Generalizing Conjunctive Queries for Informative Answers

Katsumi Inoue and Lena Wiese*

National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
{ki|wiese}@nii.ac.jp

Abstract. Deductive generalization of queries is one method to provide informative answers to failing queries. We analyze properties of operators that generalize conjunctive queries consisting of positive as well as negative literals. We show that for the stepwise combination of these operators it suffices to apply the operator in one certain order.

1 Introduction and Related Work

Retrieval of data stored in database systems is a basic use case in the information society. Hence answering a user's queries is a central issue for database systems (apart from other aspects like for example integrity, availability and efficiency of the system). However, a database system may not always be able to answer queries in a satisfactory manner. In particular, if a database answer is empty the corresponding query is said to be a "failing query" (see for example [15]). The reasons for this can be manifold; for instance, in a selection query, selection conditions may be too strict to yield any result. Some authors (for example [5]) differentiate between "misconceptions" and "false presuppositions" as causes for failure. **Cooperative database systems** search for answers that – although not exactly matching the user's original query – are **informative answers** for the user: they provide data that are "closely related" to the user's intention; or they fix misconceptions in a query and return answers to the modified query. Current search engines, web shops or expert systems use similar cooperative techniques to provide users with data that might be close to their interest.

In this article we want to foster the logical foundation of cooperative query answering. We analyze a set of generalization operators that can be applied to failing queries. This logical point of view has some history of related work which we will survey briefly. The term "cooperative database system" was for example used in [2] for a system called "CoBase" that relies on several type abstraction hierarchies (TAH) to relax queries and hence to return a wider range of answers. In a similar manner, Halder and Cortesi [8] employ abstraction of domains and define optimality of answers with respect to some user-defined relevancy constraints. The approach by Pivert et al using fuzzy sets [15] analyzes cooperative

* Lena Wiese gratefully acknowledges a postdoctoral research grant of the German Academic Exchange Service (DAAD).

query answering based on semantic proximity. With what they call "incremental relaxation" they apply generalization operators to single conjuncts in a conjunctive query; hence generalized queries form a lattice structure: queries with the same number of applications generalization operators are in the same level of the lattice. Other related systems are Flex [13], Carmin [6] and Ishmael [5] that introduce and analyze dedicated generalization operators. Hurtado et al [9] relax RDF queries based on an ontology. In a multi-agent negotiation framework, Sakama and Inoue [17] devise procedures for generating "neighborhood proposals" in a negotiation process using the three operators we will also analyze in this paper.

We complement and advance these previous works by

- analyzing the properties of the three generalization operators "dropping conditions", "anti-instantiation", and "goal replacement" for deductive generalization of queries.
- combining these operators in a breadth-first search manner while employing a notion of minimality based on the number of applications of operators.

The paper is outlined as follows: Sections 2 and 3 set the basic terminology for cooperative query answering and query generalization. Section 4 introduces three basic generalization operators for conjunctive queries and Section 5 analyzes the combination of these. Section 6 concludes the paper with a brief discussion.

2 Cooperative Query Answering for Knowledge Bases

In this article we will follow a formal approach to achieve informative answers in a cooperative knowledge base system. In particular, we will base query answering on a consequence operator (denoted $\models$) in a chosen logic. Throughout this article we assume a logical language $\mathscr{L}$ consisting of a finite set of predicate symbols (for example denoted *Ill*, *Treat* or P), a possibly infinite set *dom* of constant symbols (for example denoted Mary or a), and an infinite set of variables (for example denoted x or y). The capital letter X denotes a vector of variables; if the order of variables in X does not matter, we identify X with the set of its variables and apply set operators – for example we write $y \in X$. We use the standard logical connectors conjunction $\wedge$, disjunction $\vee$, negation $\neg$ and material implication $\rightarrow$ and universal $\forall$ as well as existential $\exists$ quantifiers. An atom is a formula consisting of a single predicate symbol only; a literal is an atom (a "positive literal") or a negation of an atom (a "negative literal"); a clause is a disjunction of atoms; a ground formula is one that contains no variables; the existential (universal) closure of a formula ϕ is written as $\exists\phi$ ($\forall\phi$) and denotes the closed formula obtained by binding all free variables of ϕ with the respective quantifier. In particular, we will later on use single-headed "range-restricted rules" of the form $L_{i_1} \wedge \ldots \wedge L_{i_m} \rightarrow L'$ where each L_{i_j} and L' are literals and all free variables of L' are contained in the free variables of the L_{i_j}. On occasion, if a set is a singleton set we will identify the set with the single formula in it. For a set of formulas, a model is an interpretation that makes all formulas in the set *true*.

