EXTRACTING CASE-BASED ANSWERS FROM CLOSED PROOF-TREES

Isabel Gomes Barbosa and Newton José Vieira
Universidade Federal de Minas Gerais, Belo Horizonte, Minas Gerais, Brazil

Keywords: First-order logic, Disjunctive answers, Case-based answers, Proof-trees.

Abstract: A question of the form "Find X such that $P(X)$ is true" may produce in the most usual inference systems an answer that has the general form $P(T_1) \vee P(T_2) \vee \ldots \vee P(T_k)$. If we have $k \geq 2$, the answer is then termed a disjunctive answer. In some scenarios, a disjunctive answer of this form may be considered too imprecise to help the user in his activities. However, an answer which specifies the cases in which each element $P(T_i)$ is true would be a perfectly appropriate answer to the question. In this paper, we propose an algorithm that, from a deduction of $P(T_1) \vee P(T_2) \vee \ldots \vee P(T_k)$, $k \geq 2$, on the form of a proof-tree, extracts a case-based answer to the exact same question. A case-based answer is an answer given in terms of a finite number of cases, each one implying a non-disjunctive answer $P(T_i)$, $1 \leq i \leq k$, to the user's question.

1 INTRODUCTION

Given a knowledge base represented in a first-order language, a question of the form

Find X such that $P(X)$ is true,

may produce in the most usual inference systems an answer of the form

$$P(T_1) \vee P(T_2) \vee \ldots \vee P(T_k),$$

where each T_i is an n-tuple of ground terms (Green, 1969).

If we have $k \geq 2$, the answer is then termed a disjunctive answer. A disjunctive answer presents different possible answers to the question, at least one of which is true. Disjunctive answers are considered less informative than non-disjunctive answers, since they inform that the disjunction as a whole is true, but do not specify which elements of the disjunction are indeed correct answers to the question. This lack of information induces uncertainty in the answer. One way to reduce this uncertainty is to analyze the disjunctive answer by an exhaustive set of cases in the problem domain. I.e., we can split the problem into several (exhaustive) cases, where each case implies a non-disjunctive answer $P(T_i)$, $1 \leq i \leq k$. Cases $\alpha_1, \alpha_2, \ldots, \alpha_m$ are exhaustive if $\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_m$ is a logical consequence of the knowledge base. An incipient way to achieve this is shown in (Chang and Lee, 1997) in the context of resolution refutation.

As an example, adapted from (Chang and Lee, 1997), consider a knowledge base defined by the clauses below:

(1) $\mathsf{adult}(\mathsf{John}) \vee \mathsf{prescribe}(\mathsf{John},a)$
(2) $\neg\mathsf{adult}(\mathsf{John}) \vee \mathsf{prescribe}(\mathsf{John},b)$

and the question "What drug should be prescribed to John?", which logical form is "$\exists y$ $\mathsf{prescribe}(\mathsf{John},y)$". In most inference systems, the answer provided to this question would be the disjunctive answer

$$\mathsf{prescribe}(\mathsf{John},a) \vee \mathsf{prescribe}(\mathsf{John},b).$$

From this answer, we cannot tell which drug, a or b, should be prescribed to John. However, if instead were returned as an answer

$$\neg\mathsf{adult}(\mathsf{John}) \rightarrow \mathsf{prescribe}(\mathsf{John},a)$$
$$\mathsf{adult}(\mathsf{John}) \rightarrow \mathsf{prescribe}(\mathsf{John},b)$$

and the user knew whether John is an adult or not, then he would determine the drug to be prescribed to John. Note that in this example only two cases are considered: the case in which John is an adult and the case in which John is not.

The deduction of an answer carries with it information about the question, many of which are not available to users. Proof-trees (Vieira, 1987) are tree-like structures that can be used to represent the history of a proof. In this work, we consider the representation of proofs by proof-trees, like that used

Gomes Barbosa I. and José Vieira N. (2010).
EXTRACTING CASE-BASED ANSWERS FROM CLOSED PROOF-TREES.
In *Proceedings of the 2nd International Conference on Agents and Artificial Intelligence - Artificial Intelligence*, pages 377-384
DOI: 10.5220/0002736703770384

in Prolog implementations (Bruynooghe, 1982), however our notion of proof-trees is extended to make it complete for full first-order logic, in a way similar to that proposed in MESON system by Loveland (1978). The representation of a proof by proof-trees induces the reading of clauses as implications, allowing answer explanations independent of the steps actually done by the inference system (Vieira, 1987).

