A Generalized QSQR Evaluation Method for Horn Knowledge Bases

EWA MADALIŃSKA-BUGAJ, University of Warsaw
LINH ANH NGUYEN, University of Warsaw

We generalize the QSQR evaluation method to give the first set-oriented depth-first evaluation method for Horn knowledge bases. The resulting procedure closely simulates SLD-resolution (to take advantages of the goal-directed approach) and highly exploits set-at-a-time tabling. Our generalized QSQR evaluation procedure is sound and complete. It does not use adornments and annotations. To deal with function symbols, our procedure uses iterative deepening search which iteratively increases term-depth bound for atoms and substitutions occurring in the computation. When the term-depth bound is fixed, our evaluation procedure runs in polynomial time in the size of extensional relations.

Categories and Subject Descriptors: H.2.4 [**Database Management**]: Systems—*Query processing*; *Rule-based databases*

General Terms: Algorithms

Additional Key Words and Phrases: QSQR evaluation method, Horn knowledge bases

ACM Reference Format:
Madalińska-Bugaj, E. and Nguyen, L.A. 2011. A Generalized QSQR Evaluation Method for Horn Knowledge Bases. ACM Trans. Comput. Logic ?, ?, Article ? (January 2012), 27 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. INTRODUCTION

The Horn fragment of first-order logic plays an important role in knowledge representation and reasoning. It is used as the language of definite logic programs and goals in logic programming. Its range-restricted function-free version is also used as the Datalog language for deductive databases.[1] The Horn fragment itself is not very expressive because of its monotonicity w.r.t. the set of positive consequences, but it has received considerable attention due to the following reasons. Firstly, there are efficient computational methods like tabled SLD-resolution for the Horn fragment. Secondly, for some restricted Horn fragments, the complexity of the problem of checking logical consequences may be reduced to polynomial time. For example, the data complexity of Datalog is in PTIME. Thirdly, the Horn fragment is a starting point for developing more expressive languages with appropriate semantics and computational procedures (e.g. normal logic programs and goals with SLDNF-resolution calculus, Datalog$^\neg$ with well-

[1] A definite program clause is range-restricted if every variable occurring in a rule's head also occurs in the rule's body.

This work was supported by the National Centre for Research and Development (NCBiR) under Grant No. SP/I/1/77065/10 by the strategic scientific research and experimental development program: "Interdisciplinary System for Interactive Scientific and Scientific-Technical Information".
Authors' address: E. Madalińska-Bugaj and L.A. Nguyen, Institute of Informatics, University of Warsaw.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

founded semantics, and stratified Datalog$^{\pm}$ with the canonical model semantics [Calì et al. 2009]).

Horn knowledge bases are definite logic programs, which are usually so big that either they cannot be totally loaded into memory or evaluations for them cannot be done totally in memory. Thus, in contrast to logic programming, for Horn knowledge bases efficient access to secondary storage is an important aspect. Horn knowledge bases can be treated as extensions of Datalog deductive databases without the range-restrictedness and function-free conditions. Developing efficient evaluation methods for Horn knowledge bases is worth not only for practical applications but also for the theory of knowledge bases. This problem, especially in our setting, did not receive much attention from researchers during the last decade, but this does not mean that the problem has been well-studied and does not need further investigations.

To develop evaluation procedures for Horn knowledge bases one can either adapt tabled SLD-resolution systems of logic programming to reduce the number of accesses to secondary storage or generalize evaluation methods of Datalog queries to deal with non-range-restricted definite logic programs and goals that may contain function symbols.

Tabled SLD-resolution systems like OLDT [Tamaki and Sato 1986], SLD-AL [Vieille 1987; 1989], linear tabulated resolution [Shen et al. 2001; Zhou and Sato 2003] are efficient computational procedures for logic programming without redundant recomputations, but they are not directly applicable to Horn knowledge bases to obtain efficient evaluation engines because they are not set-oriented (set-at-a-time). In particular, the suspension-resumption mechanism and the stack-wise representation as well as the "global optimizations of SLD-AL" are all tuple-oriented (tuple-at-a-time). Data structures for them are too complex so that they must be dropped if one wants to convert the methods to efficient set-oriented ones. Of course, one can use, e.g., XSB [Sagonas and Swift 1998; Sagonas et al. 1994] (a state-of-the-art implementation of OLDT) as a Horn knowledge base engine, but as pointed out in [Freire et al. 1997], it is tuple-oriented and not suitable for efficient access to secondary storage. The try of converting XSB to a set-oriented engine [Freire et al. 1997] removes essential features of XSB and is not natural.[2]

In [Vieille 1989], Vieille adapted SLD-AL resolution for Datalog deductive databases to obtain the top-down QoSaQ evaluation method by representing (sets of) goals by means of (sets of) tuples and translating the operations of SLD-AL on goals into operations on tuples. This evaluation method can be implemented as a set-oriented procedure, but Vieille stated that *"We would like, however, to go even further and to claim that the practical interest of our approach lies in its one-inference-at-a-time basis, as opposed to having a set-theoretic basis. First, this tuple-based computational model permits a fine analysis of the duplicate elimination issue. . . ."* [Vieille 1989, page 5]. Moreover, the specific techniques of QoSaQ like "instantiation pattern", "rule compilation", "projection" are heavily based on the range-restrictedness and function-free conditions.

In [Vieille 1986], Vieille gave the query-subquery recursive (QSQR) evaluation method for Datalog deductive databases, which is a top-down method based on SLD-resolution and the set-at-a-time technique. The first version of Vieille's QSQR algorithm presented in [Vieille 1986] is incomplete [Vieille 1989; Nejdl 1987]. As pointed out by Mohamed Yahya in a personal communication, the presentation of QSQR method in the book [Abiteboul et al. 1995] and our generalization for Horn knowledge bases given in the conference paper [Madalińska-Bugaj and Nguyen 2008] are also incomplete. This is corrected in this paper by using an outer loop which clears

[2]The original XSB uses depth-first search, while Breadth-First XSB [Freire et al. 1997] does not.

global $input\_$ relations for each iteration.[3] The QSQR method [Vieille 1986; Abiteboul et al. 1995] uses adornments to simulate SLD-resolution in pushing constant symbols from goals to subgoals. The annotated version of QSQR also uses annotations to simulate SLD-resolution in pushing repeats of variables from goals to subgoals (see, e.g., [Abiteboul et al. 1995]).

The magic-set technique [Bancilhon et al. 1986; Rohmer et al. 1986] is another formulation of tabling for Datalog deductive databases. It simulates the top-down QSQR evaluation by rewriting a given query to another equivalent one that when evaluated using a bottom-up technique (e.g. the seminaive evaluation) produces only facts produced by the QSQR evaluation. Adornments are used as in the QSQR evaluation. To simulate annotations, the magic-set transformation is augmented with subgoal rectification (see, e.g., [Abiteboul et al. 1995]). For the connection between top-down and bottom-up approaches to Datalog deductive databases we refer the reader to Bry's elegant unifying framework [Bry 1990]. Some authors have extended the magic-set technique for Horn knowledge bases [Ramakrishnan et al. 1992; Freire et al. 1997]. To deal with non-range-restrictedness and function symbols, "magic predicates" are used without adornments.

As seen from the above discussion, there are tuple-oriented depth-first evaluation methods (e.g. [Sagonas et al. 1994]) and (set-oriented) breadth-first evaluation methods [Ramakrishnan et al. 1992; Freire et al. 1997] (based on the magic-set transformation and the bottom-up seminaive evaluation) for Horn knowledge bases. However, as far as we know, no set-oriented depth-first evaluation method was developed for Horn knowledge bases.

In this paper, we generalize the QSQR evaluation method to give a set-oriented depth-first evaluation method for Horn knowledge bases. The resulting procedure closely simulates SLD-resolution (to take advantages of the goal-directed approach) and highly exploits set-at-a-time tabling. Our generalized QSQR evaluation procedure is sound and complete. It does not use adornments and annotations (but has the effects of the annotated QSQR method). To deal with function symbols, our procedure uses iterative deepening search which iteratively increases term-depth bound for atoms and substitutions occurring in the computation. When the term-depth bound is fixed, our evaluation procedure runs in polynomial time in the size of extensional relations.

The rest of this paper is structured as follows. In Section 2 we recall some notions of first-order logic, logic programming, and Horn knowledge bases. In Section 3 we present our generalized QSQR evaluation method for Horn knowledge bases. We start the section with an informal description and an illustrative example. Next, we present a formal tuple-at-a-time version of the method and then prove its soundness and completeness. In Section 4 we convert the tuple-at-a-time version to a set-at-a-time one, estimate the data complexity of the new version, and discuss optimizations and the problem of relaxing term-depth bound. In Section 5 we provide some preliminary experimental results. Section 6 concludes this work.

2. PRELIMINARIES

First-order logic is considered in this work and we assume that the reader is familiar with it. We recall only the most important definitions for our work and refer the reader to [Lloyd 1987; Apt 1997] for further reading.

A signature for first-order logic consists of constant symbols, function symbols, and predicate symbols. Terms and formulae over a fixed signature are defined using the symbols of the signature and variables in the usual way. An *atom* is a formula of the

[3] See Remark 3.2 (on page 9) for a discussion on this problem.

form $p(t_1, \ldots, t_n)$, where p is an n-ary predicate and $t_1, \ldots, t_n$ are terms. An *expression* is either a term, a tuple of terms or a formula without quantifiers, and a *simple expression* is either a term or an atom. The *term-depth of an expression* is the maximal nesting depth of function symbols occurring in that expression.

2.1. Substitution and Unification

A *substitution* is a finite set $\theta = \{x_1/t_1, \ldots, x_k/t_k\}$, where $x_1, \ldots, x_k$ are pairwise distinct variables, $t_1, \ldots, t_k$ are terms, and $t_i \neq x_i$ for all $1 \leq i \leq k$. The set $dom(\theta) = \{x_1, \ldots, x_k\}$ is called the *domain* of θ, while the set $range(\theta) = \{t_1, \ldots, t_k\}$ is called the *range* of θ. By ε we denote the *empty substitution*. The restriction of a substitution θ to a set X of variables is the substitution $\theta_{|X} = \{(x/t) \in \theta \mid x \in X\}$. The *term-depth of a substitution* is the maximal nesting depth of function symbols occurring in that substitution.

Let $\theta = \{x_1/t_1, \ldots, x_k/t_k\}$ be a substitution and E be an expression. Then $E\theta$, the *instance* of E by θ, is the expression obtained from E by simultaneously replacing all occurrences of the variable x_i in E by the term t_i, for $1 \leq i \leq k$.

Let $\theta = \{x_1/t_1, \ldots, x_k/t_k\}$ and $\delta = \{y_1/s_1, \ldots, y_h/s_h\}$ be substitutions (where $x_1, \ldots, x_k$ are pairwise distinct variables, and $y_1, \ldots, y_h$ are also pairwise distinct variables). Then the *composition* $\theta\delta$ of θ and δ is the substitution obtained from the sequence $\{x_1/(t_1\delta), \ldots, x_k/(t_k\delta), y_1/s_1, \ldots, y_h/s_h\}$ by deleting any binding $x_i/(t_i\delta)$ for which $x_i = (t_i\delta)$ and deleting any binding y_j/s_j for which $y_j \in \{x_1, \ldots, x_k\}$.

If θ and δ are substitutions such that $\theta\delta = \delta\theta = \varepsilon$, then we call them *renaming substitutions*. We say that an expression E is a *variant* of an expression E' if there exist substitutions θ and γ such that $E = E'\theta$ and $E' = E\gamma$.

A substitution θ is *more general* than a substitution δ if there exists a substitution γ such that $\delta = \theta\gamma$. Note that according to this definition, θ is more general than itself.

Let Γ be a set of simple expressions. A substitution θ is called a *unifier* for Γ if $\Gamma\theta$ is a singleton. If $\Gamma\theta = \{\varphi\}$ then we say that θ unifies Γ (into φ). A unifier θ for Γ is called a *most general unifier* (mgu) for Γ if θ is more general than every unifier of Γ.

There is an effective algorithm, called the *unification algorithm*, for checking whether a set Γ of simple expressions is unifiable (i.e. has a unifier) and computing an mgu for Γ if Γ is unifiable (see, e.g., [Lloyd 1987]).

If E is an expression or a substitution then by $Var(E)$ we denote the set of variables occurring in E. If φ is a formula then by $\forall(\varphi)$ we denote the *universal closure* of φ, which is the formula obtained by adding a universal quantifier for every variable having a free occurrence in φ.