For two sets of formulas S and S', logical implication $S \models S'$ means that every model of S is also a model of S'. Two formulas ϕ, ϕ' are equivalent (denoted $\phi \equiv \phi'$) if $\phi \models \phi'$ and $\phi' \models \phi$. A particular case of equivalent formulas are "variants" that are syntactically identical up to variable renaming.

In this article, we assume that data are stored in a **knowledge base**. A knowledge base (denoted Σ) is a set of formulas in the chosen logic; if formulas in Σ contain free variables, these variables are assumed to be universally quantified; in other words, the formulas in Σ are identified with their universal closure. A knowledge base represents a set of "possible worlds"; that is, a set of models of the knowledge base. As an example, consider an information system with medical data on patient's illnesses and medical treatments. For the knowledge base $\Sigma = \{\mathit{Ill}(\mathsf{Mary}, \mathsf{Cough}) \vee \mathit{Ill}(\mathsf{Mary}, \mathsf{Flu})\}$, a possible world would be $\{\mathit{Ill}(\mathsf{Mary}, \mathsf{Cough})\}$ making only this single ground atom *true* and all other ground atoms *false*. Note that with an infinite underlying domain *dom* there are in general infinitely many worlds (and also infinitely many possible worlds) for a knowledge base. In addition, we implicitly assume that a knowledge base is consistent: it has at least one possible world. In an inconsistent database (that is, one containing a logical contradiction like $\Sigma = \{P(a) \wedge \neg P(a)\}$), any formula can be derived which makes query answering useless. Knowledge bases serve as a quite generic data model for the representation of knowledge in multiagent systems, belief revision or ontology-based reasoning. Note that when restricting a knowledge base to a set of ground atoms and accompanying it with a "closed world assumption" (that makes all ground atoms not contained in the knowledge base *false*), then the concept of a knowledge base can emulate the relational data model; our analysis will however apply to the general case also without this assumption.

A user can issue queries to a knowledge base to retrieve data from it. In the following we analyze operators that can generalize conjunctive queries with positive as well as negative literals.

Definition 1 (Query). *A* query *is a conjunctive formula $L_1 \wedge \ldots \wedge L_n$ where each L_i is a literal. We often abbreviate a query as $Q(X)$, where Q stands for the conjunction of literals and X is an n-tuple of variables appearing in Q.*

For query answering different semantics can be employed. For example, an open query $Q(X)$ can be answered by finding all substitutions for the free variables of $Q(X)$; more formally, for a substitution θ that maps the free variables of $Q(X)$ to arbitrary terms (including variables), $\forall Q(X)\theta$ is a correct answer if $\Sigma \models \forall Q(X)\theta$. [11] analyze minimal disjunctive answers whereas [17] use answer set semantics for extended disjunctive logic programs. In this article, we apply generalization operators to queries. This can be done independent of a specific query answering semantics. We just assume that a dedicated query answering function *ans* represents the set of all correct answers.

Definition 2 (Answer set). *For a query $Q(X)$ and a knowledge base Σ, the set of correct answers (or* answer set*, for short) $ans(Q(X), \Sigma)$ is a set of closed formulas such that for each $\phi \in ans(Q(X), \Sigma)$ it holds that $\Sigma \models \phi$ and ϕ is derived from $Q(X)$ by some query answering semantics.*

For a given knowledge base, not all queries can be answered correctly. If no correct answer for a query exists, the query is said to fail.

Definition 3 (Failing query). *Let Σ be a knowledge base, $Q(X)$ be a query. If $ans(Q(X), \Sigma) = \emptyset$, the query $Q(X)$* fails *(in Σ).*

3 Query Generalization

Our purpose is to devise strategies for **cooperative query answering with generalization**: if for some query $Q(X)$ the knowledge base answer is the empty set, $Q(X)$ is transformed into a more general query $Q^{gen}(X, Y)$ that has a non-empty answer in the knowledge base. This idea is visualized in Figure 1. Properties of inductive and deductive generalization have already been discussed

