSIC, explanations are given based on structural subsumption comparisons. Lengthy explanations are decomposed into smaller steps and a single step explanation is followed by more detailed explanations. This work is extended in [1] 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 the behavior of human reasoning. In contrast to these works, [17] provides algorithms to pinpoint unsatisfiable concepts and related axioms. This approach first excludes axioms which are irrelevant to the inconsistency and then provide simplified definitions which highlight the exact position of the contradiction. This work is extended in [16] to debug OWL ontologies. This approach consists of two parts: 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 source of inconsistency. Black box approach uses reasoners as oracles and relies on the users to perform navigational search to show unsatisfiability dependency. On the other hand, most existing explanation facilities in resolution based automated theorem proving transform the proofs into natural language style explanations [11, 12, 14]. They are specifically designed to solve problems in theorem proving, particularly mathematical theorems. They focus on proving conclusions using theorems, lemmas and premises and in general not suitable for indirect proofs.

3 Preliminaries

3.1 Description Logics

Description logics are a family of concept-based knowledge representation formalisms. It represents 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.

Among DL frameworks, $\mathcal{ALC}$ ($\mathcal{AL}$ stands for *Attribute language* and $\mathcal{C}$ stands for *Complement*) has been considered as a basic DL language of interests in numerous studies in DL. In $\mathcal{ALC}$ and other DL languages as well, basic descriptions are *atomic concepts*, designated by unary predicates, and *atomic roles*, designated by binary predicates to express relationships between concepts. Arbitrary concept descriptions such as *C* and *D* are built from atomic concepts and roles recursively according to the following rules:

$$
\begin{array}{rl}
C, D \rightarrow A| & \text{(atomic concept)} \\
\neg C| & \text{(arbitrary concept negation)} \\
C \sqcap D| & \text{(intersection)} \\
C \sqcup D| & \text{(union)} \\
\forall R.C| & \text{(value restriction)} \\
\exists R.C & \text{(existential quantification)}
\end{array}
$$

where *A* denotes an atomic concept and *R* denotes an atomic role. The *intersection* (or *union*) of concepts, which is denoted $C \sqcap D$ (or $C \sqcup D$), is used to restrict the individuals to those that belong to both *C* and *D* (or either *C* or *D*). The *value restriction*, denoted $\forall R.C$, requires that all the individuals that are in the relationship *R* with an individual of the value restriction belong to the concept *C*. The *existential quantification*, written $\exists R.C$, defines that for all individuals of the existential quantification there must exist an individual in the relationship *R* that belongs to the concept *C*. The *universal concept* $\top$ is a synonym of $A \sqcup \neg A$. The *bottom concept* $\bot$ is a synonym of $A \sqcap \neg A$.

An interpretation $\mathcal{I}$ defines a formal semantics of concepts and individuals in $\mathcal{ALC}$. It consists of a non-empty set $\Delta^{\mathcal{I}}$, called the domain of the interpretation, and an interpretation function, which maps every atomic

concept A to a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, 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 $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. The interpretation $\mathcal{I}$ is extended to concept descriptions, as shown in Table 1.

Constructors	Semantics
A	$A^{\mathcal{I}}$
$\neg C$	$\Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}$
$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
$\forall R.C$	$\{a \in \Delta^{\mathcal{I}} \mid \forall b.(a,b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\}$
$\exists R.C$	$\{a \in \Delta^{\mathcal{I}} \mid \exists b.(a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}$
$a : A$	$a^{\mathcal{I}} \in A^{\mathcal{I}}$
$(a,b) : R$	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$

Table 1. Interpretations of constructors in $\mathcal{ALC}$.

Axioms express how concepts and roles are related to each other. Generally, an axiom is a statement of the form $C \sqsubseteq D$ or $C \equiv D$, where *C* and *D* are concept descriptions. An interpretation $\mathcal{I}$ satisfies $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. It satisfies $C \equiv D$ if $C^{\mathcal{I}} = D^{\mathcal{I}}$.

The basic inference services in TBoxes include satisfiability, subsumption, equivalence, and disjointness. A concept in a TBox *T* is said to be *satisfiable* w.r.t *T* if there exists an interpretation $\mathcal{I}$ that is a model of *T*. A model for *T* is an interpretation that satisfies it. The other three inference services can all be reduced to (un)satisfiability. Another important reasoning service in TBoxes is to check whether a TBox *T* is *consistent*, i.e., whether there exists a model for *T*. The basic reasoning tasks in ABoxes are instance check, realization, and retrieval. The instance check verifies if a given individual is an instance of a specified concept. The instance realization computes the most specific concepts that an individual is an instance of. The instance retrieval returns all the individuals in the knowledge base that are instances of a given concept. An ABox *A* is *consistent* w.r.t a TBox *T*, if there is an interpretation that is a model of both *A* and *T*. Similar to the inference services in TBoxes, the other three inference services in ABoxes can also be reduced to the consistency problem of ABoxes. Further details of description logics can be found in [3].