2.2. Positive Logic Programs and SLD-Resolution

A (positive or definite) *program clause* is a formula of the form $\forall(A \vee \neg B_1 \vee \ldots \vee \neg B_k)$ with $k \geq 0$, written as $A \leftarrow B_1, \ldots, B_k$, where A, B_1, …, B_k are atoms. A is called the *head*, and $(B_1, \ldots, B_k)$ the *body* of the program clause. If p is the predicate of A then the program clause is called a program clause defining p.

A *positive* (or *definite*) *logic program* is a finite set of program clauses.

A *goal* (also called a *negative clause*) is a formula of the form $\forall(\neg B_1 \vee \ldots \vee \neg B_k)$, written as $\leftarrow B_1, \ldots, B_k$, where $B_1, \ldots, B_k$ are atoms. If $k = 1$ then the goal is called a *unary goal*. If $k = 0$ then the goal stands for falsity and is called the *empty goal* (or the *empty clause*) and denoted by $\Box$.

If P is a positive logic program and $G = \leftarrow B_1, \ldots, B_k$ is a goal, then θ is called a *correct answer* for $P \cup \{G\}$ if $P \models \forall((B_1 \wedge \ldots \wedge B_k)\theta)$.

We now give definitions for SLD-resolution.

A goal G' is *derived* from a goal $G = \leftarrow A_1, \ldots, A_i, \ldots, A_k$ and a program clause $\varphi = (A \leftarrow B_1, \ldots, B_h)$ using A_i as the *selected atom* and θ as the most general unifier

(mgu) if θ is an mgu for A_i and A, and $G' = \leftarrow (A_1, \ldots, A_{i-1}, B_1, \ldots, B_h, A_{i+1}, \ldots, A_k)\theta$. We call G' a *resolvent* of G and φ. If $i = 1$ then we say that G' is derived from G and φ using *the leftmost selection function*.

Let P be a positive logic program and G be a goal.

An *SLD-derivation* from $P \cup \{G\}$ consists of a (finite or infinite) sequence $G_0 = G$, G_1, $G_2, \ldots$ of goals, a sequence $\varphi_1, \varphi_2, \ldots$ of variants of program clauses of P and a sequence $\theta_1, \theta_2, \ldots$ of mgu's such that each G_{i+1} is derived from G_i and φ_{i+1} using θ_{i+1}. Each φ_i is a suitable variant of the corresponding program clause. That is, φ_i does not have any variables which already appear in the derivation up to G_{i-1}. Each program clause variant φ_i is called an *input program clause*.

An *SLD-refutation* of $P \cup \{G\}$ is a finite SLD-derivation from $P \cup \{G\}$ which has the empty clause as the last goal in the derivation.

A *computed answer* θ for $P \cup \{G\}$ is the substitution obtained by restricting the composition $\theta_1 \ldots \theta_n$ to the variables of G, where $\theta_1, \ldots, \theta_n$ is the sequence of mgu's occurring in an SLD-refutation of $P \cup \{G\}$.

THEOREM 2.1 (SOUNDNESS AND COMPLETENESS OF SLD-RESOLUTION). *Let P be a positive logic program and G be a goal. Then every computed answer for $P \cup \{G\}$ is a correct answer for $P \cup \{G\}$. Conversely, for every correct answer θ for $P \cup \{G\}$, using any selection function there exists a computed answer δ for $P \cup \{G\}$ such that $G\theta = G\delta\gamma$ for some substitution γ.* [Clark 1979; Stärk 1990]

We will use also the following well-known lemmas:

LEMMA 2.2 (LIFTING LEMMA). *Let P be a positive logic program, G be a goal, θ be a substitution, and l be a natural number. Suppose there exists an SLD-refutation of $P \cup \{G\theta\}$ using mgu's $\theta_1, \ldots, \theta_n$ such that the variables of the input program clauses are distinct from the variables in G and θ and the term-depths of the goals and the composition $\theta_1 \ldots \theta_n$ are bounded by l. Then there exist a substitution γ and an SLD-refutation of $P \cup \{G\}$ using the same sequence of input program clauses, the same selected atoms and mgu's $\theta'_1, \ldots, \theta'_n$ such that the term-depths of the goals and the composition $\theta'_1 \ldots \theta'_n$ are bounded by l and $\theta\theta_1 \ldots \theta_n = \theta'_1 \ldots \theta'_n\gamma$.*

The Lifting Lemma given in [Lloyd 1987] does not contain the condition "the variables of the input program clauses are distinct from the variables in G and θ" and is therefore inaccurate (see, e.g., [Apt 1997]). The correct version given above follows from the one presented, amongst others, in [Staab 2008]. For applications of this lemma in this paper, we assume that *fresh variables* from a special infinite list of variables are used for renaming variables of input program clauses in SLD-derivations, and that mgu's are computed using a standard method. The mentioned condition will thus be satisfied.

In a computational process, a *fresh variant* of a formula φ, where φ can be an atom, a goal $\leftarrow A$ or a program clause $A \leftarrow B_1, \ldots, B_k$ (written without quantifiers), is a formula $\varphi\theta$, where θ is a renaming substitution such that $dom(\theta) = Var(\varphi)$ and $range(\theta)$ consists of fresh variables that were not used in the computation (and the input).

LEMMA 2.3 (LEMMA 8.5 IN [LLOYD 1987]). *Let P be a positive logic program and A be an atom. Suppose that $P \models \forall(A)$. Then there exists an SLD-refutation of $P \cup \{\leftarrow A\}$ with the empty substitution as the computed answer.*

2.3. Definitions for Horn Knowledge Bases

Similarly as for deductive databases, we classify each predicate either as *intensional* or as *extensional*. A *generalized tuple* is a tuple of terms, which may contain function symbols and variables. A *generalized relation* is a set of generalized tuples of the same

arity. A *Horn knowledge base* is defined to be a pair consisting of a positive logic program for defining intensional predicates and a *generalized extensional instance*, which is a function mapping each extensional n-ary predicate to an n-ary generalized relation. Note that intensional predicates are defined by a positive logic program which may contain function symbols and not be range-restricted. From now on, we use the term "relation" to mean a generalized relation, and the term "extensional instance" to mean a generalized extensional instance.

Note: We will treat a tuple $\bar{t}$ from a relation associated with a predicate p as the atom $p(\bar{t})$. Thus, a relation (of tuples) of a predicate p is a set of atoms of p, and an extensional instance is a set of atoms of extensional predicates. Conversely, a set of atoms of p can be treated as a relation (of tuples) of the predicate p.

Given a Horn knowledge base specified by a positive logic program P and an extensional instance I, a *query* to the knowledge base is a positive formula $\varphi(\overline{x})$ without quantifiers, where $\overline{x}$ is a tuple of all the variables of φ.[4] A (*correct*) *answer* for the query is a tuple $\bar{t}$ of terms of the same length as $\overline{x}$ such that $P \cup I \models \forall(\varphi(\bar{t}))$. When measuring *data complexity*, we assume that P and φ are fixed, while I varies. Thus, the pair $(P, \varphi(\overline{x}))$ is treated as a *query* to the extensional instance I. We will use the term "query" in this meaning.

It can easily be shown that, every query $(P, \varphi(\overline{x}))$ can be transformed in polynomial time to an equivalent query of the form $(P', q(\overline{x}))$ over a signature extended with new intensional predicates, including q. The equivalence means that, for every extensional instance I and every tuple $\bar{t}$ of terms of the same length as $\overline{x}$, $P \cup I \models \forall(\varphi(\bar{t}))$ iff $P' \cup I \models \forall(q(\bar{t}))$. The transformation is based on introducing new predicates for defining complex subformulae occurring in the query. For example, if $\varphi = p(x) \wedge r(x, y)$, then $P' = P \cup \{q(x, y) \leftarrow p(x), r(x, y)\}$, where q is a new intensional predicate.

Without loss of generality, we will consider only queries of the form $(P, q(\overline{x}))$, where q is an intensional predicate. Answering such a query on an extensional instance I is to find (correct) answers for $P \cup I \cup \{\leftarrow q(\overline{x})\}$.

3. GENERALIZING THE QSQR EVALUATION ALGORITHM

3.1. Informal Description

We first extend SLD-resolution with tabulation. We set up the problem as follows: given a positive logic program P, an extensional instance I and an atom A of an intensional predicate p, construct an answer relation ans_p such that for every SLD-refutation of $P \cup I \cup \{\leftarrow A\}$ with computed answer θ, $A\theta$ is an instance of a variant of some atom from ans_p (i.e., ans_p contains a more general answer than θ). The mentioned property is called completeness (of the evaluation method). We expect also soundness, which means that, for every atom A' of ans_p, $P \cup I \models \forall(A')$. The relation ans_p contains tuples (as for the predicate p) that are treated as atoms of p.

For each intensional predicate q, we use a global variable ans_q to keep an answer relation for q. Tuples of ans_q are treated as atoms of the predicate q. At the beginning, we set all of such variables to empty relations. Consider an SLD-refutation of $P \cup I \cup \{\leftarrow A\}$. Let the first input program clause applied to $\leftarrow A$ be $\varphi = (A' \leftarrow B_1, \ldots, B_n)$ and the used mgu (for A and A') be θ. The next goal is thus $\leftarrow (B_1, \ldots, B_n)\theta$.

Let $\delta_0 = \theta$. For each $1 \leq i \leq n$, we process $\leftarrow B_i\delta_{i-1}$ as follows, where δ_{i-1} is the substitution containing the bindings of variables after processing $\leftarrow B_{i-1}\delta_{i-2}$. Let p_i be the predicate of B_i.

[4] A *positive formula without quantifiers* is a formula built up from atoms using only connectives $\wedge$ and $\vee$.

(1) Case p_i is an extensional predicate: If γ_i is an mgu for $B_i\delta_{i-1}$ and a fresh variant of some atom from $I(p_i)$ then let $\delta_i := \delta_{i-1}\gamma_i$ and continue to process the next goal atom $B_{i+1}\delta_i$.
(2) Case p_i is an intensional predicate:
   (a) Recursively process $\leftarrow B_i\delta_{i-1}$ in the same way as for $\leftarrow A$. This task does not pass bindings of variables directly outside but it updates the answer relations that are global variables of the algorithm.
   (b) If γ_i is an mgu for $B_i\delta_{i-1}$ and a fresh variant of some atom from ans_p_i then let $\delta_i := \delta_{i-1}\gamma_i$ and continue to process the next goal atom $B_{i+1}\delta_i$.

Then δ_n holds a correct answer for $P \cup I \cup \{\leftarrow A\}$. Thus, if $A\delta_n$ is not an instance of a fresh variant of any atom from the answer relation ans_p, where p is the predicate of A, then we can add $A\delta_n$ to ans_p.

To obtain all answers for the goal $\leftarrow A$, all the choices are systematically tried, and the process is repeated until no changes were made to the global $ans\_$ variables during the last iteration of the main loop. To guarantee termination, in each iteration of the main loop, each goal like $\leftarrow A$ is processed only once. Furthermore, to avoid redundant recomputation we check that $\leftarrow A$ is not an instance of a fresh variant of any goal that has been processed during the current iteration of the main loop. To do this we record A in a relation held by a global variable $input\_p$, where p is the predicate of A. Such a relation is called an $input\_$ ***relation*** (or a ***goal relation***). It can be represented as a generalized relation and we treat tuples of $input\_p$ as atoms of the predicate p. The global $input\_$ variables are reset to empty relations for each iteration of the main loop.

Notice that we concentrate on unary goals $\leftarrow B_i\delta_{i-1}$ instead of $\leftarrow (B_i, \ldots, B_n)\delta_{i-1}$.

The described method follows the tuple-at-a-time approach. It will be formally presented and studied in the next two subsections. The set-at-a-time version of the method will be presented in Section 4.

Example 3.1. Consider the following query, in which x, y, z denote variables, and a, b, c, d, e, f, g denote constant symbols:

— program P:

$$p(x,y) \leftarrow q(x,y)$$
$$p(x,y) \leftarrow p(x,z), q(z,y)$$
$$r(x) \leftarrow p(a,x)$$

— extensional instance I:

$$q(a,b) \quad q(b,d) \quad q(d,f)$$
$$q(a,c) \quad q(c,e) \quad q(e,g)$$

— query: $r(x)$.

The intensional predicate p is defined to be the transitive closure of the extensional relation q. Notice the order of atoms in the body of the second clause defining p.