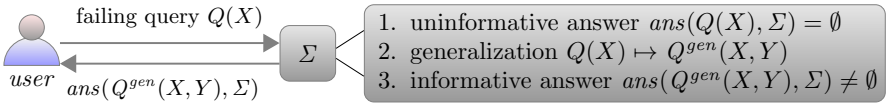

Fig. 1. Query generalization for informative answers

by de Raedt [3]. He argues that specialization and generalization are inverse operators and can be applied for both inductive and deductive reasoning. He also shows that an inductive operator applied to the negation of a formula behaves as a deductive operator (when again the result of the induction is negated). For query generalization we will apply deduction. Generalization can take place either in a way *independent* of knowledge base contents: it acts solely on the query formula $Q(X)$ like the generalization operator "dropping conditions". On the other hand, generalization can *depend* on the knowledge base by applying rules contained in the knowledge base: with the operator of "goal replacement" the query $Q(X)$ is modified by substituting a part of the query according to some rule in the knowledge base. In this sense, we will speak of generalization "with respect to a knowledge base". This notion has been used for "relaxation" in [4]. As it can be applied to open formulas we borrow it here; however, we combine operators in a different manner and employ an operator-based distance.

Definition 4 (Deductive generalization wrt. knowledge base [4]). *Let Σ be a knowledge base, $\phi(X)$ be a formula with a tuple X of free variables, and $\psi(X, Y)$ be a formula with an additional tuple Y of free variables disjoint from X. The formula $\psi(X, Y)$ is a* deductive generalization *of $\phi(X)$, if it holds in Σ that the less general ϕ implies the more general ψ where for the free variables X (the ones that occur in ϕ and possibly in ψ) the universal closure and for free variables Y (the ones that occur in ψ only) the existential closure is taken:*

$$\Sigma \models \forall X \exists Y \, (\phi(X) \rightarrow \psi(X, Y))$$

As a simple example, assuming classical first-order logic, for the formula $\phi_1 = \mathit{Ill}(\mathsf{Mary}, \mathsf{Cough})$ the formula $\mathit{Ill}(y, \mathsf{Cough})$ is a deductive generalization because it holds in any first-order model that $\exists y(\mathit{Ill}(\mathsf{Mary}, \mathsf{Cough}) \rightarrow \mathit{Ill}(y, \mathsf{Cough}))$. Analogously, $\mathit{Ill}(\mathsf{Mary}, y')$ and $\mathit{Ill}(y, y')$ are generalizations for the same formula ϕ_1. But note that $\mathit{Ill}(y, y')$ is also a generalization of $\mathit{Ill}(y, \mathsf{Cough})$ because it holds in any first-order model that $\forall y \exists y' (\mathit{Ill}(y, \mathsf{Cough}) \rightarrow \mathit{Ill}(y, y'))$. Whereas for $\mathit{Ill}(y, \mathsf{Cough})$ and $\mathit{Ill}(\mathsf{Mary}, y')$ neither is a generalization of the other.

We can now give a formal definition of generalized queries.

Definition 5 (Generalized query with informative answer). *A query formula $Q^{gen}(X, Y)$ is a* generalized query with informative answer *for a query $Q(X)$ (with respect to a knowledge base Σ) if the following properties hold:*

1. ***Failing query:*** $ans(Q(X), \Sigma) = \emptyset$
2. ***Generalized query:*** $Q^{gen}(X, Y)$ *is a deductive generalization of* $Q(X)$
3. ***Informative answer:*** $ans(Q^{gen}(X, Y), \Sigma) \neq \emptyset$

To capture a notion of closeness between the original query and the generalized query, the generalized query should have some property of **minimal** generalization. There are several definitions of minimality available. For example, Plotkin [16] devised subsumption-based "least general generalization". For generalization based on an abstraction hierarchy, minimal distance can be defined by a shortest path in the hierarchy [18]. Another established method is a model-based distance as for example used for dilation operators [7]. The number of applications of generalization operators is counted in [1]. We also employ a notion of minimality that counts the number of individual generalization steps based on a set *GenOp* of generalization operators. In contrast to [1], the operators we analyze (in particular "goal replacement") need not apply only to single conjuncts in a query but can also apply to several conjuncts at the same time; thus our operators lead to a generalization behavior different from [1].

Definition 6 (Generalization operator). *For sets S and S' of formulas, an operator o is a* generalization operator *if $o(S) = S'$ and for each $\psi \in S'$ there is a $\phi \in S$ such that ψ is a deductive generalization of ϕ.*

Note that o might not be applicable to all formulas in S and hence the mapping is only surjective. Such a generalization operator defines one single atomic generalization step – for example, as we will use later, dropping one condition, or replacing one goal, or anti-instantiating one constant in a query. We now assume that a set *GenOp* has been specified and define the operator-based distance; however we defer the description of generalization operators until Section 4.