Thus, in this paper, we present an algorithm that, from a proof-tree that corresponds to a disjunctive answer of the form

$$P(T_1) \vee P(T_2) \vee \ldots \vee P(T_k),$$

$k \geq 2$, extracts a case-based answer, of the form

$$\begin{array}{l} \alpha_1 \rightarrow P(T_{i_1}) \\ \alpha_2 \rightarrow P(T_{i_2}) \\ \ldots \\ \alpha_m \rightarrow P(T_{i_m}) \end{array}$$

$1 \leq i_1, i_2, \ldots, i_m \leq k$, where $\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_m$ is a logical consequence of the knowledge base and α_j is a case that implies the answer $P(T_{i_j})$. Moreover, we assume that each T_i is a n-tuple of ground terms, and so the answer is a "specific answer" according to the classification of Burhans and Shapiro (2007). As far as we know, the presentation of a specific disjunctive answer in the above format is original. We are still working in the generalization of our method to the other classes of answers (generic and hypothetical answers).

We must, however, emphasize that the quality of the answers of our algorithm depends on the knowledge present in the proof-tree. In particular, if the knowledge is intrinsically disjunctive, then it may not be possible to obtain an answer that has the general form indicated above. For example, if you ask the question "What subject does John teach?", and the knowledge base contains the fact that "John teaches Mathematics or John teaches Logic", then from this fact it follows immediately the statement "John teaches some subject.". Obviously, a corresponding proof-tree does not contain information that would allows us to extract or construct cases that imply the answer "John teaches Mathematics" or "John teaches Logic". Even if the proof-free has sufficient knowledge for the extraction of an answer with the above format, a meaningful answer could depend on characteristics of the user not present in a proof-tree (typically, one would try to capture such characteristics in an user model). Any way, the non deterministic algorithm to be presented will not exclude any of the meaningful answers.

This work builds one additional step in the direction of making formal methods of reasoning more amenable to use in practice. It exhibits information already present in a proof but otherwise concealed from users. As a consequence, it makes a question-answering system more "collaborative" at practically no additional cost.

This paper is organized in the following way: section 2 presents the definition of proof-trees and its properties. Section 3 makes a brief discussion about information extraction from closed proof-trees and introduces an example that will be used to explain the algorithm proposed. The algorithm for extracting case-based answers from closed proof-trees is described in section 4. Finally, the conclusions of this work are presented in section 5.

2 PROOF-TREES

A proof-tree is essentially what in (Letz and Stenz, 2001) is defined as a clausal tableau. Proof-trees have its nodes labeled with literals. In our definition of proof-trees, we will use a linear notation. A proof-tree is an ordered pair $L\alpha$, such that L is a literal and α is a set (possibly empty) of literals and/or proof-trees. The literals M of every subtree $M\beta$ that occurs in $L\alpha$ are termed expanded literals. The remaining literals are divided into two groups: reduced and candidate literals. Candidate literals are literals that have yet to be proven. Expanded and reduced literals are literals already used in some inference rule of the proof procedure.

A proof-tree is defined inductively by the following inference rules. The notation $\overline{L}$ is used to refer to the complement of literal L, while $|L|$ is used to refer to the atom of literal L.

(Codification rule) The codification of an input clause $L_1 \vee L_2 \vee \ldots \vee L_m$ produces the proof-tree $\bot\{\overline{L}_1, \overline{L}_2, \ldots, \overline{L}_m\}$. Thus, the root of every proof-tree is the literal $\bot$ (*falsum*).

(Expansion rule) The expansion of a proof-tree $\bot\{\ldots L_1 \ldots\}$, where L_1 is a candidate literal, with an input clause $M_0 \vee M_1 \vee \ldots \vee M_n$ such that L_1 and M_0 are unifiable with most general unifier (mgu) σ, produces the proof-tree $\bot\{\ldots L_1\{\overline{M}_1, \overline{M}_2, \ldots, \overline{M}_n\} \ldots\}\sigma$.

(Reduction rule) The reduction of a proof-tree $\bot\{\ldots L_1\{\ldots L_2 \ldots\} \ldots\}$, where L_2 is a candidate literal of opposite sign to L_1 and $|L_1|$ and $|L_2|$ are unifiable with mgu σ, produces the proof-tree $\bot\{\ldots L_1\{\ldots \langle L_2 \rangle \ldots\} \ldots\}\sigma$.

Every deduction starts with the application of the codification rule, which produces the initial proof-tree. The expansion and reduction rules are similar to the extension and reduction rules of model elimination

(Loveland, 1969). Expanded and candidate literals are termed *A*-literals and *B*-literals in model elimination proof procedures. The structure provided by proof-trees is similar to that proposed by the MESON system of Loveland (1978).