Consider evaluation of this query by using the described tuple-at-a-time method. At the beginning, the global variables ans_r and ans_p are set to empty relations. The first iteration of the main loop of the evaluation process executes the steps illustrated by the SLD-like tree given in Figure 1(a), and the second iteration of that loop executes the steps illustrated by the SLD-like tree given in Figure 2(a). The third iteration of that main loop does not change value of any $ans\_$ variable and the process terminates with $ans\_r = \{b, c, d, e, f, g\}$.

In the case the set-at-a-time technique is used, "similar" goals are processed together and the first and second iterations of the main loop would execute the steps illustrated by Figure 3(a) and Figure 3(c), respectively.

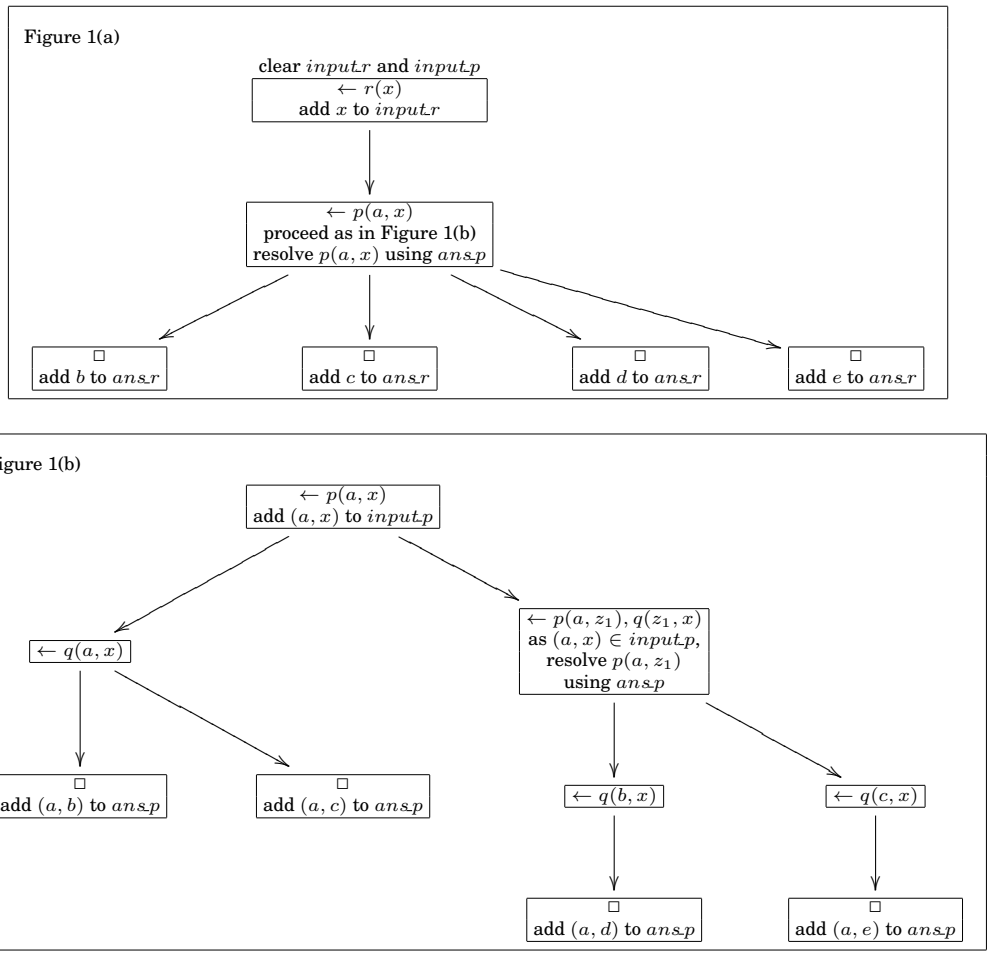


Fig. 1. An illustration for Example 3.1.

Consider the case when the global $input\_$ relations are not cleared (i.e., emptied) for subsequent iterations of the main loop of the evaluation process. Then, at the second iteration, as x belongs to $input\_r$, the processing of the goal $\leftarrow r(x)$ (tuple-at-a-time) or the goal relation $\{r(x)\}$ (set-at-a-time) exits immediately without changing the $ans\_$ relations. Consequently, the evaluation process terminates with an incomplete set of answers $ans\_r = \{b, c, d, e\}$. Even in the case we treat $r(x)$ in a special way by ignoring the fact $x \in input\_r$, the presence of (a, x) in $input\_p$ will cause similar effects.

3.2. A Formal Tuple-at-a-Time Version of the Evaluation Method

Let l be a fixed natural number, which we will use as the bound imposed on term-depth of atoms and substitutions occurring in evaluation of queries. Algorithm 1 given on page 11 is a formal presentation of the evaluation method described in the previous subsection.

Example A.1 in the electronic appendix of this paper demonstrates the run of Algorithm 1 on a Datalog query taken from [Nejdl 1987] (which is similar to the "reverse-same-generation" query used in [Abiteboul et al. 1995]). It is convenient for a better

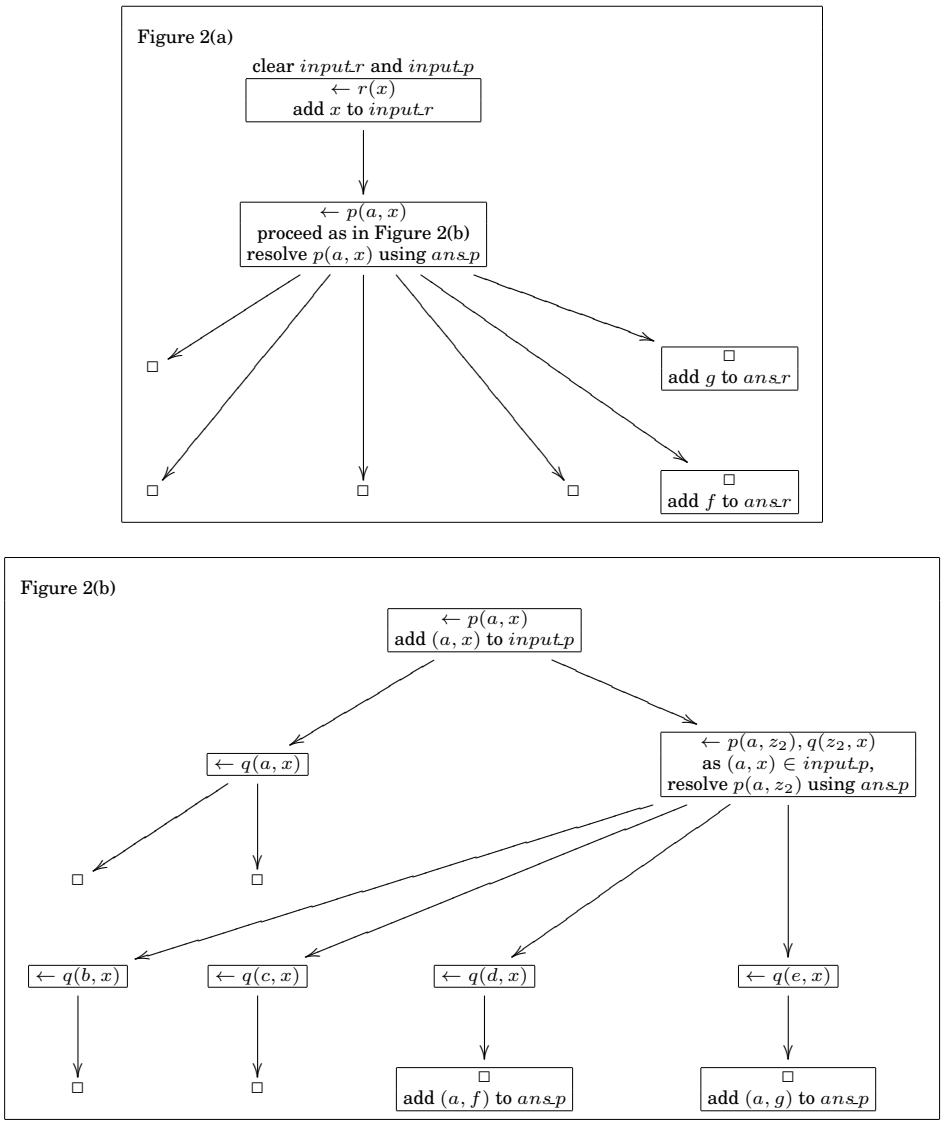


Fig. 2. An illustration for Example 3.1.

understanding of the algorithm, but too long for putting here. Note that our algorithm works not only for Datalog queries but also for queries to Horn knowledge bases.

Remark 3.2. If we change Algorithm 1 by moving the call `clear-input-vars` from the inside of the "repeat" loop to the place before the loop then it becomes incomplete. This was illustrated in Example 3.1 and can be checked by using the implementation [Nguyen 2011]. Without clearing the global $input\_$ relations for subsequent iterations of the main loop there are situations when $ans\_$ atoms derived in some earlier steps cannot be exploited for the currently considered subquery to derive further results because the subquery is subsumed by a previously considered subquery and is then omitted. In other words, since the QSQR evaluation procedure is specified as a

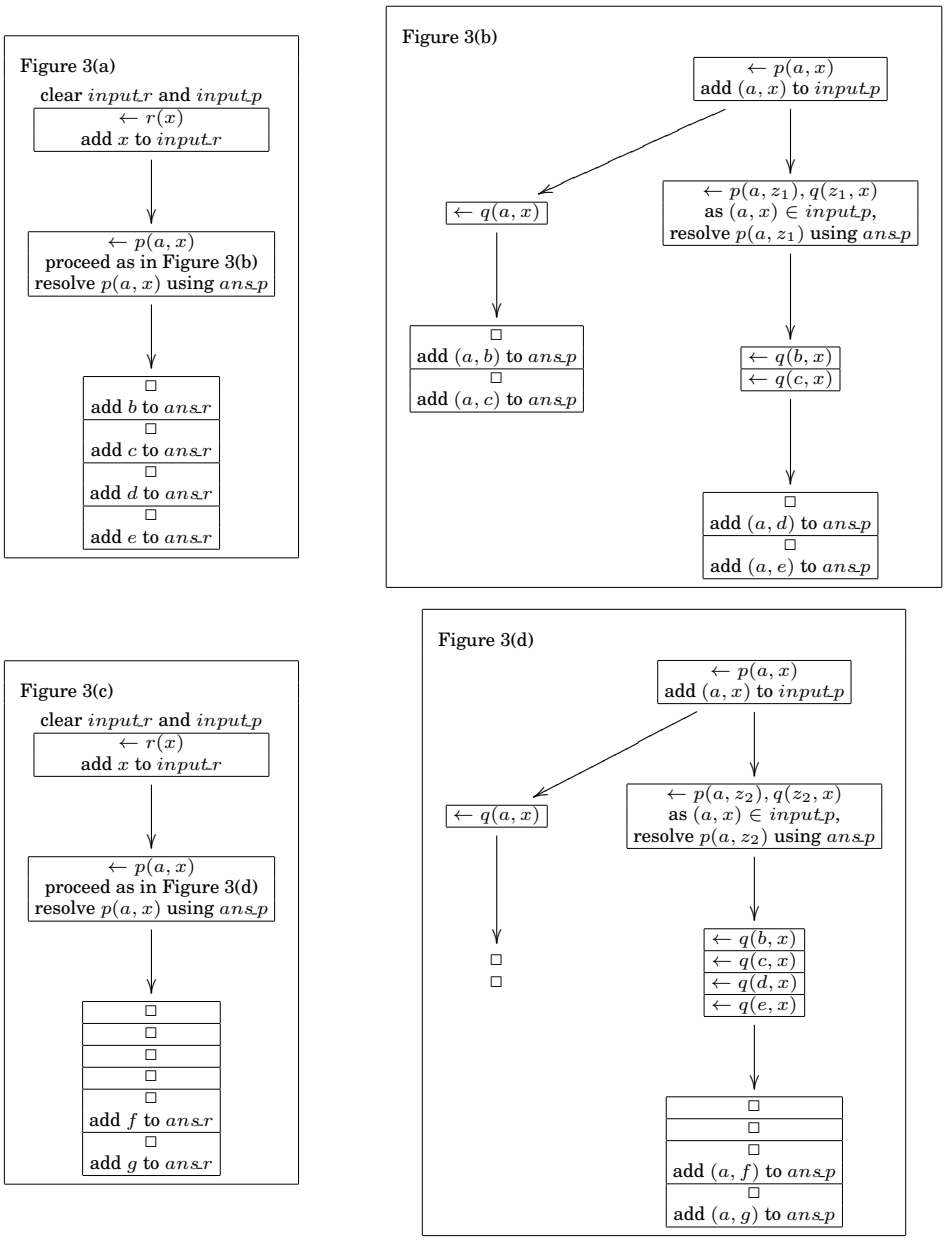


Fig. 3. An illustration for Example 3.1.