Definition 7 (Operator-based distance). *Let Σ be a knowledge base, $\phi(X)$ be a formula, and GenOp be a set of generalization operators. Let $\mathcal{G}_i$ be the set of formulas obtained by applying i operators from GenOp to $\phi(X)$ in any possible way without allowing formulas that are equivalent to any formula in $\mathcal{G}_j$ $(j \leq i)$:*

$$\mathcal{G}_0 := \{\phi(X)\}$$
$$\mathcal{G}_i := \{\psi(X, Y) \mid \psi(X, Y) \in o(\mathcal{G}_{i-1}) \text{ where } o \in \mathit{GenOp} \text{ and for every } j \leq i \text{ there is no } \psi'(X, Y') \in \mathcal{G}_j \text{ such that } \psi'(X, Y') \equiv \psi(X, Y)\}$$

A formula $\psi(X,Y)$ has distance l *to $\phi(X)$ (with respect to Σ and GenOp) if $\psi(X,Y)$ can be obtained by applying at least l generalization operators to $\phi(X)$; in other words, if $\psi(X,Y) \in \mathcal{G}_l$.*

4 Cooperative Query Answering for Conjunctive Queries

In the following subsections, we analyze three generalization operators that modify a given query in order to obtain a generalized query with informative answer (if there is any at all); they are called Dropping condition (DC), Anti-instantiation (AI), and Goal replacement (GR).

4.1 Dropping condition

As we only consider conjunctions in a query, ignoring one of the conjuncts makes the query more general. For a given conjunctive query, we will call a "subquery" a conjunctive query that contains a subset of the conjuncts of the original query. Assume the original query consists of n conjuncts (we will say that the query has length n). The generalization operator dropping condition (see Operator 1) returns a subquery of length $n-1$. Note that in this case the free variables of Q^{gen} are a subset of the variables of Q and hence Y (from Definition 5) is empty because no new variables are introduced; this is why Y is left out of Q^{gen}.

Operator 1 Dropping condition (DC)

Input: Query $Q(X) = L_1 \wedge \ldots \wedge L_n$ of length n
Output: Generalized query $Q^{gen}(X)$ as a subquery of length $n-1$
1: Choose literal L_j from $L_1, \ldots, L_n$
2: **return** $L_1 \wedge \ldots \wedge L_{j-1} \wedge L_{j+1} \wedge \ldots \wedge L_n$

There are $\binom{n}{n-1}$ such subqueries of length $n-1$. For example, $P(x) \wedge Q(x)$ is a subquery of length 2 of $P(x) \wedge Q(x) \wedge R(x)$ generated by dropping the condition $R(x)$. The operator DC complies with Definition 4 because it holds tautologically (that is, in any knowledge base) that a conjunctive query implies all its subqueries. In particular, when $Q(X)$ is a conjunctive query of length n and $Q^{gen}(X)$ is a subquery of length $n-1$, then $\models \forall X\,(Q(X) \rightarrow Q^{gen}(X))$.

Proposition 1. *DC is a deductive generalization operator.*

4.2 Anti-instantiation

With anti-instantiation (see Operator 2) a new variable is introduced in the query: the free variables of Q^{gen} are equal to the free variables of Q plus one new variable y; hence $Y = \{y\}$. Thus, some conditions in the query are relaxed and the resulting query also covers answers with different values for y.

Note that here AI not only applies to constants but also to variables. In particular, AI covers these special cases:

Operator 2 Anti-instantiation (AI)

Input: Query $Q(X) = L_1 \wedge \ldots \wedge L_n$ of length n
Output: Generalized query $Q^{gen}(X, Y)$ with Y containing one new variable
1: From $Q(X)$ choose a term t such that t is
   – either a variable occurring in $Q(X)$ at least twice
   – or a constant
2: Choose one literal L_j where t occurs
3: Let L'_j be the literal with one occurrence of t replaced with a new variable
4: **return** $L_1 \wedge \ldots \wedge L_{j-1} \wedge L'_j \wedge L_{j+1} \wedge \ldots \wedge L_n$

– turning constants into variables: $P(a)$ is converted to $P(x)$ (see [12])
– breaking joins: $P(x) \wedge S(x)$ is converted to $P(x) \wedge S(y)$ (introduced in [4])
– naming apart variables inside atoms: $P(x, x)$ is converted to $P(x, y)$

For each constant a all occurences must be anti-instantiated (that is, there are $\sum_a |occ(a, Q(X))|$ possible anti-instantiations); the same applies to variables v (that is, there are $\sum_v(|occ(v, Q(X))|)$ possible anti-instantiations) – however, with the exception that if v only occurs twice ($|occ(v, Q(X))| = 2$), one occurence of v need not be anti-instantiated due to equivalence . AI is a deductive generalization operator according to Definition 4 because it holds tautologically that the literal containing term t implies the literal where t is replaced by a new variable y; thus, $\models \forall X \exists y\, (L_j \rightarrow L'_j)$, because the term t can be taken as a witness for the existence of a value for y. The same applies to the whole query: $\models \forall X \exists y\, (Q(X) \rightarrow Q^{gen}(X, Y))$.