A proof-tree without candidate literals is called a closed proof-tree. A finite set of clauses S is unsatisfiable iff there exists a closed proof-tree for S (Vieira, 1987). Thus, we can prove by contradiction that a formula is a logical consequence of a set of hypotheses, assuming, along with the hypothesis, the negation of the formula and obtaining a closed proof-tree from the resulting set.

Example 1. Consider the set of clauses $S1$:

1. $P(x) \vee R(x) \vee Q(x,y)$
2. $\neg Q(x,y) \vee S(x)$
3. $\neg S(x) \vee \neg Q(x,b)$
4. $\neg R(a)$

and the question "$S1 \models \exists x P(x)$?". Negating and producing clauses:

5. $\neg P(x)$

From 1 to 5 we generate the following refutation:

6. $\bot\{P(x)\}$ (cod 5)
7. $\bot\{P(x)\{\neg R(x), \neg Q(x,y)\}\}$ (exp 1)
8. $\bot\{P(a)\{\neg R(a)\{\}, \neg Q(a,y)\}\}$ (exp 4)
9. $\bot\{P(a)\{\neg R(a)\{\}, \neg Q(a,y)\{\neg S(a)\}\}\}$ (exp 2)
10. $\bot\{P(a)\{\neg R(a)\{\}, \neg Q(a,y)\{\neg S(a)\{Q(a,b)\}\}\}\}$ (exp 3)
11. $\bot\{P(a)\{\neg R(a)\{\}, \neg Q(a,b)\{\neg S(a)\{\langle Q(a,b)\rangle\}\}\}\}$ (red)

The closed proof-tree 11 is represented graphically in Fig. 1.

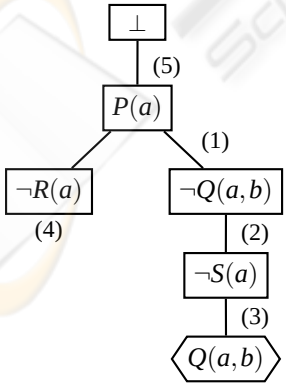

Figure 1: A closed proof-tree generated from the clauses 1-5.

Note that a proof-tree makes explicit a set of clause instances in the form of implications. For example, the proof-tree in Fig. 1 shows the following instances:

$$P(a) \rightarrow \bot \quad \text{(clause 5)}$$
$$\neg R(a) \quad \text{(clause 4)}$$
$$\neg R(a) \wedge \neg Q(a,b) \rightarrow P(a) \quad \text{(clause 1)}$$
$$\neg S(a) \rightarrow \neg Q(a,b) \quad \text{(clause 2)}$$
$$Q(a,b) \rightarrow \neg S(a) \quad \text{(clause 3)}$$

One can transform a proof-tree into an equivalent proof-tree where the codified clause is any clause used in expansions. Both trees are equivalent in the sense that they result from diferent proofs of the same theorem and lead to the same answer; more precisely, in terms of the Herbrand theorem, the set of insatisfiable instances is the same in both cases. That transformation, explained in (Vieira, 1987), is based on the fact that the formula

$$L_1 \wedge \ldots \wedge L_n \rightarrow M$$

is logically equivalent to n formulas

$$L_1 \wedge \ldots \wedge L_{i-1} \wedge L_{i+1} \wedge \ldots \wedge L_n \wedge \overline{M} \rightarrow \overline{L_i},$$

obtained by complementing and interchanging the literal M with each literal L_i in turn, and to the formula

$$L_1 \wedge \ldots \wedge L_n \wedge \overline{M} \rightarrow \bot.$$

Using these equivalences, the proof-tree of the last example could be transformed in order that the clause 2 appears as the codified clause, as shown in Fig. 2.

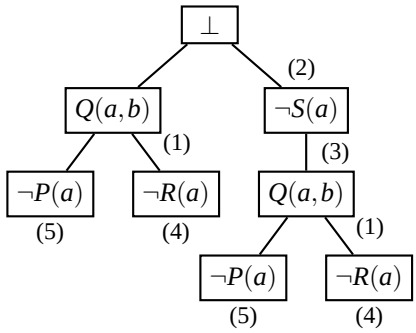

Figure 2: A proof-tree equivalent to that of Fig. 1.

In the next section we show how an answer can be extracted from a proof-tree.

3 ANSWER EXTRACTION

In this section we explain how disjunctive answers are extracted from a proof-tree. Afterward, in the next

section, we will show how to extract cases that justify each disjunct.