recursive function, newly derived $ans\_$ atoms are not directly propagated to all recursive calls. That is, the intermediary $ans\_$ relations are somehow local to each recursive call although the $ans\_$ variables are global. This leads to the need to clear the $input\_$ relations occasionally (e.g., at the beginning of each iteration of the main loop as in Algorithm 1, or after/before each recursive call) in order to allow recomputations using updated $ans\_$ relations. Sometimes such recomputations are redundant, e.g., as in the case of the leftmost evaluation branches of Figures 2(a) and 2(b). As observed by

ALGORITHM 1: for evaluating a query $(P, q(\overline{x}))$ on an extensional instance I.

```
init-ans-vars;
repeat
    clear-input-vars;
    process-goal (q(x̄));
until ans_ variables were not changed during the last iteration;
return ans_q;
```

Procedure init-ans-vars

foreach *intensional predicate* p *of* P **do**
    set the global variable ans_p to the empty relation;

Procedure clear-input-vars

foreach *intensional predicate* p *of* P **do**
    set the global variable $input\_p$ to the empty relation;

Procedure process-goal(A)

```
/* for processing the goal ← A                                   */
```

let p be the predicate of A;

if A *is an instance of a fresh variant of some atom from* $input\_p$ **then exit**
else add A to $input\_p$;

foreach *program clause* φ *defining* p *in* P **do**
    process-goal-using-clause (A, a fresh variant of φ) ; `// defined on page 12`

Vieille [Vieille 1989], the QSQR evaluation method is like iterative deepening search. It has both advantages and disadvantages.

3.3. Soundness and Completeness

In this subsection, we prove that the top-down evaluation method presented by Algorithm 1 is sound and complete. Roughly speaking, Algorithm 1 is a reformulation of SLD-resolution with a different way of passing bindings of variables. Our proofs are therefore related to soundness and completeness of the SLD-resolution calculus.

THEOREM 3.3 (SOUNDNESS). *After a run of Algorithm 1 on a query* $(P, q(\overline{x}))$ *and an extensional instance* I*, for all intensional predicates* p *of* P*, every computed answer* $A'' \in ans\_p$ *is a correct answer in the sense that* $P \cup I \models \forall(A'')$.

PROOF. We prove $P \cup I \models \forall(A'')$ by induction on the number of the step at which $A'' = A\delta_n$ is added to ans_p in Step 23 of an execution of procedure process-goal-using-clause for A and $\varphi = (A' \leftarrow B_1, \ldots, B_n)$. Let $\delta_0, \ldots, \delta_{n-1}$ and $\gamma_1, \ldots, \gamma_n$ be the substitutions that were used to compute δ_n. We have that $\delta_i = (\delta_{i-1}\gamma_i)_{|X}$ for $1 \leq i \leq n$. Hence, $\delta_n = (\delta_{i-1}\gamma_i \ldots \gamma_n)_{|X}$ for $1 \leq i \leq n$.

We will construct an SLD-refutation for the goal $\leftarrow B_1\delta_n, \ldots, B_n\delta_n$ by tracing the mentioned execution of process-goal-using-clause.

Consider Step 12 of procedure process-goal-using-clause. Since γ_i is an mgu for $B_i\delta_{i-1}$ and a variant of $B'_i \in I(p_i)$, $B_i\delta_n = B_i\delta_{i-1}\gamma_i \ldots \gamma_n$ is an instance of a variant of

Procedure process-goal-using-clause(A, φ)

```
/* for processing ← A using the program clause φ and the term-depth bound l      */
1  let X = Var(A) ∪ Var(φ) and φ = (A' ← B_1, ..., B_n);
2  if A and A' are unifiable using an mgu δ_0 then
3      if term-depth(δ_0) ≤ l then
4          sup_0 := {δ_0} ;        // sup_i denotes the so called ith ''supplementary'' relation
5          foreach i from 1 to n do
6              let p_i be the predicate of B_i;
7              sup_i := ∅;
8              foreach δ_{i-1} ∈ sup_{i-1} do
9                  if term-depth(B_i δ_{i-1}) ≤ l then
10                     if p_i is an extensional predicate then
11                         foreach B'_i ∈ I(p_i) do
12                             if B_i δ_{i-1} is unifiable with a fresh variant of B'_i using an mgu γ_i then
13                                 if term-depth((δ_{i-1} γ_i)_|X) ≤ l then add (δ_{i-1} γ_i)_|X to sup_i;
14                     else // p_i is an intensional predicate
15                         process-goal (B_i δ_{i-1}) ;                    // defined on page 11
16                         foreach B'_i ∈ ans_p_i do
17                             if B_i δ_{i-1} is unifiable with a fresh variant of B'_i using an mgu γ_i then
18                                 if term-depth((δ_{i-1} γ_i)_|X) ≤ l then add (δ_{i-1} γ_i)_|X to sup_i;
19             let p be the predicate of A;
20             foreach δ_n ∈ sup_n do
21                 if Aδ_n is not an instance of a fresh variant of any atom from ans_p then
22                     delete from ans_p every atom whose fresh variant is an instance of Aδ_n;
23                     add Aδ_n to ans_p ;
```

B'_i. Hence $P \cup I \cup \{\leftarrow B_i\delta_n\}$ has an SLD-refutation with the empty substitution as the computed answer.

Consider Step 17 of procedure process-goal-using-clause. Since γ_i is an mgu for $B_i\delta_{i-1}$ and a variant of $B'_i \in ans\_p_i$, $B_i\delta_n = B_i\delta_{i-1}\gamma_i \ldots \gamma_n$ is an instance of a variant of $B'_i \in ans\_p_i$. By the inductive assumption, $P \cup I \models \forall(B'_i)$, and hence $P \cup I \models \forall(B_i\delta_n)$. By Lemma 2.3, it follows that $P \cup I \cup \{\leftarrow B_i\delta_n\}$ has an SLD-refutation with the empty substitution as the computed answer.

The refutations with empty computed answers of $P \cup I \cup \{\leftarrow B_i\delta_n\}$ for $1 \leq i \leq n$ can be combined into an SLD-refutation of $P \cup I \cup \{\leftarrow B_1\delta_n, \ldots, B_n\delta_n\}$ with an empty computed answer. By Theorem 2.1 on soundness of SLD-resolution, we have that $P \cup I \models \forall((B_1 \wedge \ldots \wedge B_n)\delta_n)$. It follows that $P \cup I \models \forall(A'\delta_n)$. Since $A\delta_0 = A'\delta_0$ and $\delta_n = (\delta_0\gamma_1 \ldots \gamma_n)_{|X}$, we also have that $A\delta_n = A'\delta_n$. Therefore $P \cup I \models \forall(A\delta_n)$, which completes the proof. □

We need the following lemma for the completeness theorem. We assume that the sets of fresh variables used for renaming variables of input program clauses in SLD-refutations and in Algorithm 1 are disjoint.

LEMMA 3.4. *After a run of Algorithm 1 (using parameter l) on a query $(P, q(\overline{x}))$ and an extensional instance I, for every intensional predicate p of P, for every tuple $\overline{t} \in input\_p$ and for every SLD-refutation of $P \cup I \cup \{\leftarrow p(\overline{t})\}$ that uses the leftmost selection function and does not contain any goal with term-depth greater than l, if $\theta_1, \ldots, \theta_h$ are*

the mgu's used in the refutation and the term-depth of $\theta_1 \dots \theta_h$ is not greater than l then there exists a tuple $\bar{t}' \in ans\_p$ such that $p(\bar{t})\theta_1 \dots \theta_h$ is an instance of a variant of $p(\bar{t}')$.

PROOF. We prove this lemma by induction on the length of the mentioned SLD-refutation. Let $A = p(\bar{t})$ and suppose that the first step of the refutation of $P \cup I \cup \{\leftarrow A\}$ uses an input program clause $\varphi' = (A'' \leftarrow B''_1, \dots, B''_n)$, giving the resolvent $\leftarrow (B''_1, \dots, B''_n)\theta_1$. Let $j_1 = 2$, $j_{n+1} = h + 1$ and suppose that, for $1 \leq i \leq n$,

$$\begin{array}{l}\text{the fragment for processing } \leftarrow B''_i\theta_1 \dots \theta_{j_i-1} \text{ of the refutation}\\ \text{of } P \cup I \cup \{\leftarrow A\} \text{ uses mgu's } \theta_{j_i}, \dots, \theta_{j_{i+1}-1}.\end{array} \tag{1}$$

Thus, after processing the atom B''_{i-1}, for $2 \leq i \leq n+1$, the next goal of the refutation of $\leftarrow A$ is $\leftarrow (B''_i, \dots, B''_n)\theta_1 \dots \theta_{j_i-1}$. (If $i = n + 1$ then the goal is empty.)

Consider the last iteration of the main loop of Algorithm 1, the execution of `process-goal`(A) in that iteration that adds $\bar{t}$ to $input\_p$, and the execution of `process-goal-using-clause`(A, φ) in that execution of `process-goal`(A), where $\varphi = (A' \leftarrow B_1, \dots, B_n)$ is a variant of φ'. Let $\varphi' = \varphi\varrho$, where ϱ is a renaming substitution that uses only variables of φ and φ'. Thus,

$$(A'' \leftarrow B''_1, \dots, B''_n) = (A' \leftarrow B_1, \dots, B_n)\varrho. \tag{2}$$

Since θ_1 is an mgu for A and $A'' = A'\varrho$, we have that $A\theta_1 = A'\varrho\theta_1$. Since ϱ does not use variables of A, we have that $A\varrho = A$. Hence $A\varrho\theta_1 = A\theta_1 = A'\varrho\theta_1$. That is, $\varrho\theta_1$ is a unifier for A and A'. Since δ_0 is an mgu for A and A', it follows that $\varrho\theta_1 = \delta_0\gamma'_0$ for some substitution γ'_0.

As an inner induction, let the induction hypothesis be that after processing the first $i - 1$ iterations of the "foreach" loop in Step 5 of the considered execution of `process-goal-using-clause`(A, φ), where $1 \leq i \leq n + 1$, it holds that

$$(\varrho\theta_1 \dots \theta_{j_i-1})_{|X} = (\delta_{i-1}\gamma'_{i-1})_{|X} \tag{3}$$

for some $\delta_{i-1} \in sup_{i-1}$ and some substitution γ'_{i-1}. This induction hypothesis holds for $i = 1$ because $j_1 = 2$, $\varrho\theta_1 = \delta_0\gamma'_0$ and the term-depth of $\varrho\theta_1$ is not greater than l. Suppose that the induction hypothesis holds for some $1 \leq i \leq n$. We show that it also holds for $i + 1$.

By (2) and the inductive assumption (3), we have that:

$$(\leftarrow B''_i\theta_1 \dots \theta_{j_i-1}) = (\leftarrow B_i\varrho\theta_1 \dots \theta_{j_i-1}) = (\leftarrow B_i\delta_{i-1}\gamma'_{i-1}). \tag{4}$$

Since the term-depth of $B_i\delta_{i-1}\gamma'_{i-1} = B''_i\theta_1 \dots \theta_{j_i-1}$ is not greater than l, the term-depth of $B_i\delta_{i-1}$ is also not greater than l. By (1), (4) and Lifting Lemma 2.2, we have that

$$\begin{array}{l}\text{there exists a refutation of } P \cup I \cup \{\leftarrow B_i\delta_{i-1}\} \text{ using the leftmost se-}\\ \text{lection function and mgu's } \theta'_{j_i}, \dots, \theta'_{j_{i+1}-1} \text{ such that the term-depths of}\\ \text{the goals and the composition } \theta'_{j_i} \dots \theta'_{j_{i+1}-1} \text{ are not greater than } l \text{ and}\\ \gamma'_{i-1}\theta_{j_i} \dots \theta_{j_{i+1}-1} = \theta'_{j_i} \dots \theta'_{j_{i+1}-1}\mu_i \text{ for some substitution } \mu_i.\end{array} \tag{5}$$

Consider the case when the predicate p_i of B_i is an extensional predicate.