Proposition 2. *AI is a deductive generalization operator.*

4.3 Goal replacement

Goal replacement (see Operator 3) checks if the body of a rule in the knowledge base can be mapped (via a substitution) to a subquery. If so, the head of the rule replaces the subquery (with the substitution applied). In this way goal replacement potentially introduces new constants and new predicate symbols in the generalized query and possibly some of the original query variables disappear. The resulting query may cover a greater range of values and hence lead to an informative answer for the user. With the GR operator, a single-headed range-restricted rule from the knowledge base Σ is involved and the property of being a deductive generalization operator now indeed depends on the knowledge base. Due to the range-restrictedness of rules, the GR operator does not introduce new variables. This is why in this case Y (from Definition 5) is the empty set and dropped from the notation.

For example, the query $P(a, y) \wedge P(b, y)$ can be converted to $Q(y) \wedge P(b, y)$ given the rule $P(x, y) \rightarrow Q(y)$ and the substitution $\theta = \{a/x, y/y\}$. The difficulty with GR is that Σ has to be checked for rules with a matching body. Showing that GR complies with Definition 4 amounts to showing that $\{\forall X\, (L_{i_1} \wedge \ldots \wedge L_{i_m}) \rightarrow$

Operator 3 Goal replacement (GR)

Input: Query $Q(X) = L_1 \wedge \ldots \wedge L_n$ of length n
Output: Generalized query $Q^{gen}(X)$ containing a replacement literal
1: From Σ choose a single-headed range-restricted rule $L_{i_1} \wedge \ldots \wedge L_{i_m} \rightarrow L'$ such that there is a substitution θ for which all literals $L_{i_1}\theta, \ldots, L_{i_m}\theta$ occur in $Q(X)$
2: Let $L_{i_{m+1}}, \ldots, L_{i_n}$ denote the literals of $Q(X)$ apart from $L_{i_1}\theta, \ldots, L_{i_m}\theta$
3: **return** $L_{i_{m+1}} \wedge \ldots \wedge L_{i_n} \wedge L'\theta$

$L'\} \models \forall X\,(Q(X) \rightarrow L'\theta)$; and this statement holds because $L_{i_1}\theta \wedge \ldots \wedge L_{i_m}\theta$ is a subquery of $Q(X)$. Hence, it holds that: $\Sigma \models \forall X\,(Q(X) \rightarrow Q^{gen}(X'))$.

Proposition 3. *GR is a deductive generalization operator.*

We remark that [4] introduce a more versatile goal replacement operator using so-called reciprocal clauses that may introduce several new conjuncts (and hence some further new predicate symbols) in the query; but for sake of simplicity we do not follow this approach further in this paper.

5 Combining Generalization Operators

We analyze the combination of the three operators DC, AI and GR. To the best of our knowledge a generalization method that combines these three operators has not been analyzed so far. In [17] generalization operators are combined with so-called conditional answers; but the generalization operators themselves are not combined with each other. Lattice properties of queries have already been analyzed by [5, 1] for operators that can be uniformly applied to any conjunct of a query (like for example dropping conditions). In contrast to this, goal replacement deletes some conjuncts and introduces a new conjunct while AI introduces a new variable; hence the generalization behavior highly depends on applicability of GR and AI. The analysis of a combination of DC, AI and GR is indeed worthwhile: the main issue is that behavior of operators can have a greater impact when used in combination with other operators. For instance, maximal succeeding subqueries (MSSs) are important when using dropping conditions alone; see the in-depth discussion in [5]. Yet, in combination with other operators, the identification of MSSs may not be the only option to get informative answers.

Moreover, for finding MSSs, [5] shows that a depth-first and top-down search procedure can efficiently find *some* of the minimal succeeding subqueries. However, a disadvantage of this depth-first search might be that the resulting query is far away from the user's original intention. Hence when searching for generalized queries that are close to the original one, an exhaustive search (either depth-first or breadth-first) up to a user-defined bound of generalization steps can be a more viable option and was also proposed in [1].