After a proof is found, we can extract values for the existentially quantified variables of the formula associated to the question from the instances of clauses of the negated question, $\neg P(X)$, that appears on the proof-tree. The questions considered in this work are those which result in disjunctive answers of the form $P(T_1) \vee P(T_2) \vee \ldots \vee P(T_k)$, $k \geq 2$. In proof-trees, disjunctive answers are caused by expansions with clauses derived from the negation of the question. Each instance of clause from the negation of the question that appears in the proof-tree gives us an element of the disjunction (Vieira, 1987).

The algorithm for extracting case-based answers will be applicable only when the question is represented by an atomic formula. This assumption does not restrict generality. If the question is a general formula $F(X)$, we add the clausal form of $\forall X\ (F(X) \rightarrow \rightarrow Q(X))$ to the knowledge base, where Q is a new predicate symbol, and the question is represented by the positive literal $Q(X)$. This strategy is followed in several works such as, for example, (Demolombe, 1992) and (Burhans and Shapiro, 2007).

Next, we present an example that will be used to demonstrate the application of the algorithm.

Example 2. Suppose a knowledge base that contains the following set of clauses, where $Q(x,n)$ means that *x* can borrow *n* books:

1. $Stud(x) \vee Staf(x) \vee Vis(x)$
   Students, staff and visitors are eligible to borrow books
2. $\neg Stud(x) \vee Und(x) \vee Grad(x)$
   Students are divided into under and graduate students
3. $\neg Stud(x) \vee \neg Und(x) \vee Q(x,4)$
   Undergraduate students can borrow 4 books
4. $\neg Stud(x) \vee \neg Grad(x) \vee Q(x,8)$
   Graduate students can borrow 8 books
5. $\neg Staf(x) \vee Acad(x) \vee Adm(x)$
   The staff is divided into academic and administrative
6. $\neg Staf(x) \vee \neg Acad(x) \vee Q(x,8)$
   The academic staff can borrow 8 books
7. $\neg Staf(x) \vee \neg Adm(x) \vee Q(x,2)$
   The administrative staff can borrow 2 books
8. $\neg Vis(x) \vee Q(x,4)$
   Visitors can borrow 4 books
9. $\neg Und(\mathrm{J})$ *John is not an undergraduate*

And the question "How many books can John borrow?", formulated as "Find *y* such that $Q(\mathrm{J},y)$". The clause from the negation of the question is:

10. $\neg Q(\mathrm{J},y)$

In Fig. 3, a closed proof-tree is given. From this proof-tree, we obtain the answer:

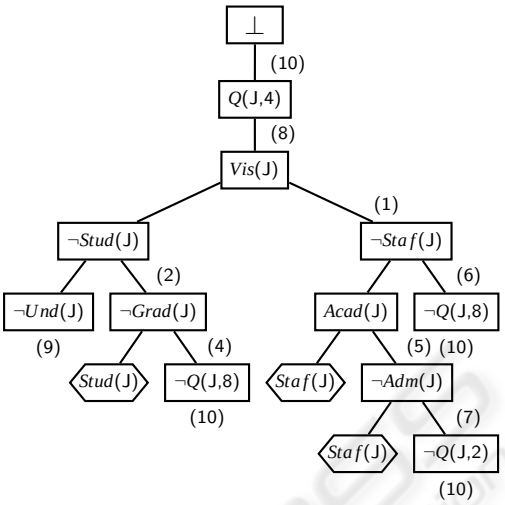

Figure 3: Proof-tree T_0 for Example 2.

$$Q(\mathrm{J},2) \vee Q(\mathrm{J},4) \vee Q(\mathrm{J},8).$$

This answer has the general form:

$$P(T_1) \vee P(T_2) \vee \ldots \vee P(T_k),\ k \geq 2.$$

The next section describes the algorithm for extracting cased-based answers from closed proof-trees and applies it to the proof-tree illustrated in Fig. 3.

4 ALGORITHM FOR CASE-BASED ANSWERS EXTRACTION

The algorithm begins from the deduction of $P(T_1) \vee P(T_2) \vee \ldots \vee P(T_k)$, $k \geq 2$, on the form of a proof-tree. We shall denote this proof-tree by T_0. The algorithm is divided into six steps, where each step has as input a tree T_i. The trees T_0 and T_1 are simply proof-trees. The other trees generated by the algorithm will be called answer-trees. The general purpose of the algorithm is to transform the initial proof-tree T_0 into a final answer-tree T_5, which corresponds to a case-based answer. Thus, from T_5, we can directly extract a cased-based answer to the user's question. Firstly, we define the notion of a tree T_i in the context of the algorithm.

Definition 1. Let Σ be a set of literals. A tree T_i over Σ is a 4-tuple (N,A,λ,r), where:

- N is a finite set of nodes;
- $A \subseteq (N \times N)$ is a set of arcs;
- λ is a function assigning a label to each node: $\lambda : N \rightarrow C$, where C is the set of (finite) conjunctions of literals from Σ. Usually, λ assigns a single

literal to a node. In the tree T_0, shown in Fig. 3, $\lambda(v)$ is the literal that appears inside each box;

- $r \in N$ is the root.