Thus,

$$j_{i+1} = j_i + 1 \tag{6}$$

and

$$B_i\delta_{i-1}\theta'_{j_i} = B'_i\sigma\theta'_{j_i} \tag{7}$$

where $B'_i\sigma$ is the input program clause used for resolving $\leftarrow B_i\delta_{i-1}$, with $B'_i \in I(p_i)$ and σ being a renaming substitution. Let σ' be the renaming substitution used in Step 12

of procedure `process-goal-using-clause` for making a variant $B'_i\sigma'$ of B'_i for unifying with $B_i\delta_{i-1}$. We have that $\sigma = \sigma'\sigma''$ for some renaming substitution σ'' which does not use variables of B_i, δ_{i-1} and X. Thus $B_i\delta_{i-1}\sigma''\theta'_{j_i} = B_i\delta_{i-1}\theta'_{j_i}$, and by using (7) and the fact that $\sigma = \sigma'\sigma''$, we have that

$$(B_i\delta_{i-1})\sigma''\theta'_{j_i} = B_i\delta_{i-1}\theta'_{j_i} = B'_i\sigma\theta'_{j_i} = (B'_i\sigma')\sigma''\theta'_{j_i}.$$

Hence, $B_i\delta_{i-1}$ and $B'_i\sigma'$ are unifiable using $\sigma''\theta'_{j_i}$, and γ_i is an mgu for them (Step 12 of `process-goal-using-clause`). Hence

$$\sigma''\theta'_{j_i} = \gamma_i\mu'_i \tag{8}$$

for some substitution μ'_i. Let

$$\gamma'_i = \mu'_i\mu_i. \tag{9}$$

We have that:

$$\begin{array}{lll}
 & (\varrho\theta_1 \dots \theta_{j_{i+1}-1})_{|X} & \\
= & ((\varrho\theta_1 \dots \theta_{j_i-1})_{|X}\theta_{j_i} \dots \theta_{j_{i+1}-1})_{|X} & \\
= & ((\delta_{i-1}\gamma'_{i-1})_{|X}\theta_{j_i} \dots \theta_{j_{i+1}-1})_{|X} & \text{(by the inner inductive assumption (3))} \\
= & (\delta_{i-1}\gamma'_{i-1}\theta_{j_i} \dots \theta_{j_{i+1}-1})_{|X} & \\
= & (\delta_{i-1}\theta'_{j_i} \dots \theta'_{j_{i+1}-1}\mu_i)_{|X} & \text{(by (5))} \\
= & (\delta_{i-1}\sigma''\theta'_{j_i} \dots \theta'_{j_{i+1}-1}\mu_i)_{|X} & \text{(since } \sigma'' \text{ does not use variables of } \delta_{i-1}, X) \\
= & (\delta_{i-1}\gamma_i\mu'_i\mu_i)_{|X} & \text{(by (6) and (8))} \\
= & (\delta_{i-1}\gamma_i\gamma'_i)_{|X} & \text{(by (9)).}
\end{array}$$

Since the term-depth of $\theta_1 \dots \theta_h$ is not greater than l and ϱ is a renaming substitution, the term-depth of $(\varrho\theta_1 \dots \theta_{j_{i+1}-1})_{|X}$ is not greater than l. It follows that the term-depth of $(\delta_{i-1}\gamma_i)_{|X}$ is also not greater than l. Hence, for $\delta_i = (\delta_{i-1}\gamma_i)_{|X} \in sup_i$, we have that $(\varrho\theta_1 \dots \theta_{j_{i+1}-1})_{|X} = (\delta_i\gamma'_i)_{|X}$. That is, the induction hypothesis of the inner induction holds for $i+1$.

Now consider the case when the predicate p_i of B_i is an intensional predicate.

We first show that ans_p_i contains an atom B'_i such that $B_i\delta_{i-1}\theta'_{j_i} \dots \theta'_{j_{i+1}-1}$ is an instance of a variant of B'_i. As `process-goal`$(B_i\delta_{i-1})$ was called in Step 15 of the considered execution of `process-goal-using-clause`(A, φ), $input\_p_i$ must contain an atom $B^\diamond_i$ such that $B_i\delta_{i-1}$ is an instance of a variant of $B^\diamond_i$. Let α be a substitution such that

$$B_i\delta_{i-1} = B^\diamond_i\alpha \tag{10}$$

and α uses only variables from $B_i\delta_{i-1}$ and $B^\diamond_i$. By (5) and Lifting Lemma 2.2, it follows that there exists a refutation of $P \cup I \cup \{\leftarrow B^\diamond_i\}$ using the leftmost selection function and mgu's $\theta''_{j_i}, \dots, \theta''_{j_{i+1}-1}$ such that the term-depths of the goals and the composition $\theta''_{j_i} \dots \theta''_{j_{i+1}-1}$ are not greater than l and

$$\alpha\theta'_{j_i} \dots \theta'_{j_{i+1}-1} = \theta''_{j_i} \dots \theta''_{j_{i+1}-1}\beta \tag{11}$$

for some substitution β. By the outer inductive assumption, ans_p_i contains an atom B'_i such that $B^\diamond_i\theta''_{j_i} \dots \theta''_{j_{i+1}-1}$ is an instance of a variant of B'_i. Since

$$\begin{array}{rcll}
B_i\delta_{i-1}\theta'_{j_i} \dots \theta'_{j_{i+1}-1} & = & B^\diamond_i\alpha\theta'_{j_i} \dots \theta'_{j_{i+1}-1} & \text{(by (10))} \\
 & = & B^\diamond_i\theta''_{j_i} \dots \theta''_{j_{i+1}-1}\beta & \text{(by (11)),}
\end{array}$$

it follows that

$$B_i\delta_{i-1}\theta'_{j_i} \dots \theta'_{j_{i+1}-1} \text{ is also an instance of a variant of } B'_i. \tag{12}$$

Let σ be the renaming substitution used in Step 17 of procedure `process-goal-using-clause` for making a variant $B'_i\sigma$ of B'_i for unifying with $B_i\delta_{i-1}$. The atom $B'_i\sigma$ does not contain variables of X, δ_{i-1} and $\theta'_{j_i} \dots \theta'_{j_{i+1}-1}$. Hence, by (12), $B_i\delta_{i-1}\theta'_{j_i} \dots \theta'_{j_{i+1}-1}$ is an instance of $B'_i\sigma$. Let ρ be a substitution with domain contained in $Var(B'_i\sigma)$ such that $B_i\delta_{i-1}\theta'_{j_i} \dots \theta'_{j_{i+1}-1} = B'_i\sigma\rho$. We have that $\theta'_{j_i} \dots \theta'_{j_{i+1}-1} \cup \rho$ is a unifier for $B_i\delta_{i-1}$ and $B'_i\sigma$. As γ_i is an mgu for $B_i\delta_{i-1}$ and $B'_i\sigma$, we have that $\gamma_i\mu'_i = (\theta'_{j_i} \dots \theta'_{j_{i+1}-1} \cup \rho)$ for some substitution μ'_i. Hence

$$(\gamma_i\mu'_i)_{|X \cup Var(\delta_{i-1})} = (\theta'_{j_i} \dots \theta'_{j_{i+1}-1})_{|X \cup Var(\delta_{i-1})}. \tag{13}$$

Let

$$\gamma'_i = \mu'_i\mu_i. \tag{14}$$

We have that:

$$\begin{aligned}
& (\varrho\theta_1 \dots \theta_{j_{i+1}-1})_{|X} \\
= \ & (\delta_{i-1}\theta'_{j_i} \dots \theta'_{j_{i+1}-1}\mu_i)_{|X} \text{ (as for the case when } p_i \text{ is an extensional predicate)} \\
= \ & (\delta_{i-1}(\theta'_{j_i} \dots \theta'_{j_{i+1}-1})_{|X \cup Var(\delta_{i-1})}\mu_i)_{|X} \\
= \ & (\delta_{i-1}(\gamma_i\mu'_i)_{|X \cup Var(\delta_{i-1})}\mu_i)_{|X} \text{ (by (13))} \\
= \ & (\delta_{i-1}\gamma_i\mu'_i\mu_i)_{|X} \\
= \ & (\delta_{i-1}\gamma_i\gamma'_i)_{|X} \text{ (by (14)).}
\end{aligned}$$

Analogously as for the case when p_i is an extensional predicate, the term-depth of $(\delta_{i-1}\gamma_i)_{|X}$ is not greater than l, and for $\delta_i = (\delta_{i-1}\gamma_i)_{|X} \in sup_i$, the induction hypothesis of the inner induction holds for $i+1$.

We have proved the induction hypothesis of the inner induction, which implies that $(\varrho\theta_1 \dots \theta_{j_{n+1}-1})_{|X} = (\delta_n\gamma'_n)_{|X}$. That is, $(\varrho\theta_1 \dots \theta_h)_{|X} = (\delta_n\gamma'_n)_{|X}$. Hence $A\delta_n\gamma'_n = A\varrho\theta_1 \dots \theta_h = A\theta_1 \dots \theta_h$ (since $A\varrho = A$). Hence $p(\bar{t})\theta_1 \dots \theta_h = A\theta_1 \dots \theta_h$ is an instance of $A\delta_n$. By Step 21 of procedure `process-goal-using-clause`, it follows that $p(\bar{t})\theta_1 \dots \theta_h$ is an instance of a variant of some atom from ans_p. □

THEOREM 3.5 (COMPLETENESS). *After a run of Algorithm 1 (using parameter l) on a query $(P, q(\overline{x}))$ and an extensional instance I, for every SLD-refutation of $P \cup I \cup \{\leftarrow q(\overline{x})\}$ that uses the leftmost selection function and does not contain any goal with term-depth greater than l, if $\theta_1, \dots, \theta_h$ are the mgu's used in the refutation and the term-depth of the composition $\theta_1 \dots \theta_h$ is not greater than l then there exists a tuple $\bar{t} \in ans\_q$ such that $\overline{x}\theta_1 \dots \theta_h$ is an instance of a variant of $\bar{t}$.*

This theorem immediately follows from Lemma 3.4. Together with Theorem 2.1 (on completeness of SLD-resolution) it makes a relationship between correct answers of $P \cup I \cup \{\leftarrow q(\overline{x})\}$ and the answers computed by Algorithm 1 for the query $(P, q(\overline{x}))$ on the extensional instance I.

Note that in the above theorem $\overline{x}\theta_1 \dots \theta_h$ is an instance of a variant of $\bar{t}$ but is neither $\bar{t}$ nor a variant of $\bar{t}$ because of the optimization made in Step 21 of procedure `process-goal-using-clause`. For knowledge bases, it is inessential to require $\overline{x}\theta_1 \dots \theta_h$ to be $\bar{t}$ or a variant of $\bar{t}$.

For queries and extensional instances without function symbols, we take term-depth bound $l = 0$ and obtain the following completeness result, which immediately follows from the above theorem.

COROLLARY 3.6. *After a run of Algorithm 1 using $l = 0$ on a query $(P, q(\overline{x}))$ and an extensional instance I that do not contain function symbols, for every computed answer θ of an SLD-refutation of $P \cup I \cup \{\leftarrow q(\overline{x})\}$ that uses the leftmost selection function, there exists a tuple $\bar{t} \in ans\_q$ such that $\overline{x}\theta$ is an instance of a variant of $\bar{t}$.*

4. DOING IT SET-AT-A-TIME

Operations for databases and knowledge bases are often done set-at-a-time instead of tuple-at-a-time in order to reduce the number of accesses to secondary storage. This approach allows various optimizations like sorting, indexing, and clustering.