3.2 Resolution

We assume that the readers are familiar with standard definitions of first-order logic (FOL) and clausal theorem proving. Resolution is one of the most widely used calculi for theorem proving in first-order logic. Resolution proves a theorem by negating the statement to be proved and adding this negated goal to the sets of axioms that are known to be true. It then uses the following inference rules to show that this leads to a contradiction.

Positive factoring:

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

where σ = MGU(A, B),

Resolution:

$$\frac{C \vee A \quad D \vee \neg B}{C\sigma \vee D\sigma}$$

where σ = MGU(A, B).

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.

4 Preprocessing

In [4], the translation between DL and FOL is straightforward based on the semantics of DL. For $\mathcal{ALC}$, concepts can be translated into the first order predicate logic over unary and binary predicates with two variables, say *x*, *y*, which is denoted as $\mathcal{L}^2$. Table 2 shows such a translation from $\mathcal{ALC}$ 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* can be translated into binary predicate $\phi_R(x, y)$. An individual name *a* is translated into a constant *a*. However, in order to guarantee complete and terminating reasoning, a simple direct translation based on the standard first order logic semantics is not appropriate. Hence we adopt the structural transformation as shown in [8, 15].

DL Constructor	FOL Formula
A	$\phi_A(x)$
$\neg C$	$\neg\phi_C(x)$
$C \sqcap D$	$\phi_C(x) \wedge \phi_D(x)$
$C \sqcup D$	$\phi_C(x) \vee \phi_D(x)$
$C \sqsubseteq D$	$\forall x(\phi_C(x) \rightarrow \phi_D(x))$
$R \sqsubseteq S$	$\forall x(\phi_R(x) \rightarrow \phi_S(x))$
$\exists R.C$	$\exists y(\phi_R(x,y) \wedge \phi_C(y))$
$\forall R.C$	$\forall y(\phi_R(x,y) \rightarrow \phi_C(y))$
$a : A$	$\phi_A(a)$
$(a,b) : R$	$\phi_R(a,b)$

Table 2 . Translation from $\mathcal{ALC}$ into $\mathcal{L}^2$

The structural transformation is a kind of conjunction normal form transformation of first-order predicate logic formulae by replacing the subformulae with some new predicates and adding a suitable definition for these predicates. We choose the structural transformation because: Firstly, it would avoid the exponential blow up of the size of the clauses. Consider the axiom $E \sqsubseteq F$, where *E* and *F* are complex concept descriptions. If *n* is the number of clauses generated by *E* and *m* is the number of clauses generated by *F* then the above formula generates *n*×*m* clauses. The reason for the exponential explosion is the duplication of subformulae obtained by the exhaustive application of the distributivity law. 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*. Secondly, it helps to preserve original structures of DL axioms after first-order logic formulae are transformed into their conjunction normal forms. Consider the axiom $\forall R.E \sqsubseteq \exists S.F$, without this transformation, the LHS and RHS of this axiom are distributed into four clauses, making it difficult to generate explanations.

A formal definition of the transformation is shown below.

Definition 1 *C* is a *qualified concept expression* if and only if *C* is of the form $\oplus R.D$ with $\oplus \in \{\forall, \exists\}$ and *D* is an arbitrary concept.

Definition 2 [15] A *position* is a word over the natural numbers. The set $pos(\varphi)$ of positions of a given formula φ is defined as follows: (i) the empty word $\varepsilon \in pos(\varphi)$ (ii) for $1 \leq i \leq n$, $i.p \in pos(\varphi)$ if $\varphi = \varphi_1 \bullet \ldots \bullet \varphi_n$ and $p \in pos(\varphi_i)$ where $\bullet$ is a first-order operator. If $p \in pos(\varphi), \varphi|_{i.p} = \varphi_i|_p$ where $\varphi = \varphi_1 \bullet \ldots \bullet \varphi_n$. 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 π in a formula φ 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 3 Let φ be a formula and $\phi = \varphi|_\pi$ be a subformula of φ at position π. For position q which is just one position below π, the DL counterpart of $\varphi|_q$ is a qualified concept expression. Let $x_1, ..., x_n$ be the free variables in ϕ and let R be a new predicate. Then the formula

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

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

$$Def_\pi^\varphi = \quad \forall x_1, ..., x_n [R(x_1, ..., x_n) \rightarrow \phi] \text{ if } Pol(\varphi, \pi) = 1$$
$$\forall x_1, ..., x_n [\phi \rightarrow R(x_1, ..., x_n)] \text{ if } Pol(\varphi, \pi) = -1$$

It is easy to see the following result after structural transformation.