Def. 1 applies to both proof-trees and answer-trees, as they have exactly the same structure, though different readings.

Let Q be the predicate symbol of the question literal. If for all nodes v of tree T_0, except its root, $\lambda(v)$ has Q as its predicate symbol, then the algorithm does not apply: the knowledge used is intrinsically disjunctive. An example is: if the knowledge base contains $Q(a) \vee Q(b)$ and the question is "Find x such that $Q(x)$", an answer is $Q(a) \vee Q(b)$; there is no case-based answer.

Next, we describe the six steps of the algorithm. Each step is exemplified using the knowledge base and question from Example 2. The proof of correctness of the algorithm is not presented here due to the limit on the number of pages.

***Step* 1.** Transform T_0 into an *equivalent* (i.e., having the same instances) proof-tree T_1, where each node v of T_1 such that $\lambda(v)$ has Q as its predicate symbol is a *leaf*. In doing this, choose as the codified clause of T_1 (a) a clause without any literal having Q as its predicate symbol or (b) a tautological clause of the form $L \vee \overline{L}$, where L is a literal that doesn't have Q as its predicate symbol. The option (b) is the only option for the simple example seen in the Introduction. Fig. 4 shows T_0 and T_1 for that example.

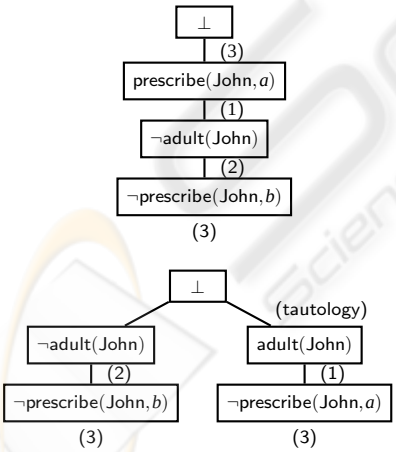

Figure 4: Trees T_0 and T_1 for the Introduction's example.

In the example of the previous section, the clause from the negation of the question is 10, which is the codified clause of Fig. 3. Taking the option (a), clauses 1, 2 or 5 can be chosen as the codified clause, because they don't have a literal with Q as predicate symbol. We transform the proof-tree T_0 of Fig. 3 into the equivalent proof-tree shown in Fig. 5, where clause 1 appears as the codified clause. Every literal in T_1 that were expanded by the clause derived from the negated question is termed a negated question-literal. In Fig. 5, such literals are shown inside the rectangles filled with color gray. Note that every negated question-literal in Fig. 5 is labeling a *leaf* node.

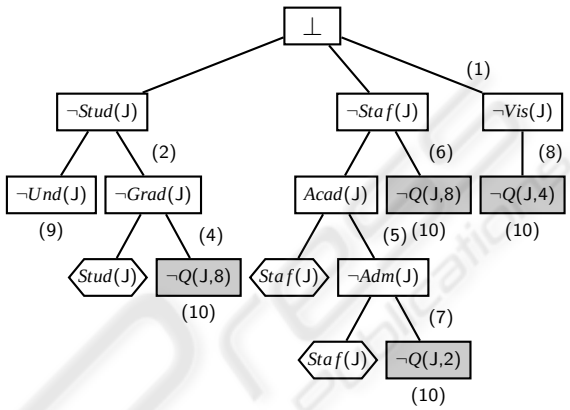

Figure 5: Proof-tree T_1 for Example 2.

***Step* 2.** Let $T_1 = (N, A, \lambda_1, r)$ be the proof-tree obtained in Step 1. From T_1, we now obtain the answer-tree $T_2 = (N, A, \lambda_2, r)$, where:

$$\lambda_2(v) = \overline{\lambda_1(v)}, \text{ for every } v \in N.$$

For our example, T_2 is shown in Fig. 6.

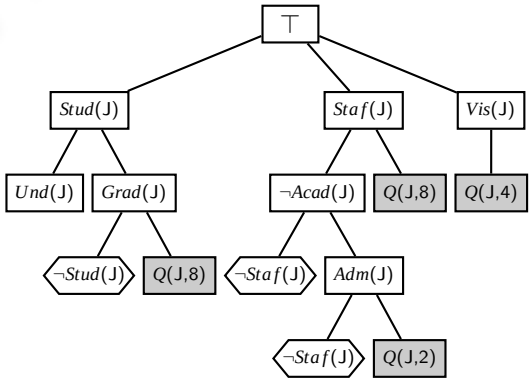

Figure 6: Answer-tree T_2.