ALGORITHM 2: for evaluating a query $(P, q(\overline{x}))$ on an extensional instance I.

```
1  init-ans-vars;
2  repeat
3  |  clear-input-vars;
4  |  s-process-goal ({q(x̄)});
5  until ans_ variables were not changed during the last iteration;
6  return ans_q;
```

Procedure s-process-goal(J)

```
/* for processing the goal relation J                                   */
1  let p be the predicate of J;
2  J := eliminate-subsumed-tuples(J, input_p) ;          // defined on page 17
3  if J is empty then exit;
4  input_p := input_p ∪ J;
5  foreach program clause φ defining p in P do
6  |  s-process-goal-using-clause (J, a fresh variant of φ);
```

Procedure s-process-goal-using-clause(J, φ)

```
/* for processing the goal relation J using the program clause φ and the term-depth
   bound l                                                              */
1  let Y = Var(φ);
2  let p be the predicate of J;
3  let φ = (A' ← B_1, ..., B_n);                          // A' is an atom of p
4  K := resolve-using-head-atom(J, A');                   // defined on page 17
5  i := 0;
6  while i < n and K is not empty do
7  |  i := i + 1;
8  |  if the predicate p_i of B_i is an extensional predicate then
9  |  |  K := resolve-using-body-atom(K, B_i, I(p_i), Y);   // defined on page 17
10 |  else
11 |  |  s-process-goal ({B_iδ_{i-1} | (A, δ_{i-1}) ∈ K and term-depth(B_iδ_{i-1}) ≤ l});
12 |  |  K := resolve-using-body-atom(K, B_i, ans_p_i, Y);
13 ans_p := merge(ans_p, {Aδ_n | (A, δ_n) ∈ K});          // defined on page 17
```

Algorithm 2 is our reformulation of Algorithm 1 using the set-at-a-time technique. The reformulation is based on processing a set of goal atoms of the same predicate instead of processing a single goal atom. Example A.2 in the electronic appendix of this paper demonstrates a run of Algorithm 2. It is also convenient for a better understanding of the algorithm (but too long for putting here).

Algorithm 1 is a tuple-at-a-time method and can be looked at as a combination of depth-first search and tabulation. Algorithm 2 is a set-at-a-time method and can be looked at as a mixture of depth-first search, breadth-first search and tabulation.

Function eliminate-subsumed-tuples(J, J')

Input: generalized relations J and J' of the same arity
Output: the set of maximally general tuples of J not subsumed by J'

```
1  J_2 := ∅;
2  foreach A ∈ J do
3  |  include := true;
4  |  foreach A' ∈ J' ∪ J_2 do
5  |  |  let A'' be a fresh variant of A';
6  |  |  if A is an instance of A'' then {include := false, break };
7  |  if include = true then
8  |  |  delete from J_2 all tuples that are an instance of a fresh variant of A;
9  |  |  J_2 := J_2 ∪ {A};
10 return J_2;
```

Function merge(J, J')

```
1  J_2 := eliminate-subsumed-tuples(J, J');
2  J_2' := eliminate-subsumed-tuples(J', J_2);
3  return J_2 ∪ J_2';
```

Function resolve-using-head-atom(J, A')

Input: a goal relation (i.e. an $input\_$ relation) J associated with the predicate of A'
Output: the set consisting of a tuple (A, δ_0) for each atom $A \in J$ such that A is unifiable with A' using mgu δ_0 with term-depth not greater than l

```
1  K := ∅;
2  foreach A ∈ J do
3  |  if A and A' are unifiable using an mgu δ_0 then
4  |  |  if term-depth(δ_0) ≤ l then K := K ∪ {(A, δ_0)};
5  return K;
```

Function resolve-using-body-atom(K, B_i, R, Y)

Input: K is a set of tuples of the form (A, δ_{i-1}), B_i is an atom, R is a generalized relation associated with the predicate of B_i, and Y is a set of variables
Output: the set consisting of a tuple (A, δ_i) for each $(A, \delta_{i-1}) \in K$ and each $B'_i \in R$ such that $B_i\delta_{i-1}$ is unifiable with a fresh variant of B'_i using an mgu γ_i and the term-depth of $\delta_i = (\delta_{i-1}\gamma_i)_{|Var(A)\cup Y}$ is not greater than l

```
1  K' := ∅;
2  foreach (A, δ_{i-1}) ∈ K do
3  |  let X = Var(A) ∪ Y;
4  |  foreach B_i' ∈ R do
5  |  |  if B_iδ_{i-1} is unifiable with a fresh variant of B_i' using an mgu γ_i then
6  |  |  |  if term-depth((δ_{i-1}γ_i)_|X) ≤ l then K' := K' ∪ {(A, (δ_{i-1}γ_i)_|X)}
7  return K';
```

Algorithm 2 simulates Algorithm 1 but the order of calls of simulations of procedure `process-goal` (by using procedure `s-process-goal`) may be different. To avoid keeping unnecessary information it also uses the same variable K (in procedure `s-process-goal-using-clause`) for the whole sequence of sup_i. Analyzing the proofs

of Theorems 3.3 and 3.5, it can be seen that they can be adapted in a straightforward way for Algorithm 2 because the difference of steering control between the algorithms does not affect the proofs. Thus, we arrive at the following theorem, whose detailed proofs are given in Appendix via Theorems A.1, A.3 and Corollary A.4.

THEOREM 4.1. *Algorithm 2 is sound and complete (i.e., Theorems 3.3, 3.5 and Corollary 3.6 still hold when "Algorithm 1" is replaced by "Algorithm 2").*

4.1. Data Complexity

In this subsection we estimate the *data complexity* of Algorithm 2, which is measured w.r.t. the size of the extensional instance I when the query $(P, q(\overline{x}))$ and the term-depth bound l are fixed.

If terms are represented as sequences of symbols or as trees then there will be a problem with complexity. Namely, unifying the terms $f(x_1, \ldots, x_n)$ and $f(g(x_0, x_0), \ldots, g(x_{n-1}, x_{n-1}))$, we get a term of exponential length.[5] If the term-depth bound l is used in all steps, including the ones of unification, then the problem will not arise. But we do not want to be so restrictive.

To represent a term we use instead a rooted acyclic directed graph which is permitted to have multiple ordered arcs and caches nodes representing the same subterm. Such a graph will simply be called a DAG. As an example, the DAG of $f(x, a, x)$ has the root n_f labeled by f, a node n_x labeled by x, a node n_a labeled by a, and three ordered edges outgoing from n_f : the first and third ones are connected to n_x, while the second one is connected to n_a.

The *size of a term* t, denoted by $size(t)$, is defined to be the size of the DAG of t (i.e. the number of nodes and edges of the DAG of t). The sizes of other term-based expressions or data structures are defined as usual. For example, we define:

— the *size of an atom* $p(t_1, \ldots, t_k)$ to be $1 + size(t_1) + \ldots + size(t_k)$
— the *size of a set* J *of atoms* to be the sum of the sizes of its elements
— the *size of a substitution* $\{x_1/t_1, \ldots, x_k/t_k\}$ to be $k + size(t_1) + \ldots + size(t_k)$.

Using DAGs to represent terms, unification of two atoms A and A' can be done in polynomial time in the sizes of A and A'. In the case A and A' are unifiable, the resulting atom and the resulting mgu have sizes that are polynomial in the sizes of A and A'. Similarly, checking whether A is an instance of A' can also be done in polynomial time in the sizes of A and A'.

The following theorem estimates the data complexity of Algorithm 2, under the assumption that terms are represented by DAGs and unification and checking instances of atoms are done in polynomial time.

THEOREM 4.2. *For a fixed query and a fixed bound l on term-depth, Algorithm 2 runs in polynomial time in the size of the extensional instance.*

PROOF. Consider a run of Algorithm 2 using parameter l on a query $(P, q(\overline{x}))$ and on an extensional instance I with size n. Here, $(P, q(\overline{x}))$ and l are fixed. Thus, if $(A \leftarrow B_1, \ldots, B_k)$ is a program clause of P then k is bounded by a constant. Similarly, if p is an intensional predicate from P then the arity of p is also bounded by a constant.

Observe that all the functions `eliminate-subsumed-tuples`, `merge`, `resolve-using-head-atom`, `resolve-using-body-atom` run in polynomial time in the sizes of their parameters. Also observe that the sizes of $ans\_$ and $input\_$ relations are bounded by a polynomial of n. The reasons are:

[5] Another example is the pair $f(x_1, \ldots, x_n, x_1, \ldots, x_n)$ and $f(y_1, \ldots, y_n, g(y_0, y_0), \ldots, g(y_{n-1}, y_{n-1}))$.

— intensional predicates come from P
— constant symbols and function symbols come from P and I
— $input\_$ relations consist of tuples with term-depth bounded by l
— $ans\_$ relations consist of tuples with term-depth bounded by $2l$ (because $input\_$ relations consist of tuples with term-depth bounded by l and the used substitutions have term-depth bounded by l)
— each of these relations cannot contain two tuples which are a variant of each other (this means that names of variables occurring in tuples of $ans\_$ and $input\_$ relations can be "standardized").

For a similar reason, during the execution of Algorithm 2, relation J in Step 6 of procedure `s-process-goal` also has a polynomial size (in n). For this, note that `s-process-goal`(J) is called only for goal relations J consisting of atoms with term-depth not greater than l. Consequently, `s-process-goal-using-clause`(J, φ) is called only for goal relations J with a polynomial size (in n). Such a call, not counting the execution of `s-process-goal` in Step 11, runs in polynomial time in the sizes of J and φ, and hence in polynomial time in n.

Observe that `s-process-goal`(J) is called only for goal relations J of polynomial size in n. A call of `s-process-goal`, not counting recursive calls of itself, runs in polynomial time in the size of its parameter, and hence in polynomial time in n.

Observe that:

— tuples are never deleted from $ans\_$ relations
— at the beginning of each iteration of the main loop of Algorithm 2, $input\_$ variables are reset to empty relations, but in the rest of the iteration:
  — tuples are never deleted from $input\_$ relations
  — a recursive call of `s-process-goal`(J) is executed only when some tuples have been put into an $input\_$ relation during the considered iteration.

Since the sizes of $ans\_$ and $input\_$ relations are bounded by a polynomial of n, and a call of `s-process-goal`, not counting recursive calls of itself, runs in polynomial time in n, we conclude that Algorithm 2 runs in polynomial time in n. □

COROLLARY 4.3. *Algorithm 2 with term-depth bound $l = 0$ is a complete evaluation algorithm with PTIME data complexity for the class of queries over a signature without function symbols.*

This corollary follows from Theorem A.1 (on soundness), Corollary A.4 (on completeness) and the above theorem (on complexity).

4.2. Some Optimizations

Some optimizations can be adopted for both Algorithms 1 and 2. For example, one can orders atoms in bodies of input program clauses using certain criteria (it can be proved that such an optimization does not affect the completeness, because it is a kind of choosing a selection function). As another example, our algorithms can be improved by using tail recursion elimination. For this purpose, we simulate Ross' magic templates with right-recursion [Ross 1996] as follows:

— We use $input\_p(\bar{t})$ to simulate $magic(p(\bar{t}))$ and use $input\_p(\bar{t}, C)$ to simulate $query(p(\bar{t}), C)$, where $query$ is the special intensional predicate used by Ross [Ross 1996].[6]

[6]We use letter C instead of A [Ross 1996, Definition 4.1] to distinguish it from our $A = p(\bar{t})$.

	Disk reading (times)	Disk writing (times)
Test1(T)	58 (23+9+26)	18 (12+6)
Test1(S)	43 (17+6+20)	15 (9+6)
Test2(T)	1201 (410+252+539)	279 (156+123)
Test2(S)	187 (71+24+92)	69 (36+33)
Test3(T)	4181 (1410+902+1869)	949 (506+443)
Test3(S)	347 (131+44+172)	129 (66+63)
Test4(T)	207 (36+74+97)	21 (14+7)
Test4(S)	43 (18+6+19)	13 (8+5)

Fig. 4. The number of disk accesses for Test1 - Test4 of [Nguyen 2011]. The letter T (resp. S) in brackets means the test is done by using the tuple-at-a-time (resp. set-at-a-time) technique. A field with values "N (N1+N2+N3)" in the "Disk reading" column means the total number of read operations on disk, the numbers of read accesses to extensional/ $inp\_$ / $ans\_$ relations. Here, $inp\_$ and $ans\_$ relations are also stored on disk as extensional relations. Similarly, a field with values "N (N1+N2)" in the "Disk writing" column means the total number of write operations on disk, the numbers of write accesses to $inp\_$ / $ans\_$ relations.

— Subqueries are generated and called accordingly to item (1) of [Ross 1996, Definition 4.1].
— Storing tuples of $ans\_$ relations is done accordingly to item (2) of [Ross 1996, Definition 4.1]. For example, if the considered rule is $[p(\overline{t}) \leftarrow q_1(\overline{t_1}), \ldots, q_n(\overline{t_n})]$ and p is "right-recursive" but q_n is not, then after processing $q_1(\overline{t_1}), \ldots, q_n(\overline{t_n})$ for $input\_p(\overline{t}, C)$ results of the form $C\delta_n$ (which may differ from $A\delta_n = p(\overline{t})\delta_n$) will be stored in the $ans\_$ relation associated with the predicate of C (which may differ from ans_p). If q_n is also "right-recursive" then the results of that processing are not "materialized" (i.e. not stored).