Definition 4 There are four types of clauses after normalization:

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

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

Specifically, clause type (1) is translated from axiom $C \sqsubseteq D$, and both C and D are complex concepts. Type (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 $\exists R.C \sqsubseteq D$.

The correctness of the translation is proved as follows.

Theorem 1 Let T be a TBox in $\mathcal{ALC}$ and C be a named concept in T. T is consistent. Let $\theta(T)$ and $\theta(C(a))$ be the resulting set of FOL formulae of T and $C(a)$ after the translation, a being a newly introduced individual. Then C is unsatisfiable if and only if the empty clause is derived under resolution given $\theta(T) \bigcup \theta(C(a))$.

Proof. As mentioned in [8], the structural transformation does not affect satisfiability, it is easy to see T and $\theta(T)$ are equisatisfiable. Since T is consistent as the known fact, $\theta(T)$ is also consistent. Since C is unsatisfiable, C does not admit any instance, i.e., $C(a)$ is false. Hence $\theta(T) \bigcup \theta(C(a))$ is inconsistent. According to the refutational completeness of resolution, the empty clause can be derived. Similarly, we can prove that if the empty clause is derived for $\theta(T) \bigcup \theta(C(a))$, C is unsatisfiable. ■

We can also easily prove the following result.

Theorem 2 Let T be a TBox and A be an ABox (either T or A can be empty). Let $\theta(T)$ and $\theta(A)$ be the resulting set of FOL formulae of T and A after the translation. Then $T \bigcup A$ is inconsistent if and only if the empty clause is derived under resolution given $\theta(T) \bigcup \theta(A)$.

5 The Algorithm to Generate Explanations

Our approach uses a refutation graph [6] to reconstruct the resolution proof in order to support explanation. Generally speaking, a refutation graph is a graph whose nodes are literals (grouped together in clauses) and its edges connect complementary nodes/literals which correspond to the resolution steps in the resolution proof. In a refutation graph, complementary literals between input clauses are directly visible. We give the fundamental definition about refutation graphs as below. Further details can be found in [4]. The algorithm of transforming a resolution proof to its refutation graph is shown in Fig.1.

Definition 5 A *refutation graph* is a quadruple $G = (L; C; M_L; K)$, where L is a finite set of literal nodes in G. C is a partition of the set of literal nodes. Its members are clause nodes in G. M_L is a mapping from L to a set of literals. The set of links K is a partition of a subset of L. There are no pure literal nodes in a refutation graph, i.e., every literal node belongs to some link in K.

Input: a resolution proof **Output**: its corresponding refutation graph

For all the steps in the resolution proof
  For all the literals that are involved in a step
    If its literal node does not exist
      create its corresponding clause (literal) node and add it into the refutation graph
    add a link between the literal nodes that are resolved (factored) together
Return the refutation graph

Fig. 1. The Transformation Algorithm.

The main idea of explanations based on the refutation graph is to start from a literal node (or nodes) and traverse the graph. After the traversal is completed, each clause node involved in each step is translated into an entry in an explanation list consisting of its source axioms in DL. After some clean-up, e.g., deleting duplicate line, this explanation list can be further transformed into natural language style explanations.

The traversal algorithms of unsatisfiability reasoning can be described as follows: Start from the literal node corresponding to the unsatisfiable concept, follow the links to its complementary literal nodes L_i, $i = 1, ..., n$. For each of the literal nodes that are in the same clause node as L_i, follow its untraversed link. Stop when there is no untraversed link left. The algorithm to explain the inconsistency reasoning is similar to the unsatisfiability case, except that the traversal will begin with one of the literal nodes involved in the first step of the resolution proof.

The pseudo code of the algorithm is shown in Fig.2. It uses a stack, called SOT, which includes the literal nodes which are yet to be traversed.