For the combination of DC, AI and GR, we iteratively apply the three generalization operators in all possible ways. In other words, we compute the sets $\mathcal{G}_i$ from Definition 7. In order to avoid unnecessary generation of duplicate queries,

we show in the following some properties of the sets of queries that are obtained by combining any two of the generalization operators DC, AI and GR. First we study the combination of AI and DC. As AI introduces a new variable, set-containment $\subseteq$ is meant up to variable renaming. When AI is followed by DC, the resulting queries can indeed be found by either dropping conditions alone or by commuting the operators.

Lemma 1 (DC following AI). *Given a query $Q(X)$, let the set of queries $S_1 = AI(DC(Q(X)))$ be obtained by first dropping conditions and then anti-instantiating, let $S'_1 = DC(Q(X))$ be the set of queries obtained by dropping conditions, and let the set of queries $S_2 = DC(AI(Q(X)))$ be obtained by first anti-instantiating and then dropping conditions. Then $S_2 \subseteq S_1 \cup S'_1$ (up to variable renaming).*

When anti-instantiating a term and then dropping the conjunct that contains the term, the two operations coincide with dropping the conjunct alone (and hence the query belongs to set S'_1). When however after anti-instantiating a term, a conjunct different from the one containing the term is dropped, the operators can be applied in reverse order (and hence the query belongs to set S_1). It follows that it suffices to apply AI after DC. In the reverse direction, it can also be shown that $S_1 \subseteq S_2$.

Lemma 2 (GR following DC). *Given a query $Q(X)$, let the set of queries $S_1 = GR(DC(Q(X)))$ be obtained by first dropping conditions and then replacing goals and let the set of queries $S_2 = DC(GR(Q(X)))$ be obtained by first replacing goals and then dropping conditions. Then $S_1 \subseteq S_2$.*

This lemma holds due to the fact that when a conjunct is dropped from a query and then a goal replacement with respect to some rule from the knowledge base is executed on the remaining query, the same can be achieved by first applying the rule and then dropping the conjunct. Hence we can avoid the application of the GR operator after the DC operator. The reverse however does not hold: in some cases DC results in a query to which GR cannot be applied anymore; for example, while $\{S(x), C(x)\}$ is the set of generalized queries obtained from the query $Q(x) = A(x) \wedge B(x) \wedge C(x)$ by first applying the rule $A(x) \wedge B(x) \rightarrow S(x)$ in a goal replacement step and then dropping conditions, the generalized query $C(x)$ can never be obtained by reversing the operators.

Lemma 3 (GR following AI). *Given a query $Q(X)$, let the set of queries $S_1 = AI(GR(Q(X)))$ be obtained by first replacing goals and then anti-instantiating, let the set of queries $S'_1 = GR(Q(X))$ be obtained by goal replacement alone, and let the set of queries $S_2 = GR(AI(Q(X)))$ be obtained by first anti-instantiating and then replacing goals. Then $S_2 \subseteq S_1 \cup S'_1$ (up to variable renaming).*

This is due to the fact that rules that can be applied to anti-instantiated queries, can also be applied to the original query first and then applying the anti-instantiation: Assume that some term t in the original query is anti-instantiated to the new variable y; then the substitution θ used in goal replacement maps

some variable of the rule to y. However the same can be achieved by first applying goal replacement by letting θ map the variable to t and then (if it is still contained in the replacement literal) anti-instantiating it to y. That is, we only have to apply AI to queries obtained by GR. The reverse direction does not hold because after AI some rule used for GR may not be applicable anymore.

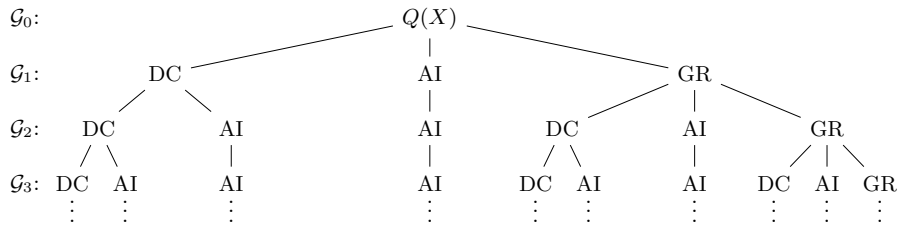

Fig. 2. Operator tree

Based on the above lemmata, we can now conclude that in order to compute the sets of generalized queries $\mathcal{G}_i$ the three operators under scrutiny need only be applied in a certain order. This is illustrated in Figure 2.