4.3. Relaxing Term-Depth Bound

Suppose that we want to compute as many as possible but no more than k correct answers for a query $(P, q(\overline{x}))$ on an extensional instance I within time limit T. Then we can use iterative deepening search which iteratively increases term-depth bound for atoms and substitutions occurring in the computation as follows:

(1) Initialize term-depth bound l to 0 (or another small natural number).
(2) Run Algorithm 2 for evaluating $(P, q(\overline{x}))$ on I within the time limit.
(3) While ans_q contains less than k tuples and the time limit was not reached yet, do:
  (a) Increase term-depth bound l by 1.
  (b) Run Algorithm 2 without resetting the $ans\_$ global variables (i.e. without Step 1).
(4) Return ans_q.

Alternatively, instead of iterative deepening search one can use D&B-search discovered recently in [Brodt et al. 2009], which also runs in polynomial space (w.r.t. the maximal depth of visited nodes in the search space) but avoids repetitions. Which method is better depends on locations of solutions in the search space.

5. PRELIMINARY EXPERIMENTAL RESULTS

We have implemented a prototype [Nguyen 2011] in Prolog for our evaluation methods. In this section we compare the set-at-a-time version with the tuple-at-a-time version with respect to the number of accesses to secondary storage. We give also other remarks.

Test1 of [Nguyen 2011] is the one taken from [Nejdl 1987]. It is presented and manually explored in Example A.1 of the electronic appendix of this paper. Test1 uses extensional predicates $p = \{(c, d), (b, c), (c, b)\}$ and $q = \{(e, a), (a, i), (i, o)\}$. Test2 of [Nguyen

2011] is similar to Test1, but the extensional predicates p and q are specified as below, using $n = 9$:

$$p = \{(c, d)\} \cup \{(b_i, c), (c, b_i) \mid 0 \leq i \leq n\}$$
$$q = \{(e, a_0)\} \cup \{(a_i, b_i), (b_i, a_{i+1}) \mid 0 \leq i \leq n\}.$$

Test3 of [Nguyen 2011] is similar to Test2, but uses $n = 19$. Test4 of [Nguyen 2011] is the "reverse-same-generation" query of [Abiteboul et al. 1995]. The experimental results on these tests, which are presented in Figure 4, clearly show that the set-at-a-time technique significantly reduces the number of accesses to secondary storage. See [Nguyen 2011] for more details.

Test5 and Test6 of [Nguyen 2011] show that the order of clauses in the used logic program affects the computational time consumed by our evaluation algorithms. Like in logic programming, more "specific" program clauses should be processed first. Similarly, Test7 - Test9 of [Nguyen 2011] show that the order of atoms in the body of a program clause affects the computational time as in the case of logic programming.

6. CONCLUDING REMARKS

We have generalized the QSQR evaluation method to give the first set-oriented depth-first evaluation method for Horn knowledge bases. The resulting procedure closely simulates SLD-resolution to take advantages of the goal-directed approach and highly exploits set-at-a-time tabling to reduce the number of accesses to secondary storage. We have proved that our procedure is sound and complete.

Our generalization uses the steering control of the (corrected) QSQR method but does not use adornments and annotations. Instead of "input" and "answer" relations consisting of tuples of constant symbols, we use "input" and "answer" relations consisting of tuples of terms (which may contain variables and function symbols), and instead of "supplementary" relations consisting of tuples of constant symbols, we use "supplementary" relations consisting of substitutions. This technique was inspired by Nguyen's computational methods for modal deductive databases [Nguyen 2007]. To deal with function symbols, we propose to use iterative deepening search which iteratively increases term-depth bound for atoms and substitutions occurring in the computation. This search strategy differs from the usual iterative derivation-deepening search. It also essentially differs from the "subgoal generalization" technique of SLD-ALG [Vieille 1989]. An alternative way is to use D&B-search [Brodt et al. 2009]. When the term-depth bound is fixed, our evaluation procedure runs in polynomial time in the size of extensional relations.

The differences between Vieille's evaluation methods QSQR [Vieille 1986] and QoSaQ [Vieille 1989] for Datalog deductive databases were characterized by himself as follows [Vieille 1989, pages 47-48]:

> "SLD-AL trees are complex structures to search, as a lemma lem to be resolved against a non-admissible goal G may be proved only *after* G was created. Therefore, two solutions can be adopted.
>
> The first one, adopted in QoSaQ, consists in *constructively* searching the SLD-AL tree. This requires that the goal G be stored until all the relevant lemmas lem have been produced and resolved against G. [...]
>
> The second solution, adopted in QSQR and implemented in DedGin, tries to take maximal advantage of depth-first techniques (and of their ease of implementation) to search SLD and SLD-AL trees. In this approach, the goal G is *not* stored until all the lemmas have been found. Rather, G is repeatedly regenerated to be resolved each time, potentially with new lemmas. [...]

> This solution, which, for short, we call an *iterative-deepening* search [30] [7] involves *redundant computation*, as the same goal must be reconstructed again and again. On the other hand, it is *economical of space*, as we do not need to store the goal G.".

One can view the QoSaQ evaluation method as a connection between the QSQ approach and the magic-set approach.[8] Recall, however, that the specific techniques of QoSaQ like "instantiation pattern", "rule compilation", "projection" are heavily based on the range-restrictedness and function-free conditions.

Our generalized QSQR evaluation method is the first set-oriented depth-first evaluation method for Horn knowledge bases. As shown in Section 5, in comparison with the tuple-oriented depth-first approach, it significantly reduces the number of accesses to secondary storage. As future work, we intend to further improve the method, for example, by using some ideas of the QoSaQ and seminaive (bottom-up) evaluation methods, and apply our method for Datalog-like rule languages of Semantic Web [Cao et al. 2011a; 2011b].

APPENDIX

In this appendix we present and prove theorems about soundness and completeness of Algorithm 2. These theorems are counterparts, respectively, of Theorems 3.3 and 3.5, with "Algorithm 1" replaced by "Algorithm 2". A counterpart of Corollary 3.6 is also given. The proofs are very similar to the ones given for Algorithm 1. We present them here just to make the paper self-contained.

THEOREM A.1 (SOUNDNESS). *After a run of Algorithm 2 on a query $(P, q(\overline{x}))$ and an extensional instance I, for all intensional predicates p of P, every computed answer $A'' \in ans\_p$ is a correct answer in the sense that $P \cup I \models \forall(A'')$.*

PROOF. We prove $P \cup I \models \forall(A'')$ by induction on the number of the step at which $A'' = A\delta_n$ is added to ans_p in Step 13 of an execution of procedure `s-process-goal-using-clause` (given on page 16) for J and $\varphi = (A' \leftarrow B_1, \ldots, B_n)$. Let $\delta_0, \ldots, \delta_{n-1}$ and $\gamma_1, \ldots, \gamma_n$ be the substitutions used in functions `resolve-using-head-atom` and `resolve-using-body-atom` (given on page 17) for computing δ_n. We have that $\delta_i = (\delta_{i-1}\gamma_i)_{|X}$ for $1 \leq i \leq n$. Hence, $\delta_n = (\delta_{i-1}\gamma_i \ldots \gamma_n)_{|X}$ for $1 \leq i \leq n$.

We will construct an SLD-refutation for the goal $\leftarrow B_1\delta_n, \ldots, B_n\delta_n$ by tracing the mentioned execution of `s-process-goal-using-clause`.

Consider Step 9 of procedure `s-process-goal-using-clause` and the call `resolve-using-body-atom`$(K, B_i, I(p_i), Y)$. Since γ_i is an mgu for $B_i\delta_{i-1}$ and a variant of $B'_i \in I(p_i)$ (see Step 5 of function `resolve-using-body-atom` on page 17), $B_i\delta_n = B_i\delta_{i-1}\gamma_i \ldots \gamma_n$ is an instance of a variant of B'_i. Hence $P \cup I \cup \{\leftarrow B_i\delta_n\}$ has an SLD-refutation with the empty substitution as the computed answer.

Consider Step 12 of procedure `s-process-goal-using-clause` and the call `resolve-using-body-atom`(K, B_i, ans_p_i, Y). Since γ_i is an mgu for $B_i\delta_{i-1}$ and a variant of $B'_i \in ans\_p_i$ (see Step 5 of function `resolve-using-body-atom` on page 17), $B_i\delta_n = B_i\delta_{i-1}\gamma_i \ldots \gamma_n$ is an instance of a variant of $B'_i \in ans\_p_i$. By the inductive assumption, $P \cup I \models \forall(B'_i)$, and hence $P \cup I \models \forall(B_i\delta_n)$. By Lemma 2.3, it follows that $P \cup I \cup \{\leftarrow B_i\delta_n\}$ has an SLD-refutation with the empty substitution as the computed answer.

[7] Although it is not quite the same.

[8] Vieille also categorized the magic-set evaluation method as a "constructive search" of SLD-AL trees [Vieille 1989, page 41].

The refutations with empty computed answers of $P \cup I \cup \{\leftarrow B_i\delta_n\}$ for $1 \leq i \leq n$ can be combined into an SLD-refutation of $P \cup I \cup \{\leftarrow B_1\delta_n, \ldots, B_n\delta_n\}$ with an empty computed answer. By Theorem 2.1 on soundness of SLD-resolution, we have that $P \cup I \models \forall((B_1 \wedge \ldots \wedge B_n)\delta_n)$. It follows that $P \cup I \models \forall(A'\delta_n)$. Since $A\delta_0 = A'\delta_0$ (see Step 3 of function `resolve-using-head-atom` on page 17) and $\delta_n = (\delta_0\gamma_1 \ldots \gamma_n)_{|X}$, we also have that $A\delta_n = A'\delta_n$. Therefore $P \cup I \models \forall(A\delta_n)$, which completes the proof. □

The following lemma is a counterpart of Lemma 3.4. We also assume that the sets of fresh variables used for renaming variables of input program clauses in SLD-refutations and in Algorithm 2 are disjoint.

LEMMA A.2. *After a run of Algorithm 2 (using parameter l) on a query $(P, q(\overline{x}))$ and an extensional instance I, for every intensional predicate p of P, for every tuple $\bar{t} \in input\_p$ and for every SLD-refutation of $P \cup I \cup \{\leftarrow p(\bar{t})\}$ that uses the leftmost selection function and does not contain any goal with term-depth greater than l, if $\theta_1, \ldots, \theta_h$ are the mgu's used in the refutation and the term-depth of $\theta_1 \ldots \theta_h$ is not greater than l then there exists a tuple $\bar{t}' \in ans\_p$ such that $p(\bar{t})\theta_1 \ldots \theta_h$ is an instance of a variant of $p(\bar{t}')$.*

PROOF. We prove this lemma by induction on the length of the mentioned SLD-refutation. Let $A = p(\bar{t})$ and suppose that the first step of the refutation of $P \cup I \cup \{\leftarrow A\}$ uses an input program clause $\varphi' = (A'' \leftarrow B''_1, \ldots, B''_n)$, giving the resolvent $\leftarrow (B''_1, \ldots, B''_n)\theta_1$. Let $j_1 = 2$, $j_{n+1} = h + 1$ and suppose that, for $1 \leq i \leq n$,

$$\begin{array}{l}\text{the fragment for processing } \leftarrow B''_i\theta_1 \ldots \theta_{j_i-1} \text{ of the refutation}\\ \text{of } P \cup I \cup \{\leftarrow A\} \text{ uses mgu's } \theta_{j_i}, \ldots, \theta_{j_{i+1}-1}.\end{array} \tag{15}$$

Thus, after processing the atom B''_{i-1}, for $2 \leq i \leq n + 1$, the next goal of the refutation of $\leftarrow A$ is $\leftarrow (B''_i, \ldots, B''_n)\theta_1 \ldots \theta_{j_i-1}$. (If $i = n + 1$ then the goal is empty.)