Input: a refutation graph **Output**: an explanation list

If it is an unsatisfiable problem
  start from the unsatisfiable concept *C*
  else start from a concept *C* involved in the first step of the resolution proof
SOT ← the associated literal node of *C*
For all the literal nodes $\boldsymbol{L_i}$ in SOT
  mark $\boldsymbol{L_i}$ as "traversed"
  put the corresponding DL axiom of $\boldsymbol{L_i}$ into the explanation list
  For all the links that are adjacent to $\boldsymbol{L_i}$
   If the link was created from a factoring step
      mark the literal node at the opposite side of the link as "traversed"
   else
   For all the literal nodes $\boldsymbol{L_k}$ that in the same clause node as the opposite side of the link
      SOT ← $\boldsymbol{L_k}$
  Remove $\boldsymbol{L_i}$ from SOT
Return the explanation list

Fig. The Traversal Algorithm.

Theorem 3 The unsatisfiability and inconsistency traversal algorithms are complete and can terminate with an explanation.

Proof. Termination: In each step of the traversal, we decrease the number of literal nodes that remain untraversed, since once a literal node is traversed, it will not be traversed again. As the number of literal nodes in a refutation graph is finite, the traversal algorithm will terminate.

Completeness: The completeness in our case means that at the end of the algorithm, no literal node is left untraversed. That we cannot reach a blocked situation follows from 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. ■

6 Example

To help understand the algorithms, we show an example KB as follows:

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

After being fed into Racer, HappyPerson is reported to be unsatisfiable. We set KB to be as below and show it to be inconsistent.

$$KB \equiv KB \cup \{HappyPerson(a)\}$$

where *a* is a fresh individual.

Since Axiom 1, 2, 3, 5 and 6 contain qualified concept expressions. But as the structural transformation does not either decrease the number of the clauses or simplify the explanations for 1, 3, 5 and 6, we only show how it is converted to FOL formulae based on structural transformation for axiom 2. We introduce new names Q for this subconcept and get

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

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

By applying unit resolution to this clause set and by converting the resolution proof to its refutation graph in our prototype system, we get the graph as shown in Fig.3.

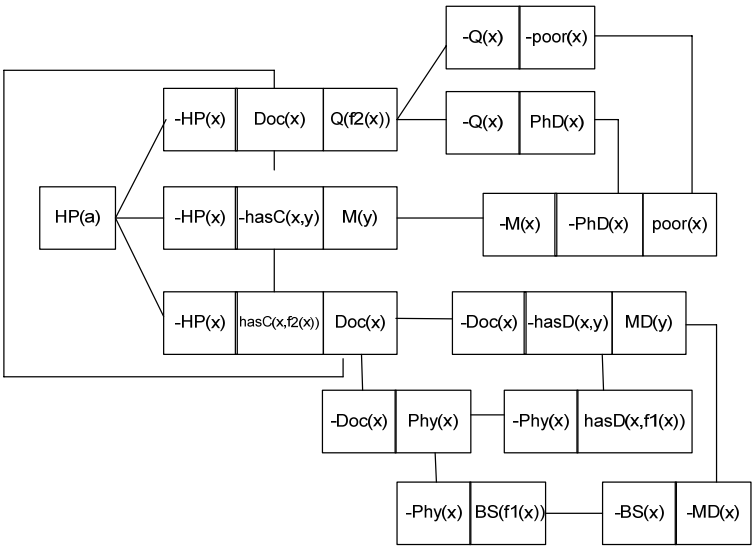

Fig.3. The Refutation Graph of the Example.

By applying the algorithm to explain unsatisfiable concepts, we get an explanation list as follows:

1. $HappyPerson(a)$
2. $HappyPerson \sqsubseteq Doctor \sqcup \exists hasChild.(PhD \sqcap \neg Poor)$
3. $Doctor \sqsubseteq \forall hasDegree.MD \sqcap Physician$
4. $Physician \sqsubseteq \exists hasDegree.BS$
5. $MD \sqsubseteq \neg BS$
6. $HappyPerson \sqsubseteq \forall hasChild.Married$
7. $PhD \sqcap Married \sqsubseteq Poor$

It reads as: 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 contradiction 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.

7 Conclusions and Future Work

In this paper, we presented a sound and complete algorithm based on resolution proofs for explaining DL reasoning. We have implemented a prototype system based on the algorithm to explain unsatisfiability and inconsistency queries w.r.t TBoxes/ABoxes in $\mathcal{ALC}$. This system uses Racer as the DL reasoner and Otter [18] as the resolution based ATP. As input the system accepts problem descriptions in the KRSS syntax. All the components are implemented in Java. As experiments show, our approach is suitable for small knowledge bases but, without any further refinements, it would generate long and complex explanations. Besides, although there is a resolution decision procedure based on the use of a particular selection function for $\mathcal{ALC}$ [5], which can decide satisfiability in ExpTime, with large and complicated examples, first-order resolution based provers will choke on such input, especially when considering to extend $\mathcal{ALC}$ to include number restrictions or transitive roles. To ensure that the first-order prover will finish, more sophisticated translations and resolution technique should be used.