Theorem 1 (Operator ordering). *When combining the generalization operators DC, AI and GR, the following computations can be avoided: GR following DC, DC following AI, and GR following AI.*

With this result, the search for generalized queries can be much more efficient. However the actual efficiency gain depends on the applicability of GR and AI: when GR is applicable, it suffices to apply it in the first generalization steps before any other operators, followed by DC steps and lastly AI steps. But even if GR is not applicable we never even have to check for its applicability after DC or AI (this check includes finding matching rules and is hence quite expensive). As AI generates a lot of generalized queries, with our result we can apply it only in the last generalization steps without missing out any generalized query.

Example 1. We now give a comprehensive example. Assume knowledge base $\Sigma = \{\mathit{Ill}(\mathsf{Pete}, \mathsf{Flu}), \mathit{Ill}(\mathsf{Mary}, \mathsf{Cough}), \mathit{Treat}(\mathsf{Mary}, \mathsf{Medi}), \mathit{Ill}(x, \mathsf{Flu}) \rightarrow \mathit{Treat}(x, \mathsf{Medi})\}$. The user asks the query $Q(X) = \mathit{Ill}(x, \mathsf{Flu}) \wedge \mathit{Ill}(x, \mathsf{Cough})$ which fails in Σ (under both exact and disjunctive answer semantics). The generalization sets up to $\mathcal{G}_4$ are shown in Table 1. These four sets comprise all possible generalizations of $Q(X)$. Already $\mathcal{G}_1$ can give informative answers (like for example $\mathit{Ill}(\mathsf{Pete}, \mathsf{Flu})$ or $\mathit{Treat}(\mathsf{Mary}, \mathsf{Medi}) \wedge \mathit{Ill}(\mathsf{Mary}, \mathsf{Cough})$ and hence might satisfy the user's needs.

6 Discussion and Conclusion

Although a given query fails, a knowledge base might still be able to return informative answers to a slightly modified query. We provided a profound anal-

$DC(Q(X))$	$\{Ill(x, \mathsf{Cough})\ ,\ Ill(x, \mathsf{Flu})\}$
$\cup\, AI(Q(X))$	$\cup \{Ill(y, \mathsf{Flu}) \wedge Ill(x, \mathsf{Cough})\ ,\ Ill(x, y) \wedge Ill(x, \mathsf{Cough}),$ $Ill(x, \mathsf{Flu}) \wedge Ill(x, y)\}$
$\cup\, GR(Q(X))$	$\cup \{Treat(x, \mathsf{Medi}) \wedge Ill(x, \mathsf{Cough})\}$
$DC(DC(Q(X)))$	$\{\varepsilon\}$
$\cup\, AI(DC(Q(X)))$	$\cup \{Ill(x, y)\}$
$\cup\, AI(AI(Q(X)))$	$\cup \{Ill(y, y') \wedge Ill(x, \mathsf{Cough}),\ Ill(y, \mathsf{Flu}) \wedge Ill(x, y'),$ $Ill(x, y) \wedge Ill(x, y')\ \}$
$\cup\, DC(GR(Q(X)))$	$\cup \{Treat(x, \mathsf{Medi})\}$
$\cup\, AI(GR(Q(X)))$	$\cup \{Treat(y, \mathsf{Medi}) \wedge Ill(x, \mathsf{Cough}),\ Treat(x, y) \wedge Ill(x, \mathsf{Cough}),$ $Treat(x, \mathsf{Medi}) \wedge Ill(x, y)\}$
$GR(GR(Q(X))) = \emptyset$	
$AI(AI(AI(Q(X))))$	$\{Ill(y, y') \wedge Ill(x, y'')\}$
$\cup\, AI(DC(GR(Q(X))))$	$\cup \{Treat(x, y)\}$
$\cup\, AI(AI(GR(Q(X))))$	$\cup \{Treat(y, y') \wedge Ill(x, \mathsf{Cough}),$ $Treat(y, \mathsf{Medi}) \wedge Ill(x, y'),\ Treat(x, y) \wedge Ill(x, y')\}$
$DC(DC(DC(Q(X)))) = AI(DC(DC(Q(X)))) = AI(AI(DC(Q(X)))) = DC(DC(GR(Q(X)))) = \emptyset$	
$AI(AI(AI(GR(Q(X)))))$	$\{Treat(y, y') \wedge Ill(x, y'')\}$
$AI(AI(AI(AI(Q(X))))) = AI(AI(DC(GR(Q(X))))) = \emptyset$	