The status of each literal (expanded or reduced) is preserved in an answer-tree. Also, if L is a negated question-literal labeling a node v in the proof-tree, the complementary literal, $\overline{L}$, that labels v in the answer-tree, is a question-literal.

The set of represented implications are reversed in such a way that

$$L_1 \wedge \ldots \wedge L_n \rightarrow M$$

becomes

$$\overline{M} \rightarrow \overline{L}_1 \vee \ldots \vee \overline{L}_n$$

in the answer-tree. Thus, if $v \in N$ is an internal node in T_2, and $u_1, \ldots, u_m \in N$ are its child nodes, then the following implication is an instance of a clause of the knowledge base:

$$\lambda_2(v) \rightarrow \lambda_2(u_1) \vee \ldots \vee \lambda_2(u_m).$$

***Step* 3.** Let $T_2 = (N, A, \lambda, r)$. For each non leaf node $v \in N$, let $P(v)$ be the set of all its ancestors[1] a such that there is a leaf b satisfying the conditions:

1. v is an ancestor of b;
2. $\lambda(b)$ is a reduced literal;
3. $\lambda(b) = \overline{\lambda(a)}$.

For a node v ancestor of a question-literal, if $a \in P(v)$, then the node a must be an ancestor of v in any posterior answer-tree generated from T_2, in order to obtain a correct final answer-tree. Fig. 7 shows the set $P(v)$ next to each v, in the cases where $P(v) \neq \emptyset$, for the answer-tree of Fig. 6 (note the use of names A and B for the first two sons of the root).

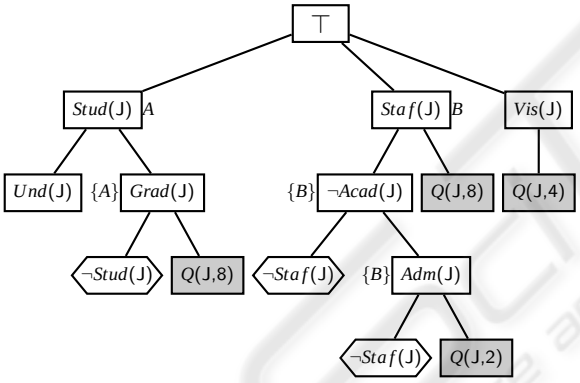

Figure 7: Answer-tree T_2 annotated with $P(v)$.

***Step* 4.** For answer-tree T_2, we now delete every node (and its associated arcs) which is not an ancestor of a question-literal. The resulting answer-tree will be T_3 as shown in Fig. 8.[2] As consequence of this step, we have that every leaf in the resulting answer-tree corresponds to a question-literal.

In answer-tree $T_3 = (N, A, \lambda, r)$, for an internal node $v \in N$, denote by $C(v)$ the conjunction of all literals $\lambda(a)$ such that $a \in P(v)$. Let $u_1, \ldots, u_m \in N$ be the child nodes of v. Then, we have that

$$C(v) \wedge \lambda(v) \rightarrow \lambda(u_1) \vee \ldots \vee \lambda(u_m)$$

follows from the knowledge base.

[1] In this paper, the ancestors of a node v are all the nodes along the path from the root to v *not* including v itself.

[2] Node names D, E and F will be used in the next step.

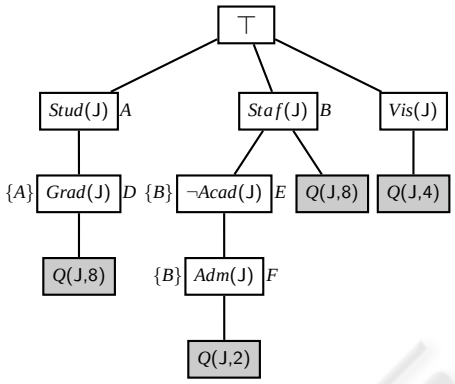

Figure 8: Answer-tree T_3.

***Step* 5.** In answer-tree $T_3 = (N, A, \lambda, r)$, let $v, u \in N$ be such that u is the only child of v, $v \neq r$ and u is not a leaf. Then one of them can be deleted from the answer-tree if it is not a member of $P(a)$ for any of its descendants a. If v is deleted, the father of v becomes the father of u, and if u is deleted, its sons become sons of v. Repeat the process for the resulting tree until you obtain an answer-tree T_4, where every node, except possibly the root, ramifies into two or more branches or has a question-literal as its only child.