Acknowledgments

This work was supported in part by Natural Sciences and Engineering Research Council (NSERC) of Canada, by Genome Quebec, and by ENCS Concordia University.

References

1. Borgida A, Franconi E, Horrocks I, McGuiness DL and Patel-Schneider PF (1999) Explaining $\mathcal{ALC}$ subsumption. In Proceedings of 1999 International Workshop on Description Logics. CEUR-WS.org, pp 37-44
2. Brachman RJ, McGuiness DL, Patel-Schneider PF, Resnick LA (1990) Living with CLASSIC: when and how to use a KL-ONE-like language. In Sowa J (ed) Principles of semantic networks. Morgan Kaufmann, San Mateo, US, pp 401--456
3. Baader F, Nutt W (2003) Basic description logic. In Baader F, Calvanese D, McGuinness DL, Nardi D, and Patel-Schneider PF (eds) The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, Cambridge, pp 5-44

4. Deng X, Haarslev V, Shiri N (2005) A framework for explaining reasoning in description logics. In Proceedings of the AAAI Fall Symposium on Explanation-aware Computing. AAAI Press, Menlo Park, US, pp 55-61
5. De Nivelle H, De Rijke M (2003) Deciding the guarded fragments by resolution. J Symbolic Computation 35(1):21-58
6. Eisinger N (1991) Completeness, confluence, and related properties of clause graph resolution. Morgan Kaufmann San Francisco, US
7. Haarslev V, Möller R (2001) Racer system description. In T Nipkow R Gori, A Leitsch (eds) Proceedings of International Joint Conference on Automated Reasoning (IJCAR). Springer-Verlag, London, UK, pp 701-705
8. Hustadt U, Motik B, Sattler U (2005) Data complexity of reasoning in very expressive description logics. In Proceedings of Nineteenth International Joint Conference on Artificial Intelligence (IJCAI). Morgan Kaufmann, San Francisco, US, pp 466-471
9. Hustadt U, Schmidt RA (1999) Issues of decidability for description logics in the framework of resolution. In Caferra R and Salzer G (eds) Automated Deduction in Classical and Non-Classical Logics. Springer-Verlag, London, UK, pp 191-205
10. Horrocks I (1998) The FaCT system. In De Swart H (ed) Automated Resoning with Analytic Tableaux and Related Methods: International Conference Tableaux. Springer-Verlag, Berlin, Germany, pp 307-312
11. Huang X (1994) Reconstructing proofs at the assertion level. In Bundy A (ed) Proceedings of 12th Conference on Automated Deduction. Springer-Verlag, London, UK, pp 738-752
12. Lingenfelder C (1996) Transformation and structuring of computer generated-proofs. Ph.D. thesis, University of Kaiserslautern
13. McGuinness DL and Borgida A (1995) Explaining subsumption in description logics. In Proceedings of the tenth International Joint Conference on Artificial Intelligence (IJCAI). Morgan Kaufmann, San Francisco, US, pp 816-821
14. Meier A (2000) TRAMP: Transformation of machine-found proofs into natural deduction proofs at the assertion level. In McAllester D (ed) Proceedings of the 17th Conference on Automated Deduction (CADE). Springer-Verlag, Berlin, Germany, pp 460-464
15. Nonnengart A, Rock G, Weidenbach C (1998) On generating small clause normal forms. In Kirchner C and Kirchner H (eds) Proceedings of the 15th International Conference on Automated Deduction. Springer-Verlag, London, UK, pp 397-411
16. Parsia B, Sirin E, Kalyanpur A (2005) Debugging OWL ontologies. In The 14th Interntional World Wide Web Conference (WWW). ACM Press, New York, US, pp 633-640
17. Schlobach S and Cornet R (2003) Non-standard reasoning services for the debugging of description logic terminologies. In Proceedings of the eighteenth International Joint Conference on Artificial Intelligence (IJCAI). Morgan Kaufmann, San Francisco, US, pp 355-362
18. WosMcCune M and Wos L (1997) Otter - the CADE-13 competition incarnations. J Automated Reasoning 18(2):211-220