Table 1. Generalization sets $\mathcal{G}_1$ to $\mathcal{G}_4$ for query $Q(X) = Ill(x, \mathsf{Flu}) \wedge Ill(x, \mathsf{Cough})$

ysis of the combination of the three operators DC, AI and GR that stepwise generalize conjunctive queries with positive and negative literals. A prototype implementation using a current theorem proving system (SOLAR [14]) is under development. One open issue is to avoid *overgeneralization* that leads to queries far from the user's original intent; e.g., the AI operator can be prevented from generating unrestricted predicates (like $Ill(x, y)$) by reinstantiating and hence computing neighborhood queries in the sense of [17]. Moreover, generalization operators could for example be restricted by defining user preferences on the literals in a query. An extension of our approach might consider other logical settings (e.g., queries of a more general form, nonmonotonic reasoning or "conditional answers" [10, 17]). Another topic of future work may be the development of an algorithm that directly and incrementally returns informative answers in our setting (as done by [9] for RDF queries and ontologies) without computing the sets of generalized queries.

References

1. Patrick Bosc, Allel HadjAli, and Olivier Pivert. Incremental controlled relaxation of failing flexible queries. *JIIS*, 33(3):261–283, 2009.
2. Wesley W. Chu, Hua Yang, Kuorong Chiang, Michael Minock, Gladys Chow, and Chris Larson. CoBase: A scalable and extensible cooperative information system. *JIIS*, 6(2/3):223–259, 1996.
3. Luc de Raedt. Induction in logic. In *MSL-96*, pages 29–38. AAAI Press, 1996.

4. Terry Gaasterland, Parke Godfrey, and Jack Minker. Relaxation as a platform for cooperative answering. *JIIS*, 1(3/4):293–321, 1992.
5. Parke Godfrey. Minimization in cooperative response to failing database queries. *IJCS*, 6(2):95–149, 1997.
6. Parke Godfrey, Jack Minker, and Lev Novik. An architecture for a cooperative database system. In *ADB-94*, volume 819 of *LNCS*, pages 3–24. Springer, 1994.
7. Nikos Gorogiannis and Anthony Hunter. Merging first-order knowledge using dilation operators. In *FoIKS2008*, volume 4932 of *LNCS*, pages 132–150. Springer, 2008.
8. Raju Halder and Agostino Cortesi. Cooperative query answering by abstract interpretation. In *SOFSEM2011*, volume 6543 of *LNCS*, pages 284–296. Springer, 2011.
9. Carlos A. Hurtado, Alexandra Poulovassilis, and Peter T. Wood. Query relaxation in RDF. In *J. Data Semantics X*, volume 4900 of *LNCS*, pages 31–61. Springer, 2008.
10. Katsumi Inoue, Koji Iwanuma, and Hidetomo Nabeshima. Consequence finding and computing answers with defaults. *J. Intell. Inf. Syst.*, 26(1):41–58, 2006.
11. Koji Iwanuma and Katsumi Inoue. Minimal answer computation and SOL. In *JELIA2002*, volume 2424 of *LNCS*, pages 245–258. Springer, 2002.
12. Ryszard S. Michalski. A theory and methodology of inductive learning. *Artificial Intelligence*, 20(2):111–161, 1983.
13. Amihai Motro. Flex: A tolerant and cooperative user interface to databases. *IEEE Transactions on Knowledge & Data Engineering*, 2(2):231–246, 1990.
14. Hidetomo Nabeshima, Koji Iwanuma, Katsumi Inoue, and Oliver Ray. SOLAR: An automated deduction system for consequence finding. *AI Communications*, 23(2-3):183–203, 2010.
15. Olivier Pivert, Hélène Jaudoin, Carmen Brando, and Allel HadjAli. A method based on query caching and predicate substitution for the treatment of failing database queries. In *ICCBR 2010*, volume 6176 of *LNCS*, pages 436–450. Springer, 2010.
16. Gordon Plotkin. *Automatic methods of inductive inference*. PhD thesis, University of Edinburgh, 1971.
17. Chiaki Sakama and Katsumi Inoue. Negotiation by abduction and relaxation. In *AAMAS2007*, pages 1010–1025. IFAAMAS, 2007.
18. Myung Keun Shin, Soon-Young Huh, and Wookey Lee. Providing ranked cooperative query answers using the metricized knowledge abstraction hierarchy. *Expert Systems with Applications*, 32(2):469–484, 2007.