In Fig. 8, node D is the only child of A. Since $A \in P(D)$, A can't be deleted. Fig. 9 shows the answer-tree $T_{3.1}$ that results after deleting node D from T_3.

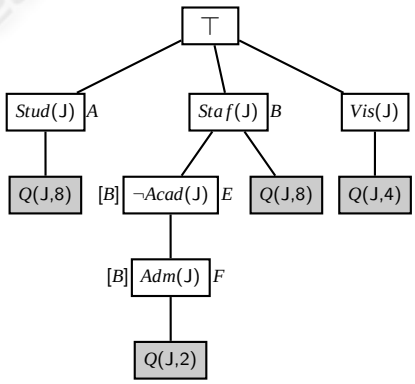

Figure 9: Answer-tree $T_{3.1}$.

In Fig. 9, we still have that node F is the only child of node E. Since E does not belong to the set $P(v)$, for any node v descendant of E, E is not the root and F is not a leaf, we can choose E to delete; similarly, we can choose F too. Deleting the node F from $T_{3.1}$, the resulting answer-tree, T_4, is as shown in Fig. 10. Now, as the only child of E is a leaf, E can't be deleted anymore.

For each answer-tree $T_{3.x}$ and for T_4, let $v \in N$ be an internal node and $u_1, \ldots, u_m \in N$ be its child nodes.

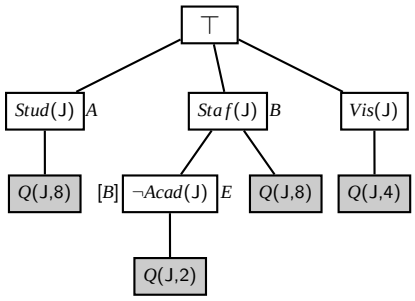

Figure 10: Answer-tree T_4.

Denote by $C(v)$ the conjunction of literals $\lambda(a)$ such that $a \in P(v)$. Then,

$$C(v) \wedge \lambda(v) \rightarrow \lambda(u_1) \vee \ldots \vee \lambda(u_m)$$

follows from the knowledge base.

***Step* 6.** In answer-tree $T_4 = (N, A, \lambda, r)$, let $q_1, q_2, \ldots, q_n \in N$ be the nodes labeled with question-literals. For each q_i, $i = 1, \ldots, n$, let $b_{i,1}, \ldots, b_{i,m} \in N$ be the brother nodes of q_i in T_4. If $m \geq 1$, i.e., q_i is not an only child in T_4, then create a new node $b_{i,m+1}$ brother of $b_{i,1}, \ldots, b_{i,m}$ and make q_i the child node of $b_{i,m+1}$. Assign the following label to $b_{i,m+1}$:

- $\lambda(b_{i,m+1}) = \overline{\lambda(b_{i,1})} \wedge \ldots \wedge \overline{\lambda(b_{i,m})}$.

This process must be repeated until all nodes labeled with question-literals are only childs. For our example, the answer-tree T_5 thus obtained is shown in Fig. 11.

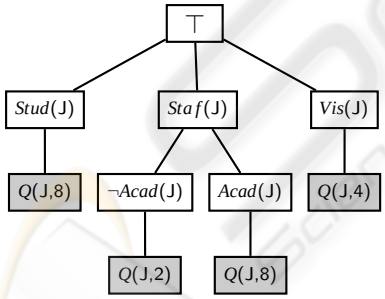

Figure 11: Answer-tree T_5.

In answer-tree $T_5 = (N, A, \lambda, r)$, if $v \in N$ is an internal node, $a_1, \ldots, a_n$ are its ancestor nodes and $u_1, \ldots, u_m \in N$ are its child nodes, then

$$\lambda(a_1) \wedge \ldots \wedge \lambda(a_n) \wedge \lambda(v) \rightarrow \lambda(u_1) \vee \ldots \vee \lambda(u_m)$$

follows from the knowledge base.

The tree in Fig. 11 is the final answer-tree obtained by the algorithm and, thus, corresponds to a case-based answer. Each path on the tree from the root to a leaf node corresponds to an implication $\alpha \rightarrow P(T_i)$, where α is formed by making a conjunction of all literals from the root to the leaf's father and $P(T_i)$ is the question-literal labeling the leaf.

In our example, to the question "Find y such that $Q(\mathrm{J}, y)$" ("How many books can John borrow?"), we extract the following case-based answer from the answer-tree illustrated in Fig. 11:

$* \; Stud(\mathrm{J}) \rightarrow Q(\mathrm{J},8)$
If John is a student, then John can borrow 8 books.

$* \; Staf(\mathrm{J}) \wedge \neg Acad(\mathrm{J}) \rightarrow Q(\mathrm{J},2)$
If John is a staff member and John is not an academic, then John can borrow 2 books.

$* \; Staf(\mathrm{J}) \wedge Acad(\mathrm{J}) \rightarrow Q(\mathrm{J},8)$
If John is is an academic staff member, then John can borrow 8 books.

$* \; Vis(\mathrm{J}) \rightarrow Q(\mathrm{J},4)$
If John is a visitor, then John can borrow 4 books.

5 CONCLUSIONS

A disjunctive answer of the form $P(T_1) \vee P(T_2) \vee \ldots \vee P(T_k)$, $k \geq 2$, although correct, may not be informative enough. The reason is that, when supplied with such answer, the user may not be able to determine which of the elements $P(T_i)$, $1 \leq i \leq k$, of the disjunction is an appropriate answer to his question. Thereby, the basic motivation of this work was to provide a more informative answer to the question posed by the user on the situations where the inference system would normally provide us with a disjunctive answer.

As defined in this work, a case-based answer to a general question $P(X)$ consists of an answer given in terms of a finite number of cases, each one implying a non disjunctive answer $P(T_i)$. The cases are extracted from a proof-tree, a structure that maintains enough information to admit such extraction.

In this current paper, we propose an algorithm that, from a deduction of $P(T_1) \vee P(T_2) \vee \ldots \vee P(T_k)$, $k \geq 2$, on the form a proof-tree, extracts a case-based answer to the question, of the form

$$\begin{array}{l} \alpha_1 \rightarrow P(T_{i_1}) \\ \alpha_2 \rightarrow P(T_{i_2}) \\ \ldots \\ \alpha_m \rightarrow P(T_{i_m}) \end{array}$$

$1 \leq i_1, i_2, \ldots, i_m \leq k$, where $\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_m$ is a logical consequence of the knowledge base and α_j is a case that implies the answer $P(T_{i_j})$.

It is interesting to note that there is a connection between our case-based answer and what is called abductive explanation (Brachman and Levesque, 2004).

Given a (consistent) knowledge base KB and a formula β to be explained, an abductive explanation of β is a formula α such that $KB \cup \{\alpha\} \models \beta$, or equivalently, $KB \models (\alpha \rightarrow \beta)$, and $KB \cup \{\alpha\}$ is consistent. Therefore, we can say that the cases $\alpha_1, \alpha_2, \ldots, \alpha_m$ are *special* abductive explanations to each disjunct, since $\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_m$ is a logical consequence of the knowledge base.

Another condition usually required for α to be qualified as an abductive explanation is that α is in the appropriate vocabulary, with the purpose that α makes sense *to the user* as an explanation. This condition can (and should) be used in a deterministic version of the algorithm here presented for the choice of the cases: depending on the choice of the cases, they may be useful or not. Thus, in practice, it is important to add to the system some knowledge about the vocabulary that allows appropriate choices.

As future work, we intend to generalize the method to apply it to non closed proof-trees, possibly using as reference the work of Burhans and Shapiro (Burhans and Shapiro, 2007), which was developed for inference procedures based on resolution.

REFERENCES

Brachman, R. and Levesque, H. (2004). *Knowledge Representation and Reasoning (The Morgan Kaufmann Series in Artificial Intelligence)*. Morgan Kaufmann.

Bruynooghe, M. (1982). The memory management of prolog implementations. In Clark, K. L. and Tärnlund, S.-A., editors, *Logic Programming*, pages 83–98. Academic Press, London.

Burhans, D. T. and Shapiro, S. C. (2007). Defining answer classes using resolution refutation. *J. Applied Logic*, 5(1):70–91.

Chang, C.-L. and Lee, R. C.-T. (1997). *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, Inc., Orlando, FL, USA.

Demolombe, R. (1992). A strategy for the computation of conditional answers. In *ECAI '92: Proceedings of the 10th European conference on Artificial intelligence*, pages 134–138, New York, NY, USA. John Wiley & Sons, Inc.

Green, C. C. (1969). *The application of theorem proving to question-answering systems*. PhD thesis, Stanford, CA, USA.

Letz, R. and Stenz, G. (2001). Model elimination and connection tableau procedures. In Robinson, J. A. and Voronkov, A., editors, *Handbook of Automated Reasoning*, pages 2015–2114. Elsevier and MIT Press.

Loveland, D. W. (1969). A simplified format for the model elimination theorem-proving procedure. *J. ACM*, 16(3):349–363.

Loveland, D. W. (1978). *Automated Theorem Proving: A logical Basis*. North–Holland.

Vieira, N. J. (1987). *Máquinas de Inferência para Sistemas Baseados em Conhecimento*. PhD thesis, PUC/RJ. In portuguese.