Consider the last iteration of the main loop of Algorithm 2, the execution of `s-process-goal` in that iteration that adds $\bar{t}$ to $input\_p$, and the execution of `s-process-goal-using-clause`(J, φ) in that execution of `s-process-goal`, where $\varphi = (A' \leftarrow B_1, \ldots, B_n)$ is a variant of φ'. We have that J contains $A = p(\bar{t})$. Let $\varphi' = \varphi\varrho$, where ϱ is a renaming substitution that uses only variables of φ and φ'. Thus,

$$(A'' \leftarrow B''_1, \ldots, B''_n) = (A' \leftarrow B_1, \ldots, B_n)\varrho. \tag{16}$$

Since θ_1 is an mgu for A and $A'' = A'\varrho$, we have that $A\theta_1 = A'\varrho\theta_1$. Since ϱ does not use variables of A, we have that $A\varrho = A$. Hence $A\varrho\theta_1 = A\theta_1 = A'\varrho\theta_1$. That is, $\varrho\theta_1$ is a unifier for A and A'. Let δ_0 be the substitution used in Step 3 of function `resolve-using-head-atom` for the considered atom A in the call `resolve-using-head-atom`(J, A') in Step 4 of the considered execution of `s-process-goal-using-clause`(J, φ). Since δ_0 is an mgu for A and A', it follows that $\varrho\theta_1 = \delta_0\gamma'_0$ for some substitution γ'_0.

As an inner induction, let the induction hypothesis be that, after processing the first $i - 1$ iterations of the "while" loop of the considered execution of `s-process-goal-using-clause`(J, φ), where $1 \leq i \leq n + 1$, it holds that

$$(\varrho\theta_1 \ldots \theta_{j_i-1})_{|X} = (\delta_{i-1}\gamma'_{i-1})_{|X} \tag{17}$$

for some $(A, \delta_{i-1}) \in K$ and some substitution γ'_{i-1}. This induction hypothesis holds for $i = 1$ because $j_1 = 2$, $\varrho\theta_1 = \delta_0\gamma'_0$ and the term-depth of $\varrho\theta_1$ is not greater than l. Suppose that the induction hypothesis holds for some $1 \leq i \leq n$. We show that it also holds for $i + 1$.

By (16) and the inductive assumption (17), we have that:

$$(\leftarrow B''_i\theta_1 \ldots \theta_{j_i-1}) = (\leftarrow B_i\varrho\theta_1 \ldots \theta_{j_i-1}) = (\leftarrow B_i\delta_{i-1}\gamma'_{i-1}). \tag{18}$$

Since the term-depth of $B_i\delta_{i-1}\gamma'_{i-1} = B''_i\theta_1 \dots \theta_{j_i-1}$ is not greater than l, the term-depth of $B_i\delta_{i-1}$ is also not greater than l. By (15), (18) and Lifting Lemma 2.2, we have that

$$\text{there exists a refutation of } P \cup I \cup \{\leftarrow\ B_i\delta_{i-1}\} \text{ using the leftmost selection function and mgu's } \theta'_{j_i}, \dots, \theta'_{j_{i+1}-1} \text{ such that the term-depths of the goals and the composition } \theta'_{j_i} \dots \theta'_{j_{i+1}-1} \text{ are not greater than } l \text{ and } \gamma'_{i-1}\theta_{j_i} \dots \theta_{j_{i+1}-1} = \theta'_{j_i} \dots \theta'_{j_{i+1}-1}\mu_i \text{ for some substitution } \mu_i. \tag{19}$$

Consider the case when the predicate p_i of B_i is an extensional predicate. Thus,

$$j_{i+1} = j_i + 1 \tag{20}$$

and

$$B_i\delta_{i-1}\theta'_{j_i} = B'_i\sigma\theta'_{j_i} \tag{21}$$

where $B'_i\sigma$ is the input program clause used for resolving $\leftarrow B_i\delta_{i-1}$, with $B'_i \in I(p_i)$ and σ being a renaming substitution. Let σ' be the renaming substitution used for making a variant of B'_i in Step 5 of function `resolve-using-body-atom` for the call `resolve-using-body-atom`$(K, B_i, I(p_i), Y)$ in Step 9 of the considered execution of `s-process-goal-using-clause`(J, φ). We have that $\sigma = \sigma'\sigma''$ for some renaming substitution σ'' which does not use variables of B_i, δ_{i-1} and X. Thus $B_i\delta_{i-1}\sigma''\theta'_{j_i} = B_i\delta_{i-1}\theta'_{j_i}$, and by using (21) and the fact that $\sigma = \sigma'\sigma''$, we have that

$$(B_i\delta_{i-1})\sigma''\theta'_{j_i} = B_i\delta_{i-1}\theta'_{j_i} = B'_i\sigma\theta'_{j_i} = (B'_i\sigma')\sigma''\theta'_{j_i}.$$

Hence, $B_i\delta_{i-1}$ and $B'_i\sigma'$ are unifiable using $\sigma''\theta'_{j_i}$, and γ_i is an mgu for them (Step 5 of the mentioned call of `resolve-using-body-atom`). Hence

$$\sigma''\theta'_{j_i} = \gamma_i\mu'_i \tag{22}$$

for some substitution μ'_i. Let

$$\gamma'_i = \mu'_i\mu_i. \tag{23}$$

We have that:

$$\begin{array}{lll}
 & (\varrho\theta_1 \dots \theta_{j_{i+1}-1})_{|X} & \\
= & ((\varrho\theta_1 \dots \theta_{j_i-1})_{|X}\theta_{j_i} \dots \theta_{j_{i+1}-1})_{|X} & \\
= & ((\delta_{i-1}\gamma'_{i-1})_{|X}\theta_{j_i} \dots \theta_{j_{i+1}-1})_{|X} & \text{(by the inner inductive assumption (17))} \\
= & (\delta_{i-1}\gamma'_{i-1}\theta_{j_i} \dots \theta_{j_{i+1}-1})_{|X} & \\
= & (\delta_{i-1}\theta'_{j_i} \dots \theta'_{j_{i+1}-1}\mu_i)_{|X} & \text{(by (19))} \\
= & (\delta_{i-1}\sigma''\theta'_{j_i} \dots \theta'_{j_{i+1}-1}\mu_i)_{|X} & \text{(since } \sigma'' \text{ does not use variables of } \delta_{i-1}, X) \\
= & (\delta_{i-1}\gamma_i\mu'_i\mu_i)_{|X} & \text{(by (20) and (22))} \\
= & (\delta_{i-1}\gamma_i\gamma'_i)_{|X} & \text{(by (23)).}
\end{array}$$

Since the term-depth of $\theta_1 \dots \theta_h$ is not greater than l and ϱ is a renaming substitution, the term-depth of $(\varrho\theta_1 \dots \theta_{j_{i+1}-1})_{|X}$ is not greater than l. It follows that the term-depth of $(\delta_{i-1}\gamma_i)_{|X}$ is also not greater than l. Hence, for $\delta_i = (\delta_{i-1}\gamma_i)_{|X}$, we have that $(\varrho\theta_1 \dots \theta_{j_{i+1}-1})_{|X} = (\delta_i\gamma'_i)_{|X}$ and (A, δ_i) was added to K' in Step 6 of the mentioned call of `resolve-using-body-atom`, which is then passed to K in Step 12 of the considered execution of `s-process-goal-using-clause`(J, φ). That is, the induction hypothesis of the inner induction holds for $i + 1$.

Now consider the case when the predicate p_i of B_i is an intensional predicate.

We first show that ans_p_i contains an atom B'_i such that $B_i\delta_{i-1}\theta'_{j_i} \dots \theta'_{j_{i+1}-1}$ is an instance of a variant of B'_i. As procedure `s-process-goal` was called for

a goal relation containing $B_i\delta_{i-1}$ (in Step 11 of the considered execution of s-process-goal-using-clause(J, φ)), $input\_p_i$ must contain an atom $B_i^\diamond$ such that $B_i\delta_{i-1}$ is an instance of a variant of $B_i^\diamond$. Let α be a substitution such that

$$B_i\delta_{i-1} = B_i^\diamond\alpha \tag{24}$$

and α uses only variables from $B_i\delta_{i-1}$ and $B_i^\diamond$. By (19) and Lifting Lemma 2.2, it follows that there exists a refutation of $P \cup I \cup \{\leftarrow B_i^\diamond\}$ using the leftmost selection function and mgu's $\theta''_{j_i}, \ldots, \theta''_{j_{i+1}-1}$ such that the term-depths of the goals and the composition $\theta''_{j_i} \ldots \theta''_{j_{i+1}-1}$ are not greater than l and

$$\alpha\theta'_{j_i} \ldots \theta'_{j_{i+1}-1} = \theta''_{j_i} \ldots \theta''_{j_{i+1}-1}\beta \tag{25}$$

for some substitution β. By the outer inductive assumption, ans_p_i contains an atom B'_i such that $B_i^\diamond\theta''_{j_i} \ldots \theta''_{j_{i+1}-1}$ is an instance of a variant of B'_i. Since

$$\begin{aligned} B_i\delta_{i-1}\theta'_{j_i} \ldots \theta'_{j_{i+1}-1} &= B_i^\diamond\alpha\theta'_{j_i} \ldots \theta'_{j_{i+1}-1} \quad \text{(by (24))} \\ &= B_i^\diamond\theta''_{j_i} \ldots \theta''_{j_{i+1}-1}\beta \quad \text{(by (25)),} \end{aligned}$$

it follows that

$$B_i\delta_{i-1}\theta'_{j_i} \ldots \theta'_{j_{i+1}-1} \text{ is also an instance of a variant of } B'_i. \tag{26}$$

Let σ be the renaming substitution used for making a variant $B'_i\sigma$ of B'_i in Step 5 of function resolve-using-body-atom for the call resolve-using-body-atom(K, B_i, ans_p_i, Y) in Step 12 of the considered execution of s-process-goal-using-clause(J, φ). The atom $B'_i\sigma$ does not contain variables of X, δ_{i-1} and $\theta'_{j_i} \ldots \theta'_{j_{i+1}-1}$. Hence, by (26), $B_i\delta_{i-1}\theta'_{j_i} \ldots \theta'_{j_{i+1}-1}$ is an instance of $B'_i\sigma$. Let ρ be a substitution with domain contained in $Var(B'_i\sigma)$ such that $B_i\delta_{i-1}\theta'_{j_i} \ldots \theta'_{j_{i+1}-1} = B'_i\sigma\rho$. We have that $\theta'_{j_i} \ldots \theta'_{j_{i+1}-1} \cup \rho$ is a unifier for $B_i\delta_{i-1}$ and $B'_i\sigma$. As γ_i is an mgu for $B_i\delta_{i-1}$ and $B'_i\sigma$, we have that $\gamma_i\mu'_i = (\theta'_{j_i} \ldots \theta'_{j_{i+1}-1} \cup \rho)$ for some substitution μ'_i. Hence

$$(\gamma_i\mu'_i)_{|X \cup Var(\delta_{i-1})} = (\theta'_{j_i} \ldots \theta'_{j_{i+1}-1})_{|X \cup Var(\delta_{i-1})}. \tag{27}$$

Let

$$\gamma'_i = \mu'_i\mu_i. \tag{28}$$

We have that:

$$\begin{aligned} & (\varrho\theta_1 \ldots \theta_{j_{i+1}-1})_{|X} \\ = \ & (\delta_{i-1}\theta'_{j_i} \ldots \theta'_{j_{i+1}-1}\mu_i)_{|X} \text{ (as for the case when } p_i \text{ is an extensional predicate)} \\ = \ & (\delta_{i-1}(\theta'_{j_i} \ldots \theta'_{j_{i+1}-1})_{|X \cup Var(\delta_{i-1})}\mu_i)_{|X} \\ = \ & (\delta_{i-1}(\gamma_i\mu'_i)_{|X \cup Var(\delta_{i-1})}\mu_i)_{|X} \text{ (by (27))} \\ = \ & (\delta_{i-1}\gamma_i\mu'_i\mu_i)_{|X} \\ = \ & (\delta_{i-1}\gamma_i\gamma'_i)_{|X} \text{ (by (28)).} \end{aligned}$$

Analogously as for the case when p_i is an extensional predicate, the term-depth of $(\delta_{i-1}\gamma_i)_{|X}$ is not greater than l, and for $\delta_i = (\delta_{i-1}\gamma_i)_{|X}$, the induction hypothesis of the inner induction holds for $i + 1$.

We have proved the induction hypothesis of the inner induction, which implies that $(\varrho\theta_1 \ldots \theta_{j_{n+1}-1})_{|X} = (\delta_n\gamma'_n)_{|X}$. That is, $(\varrho\theta_1 \ldots \theta_h)_{|X} = (\delta_n\gamma'_n)_{|X}$. Hence $A\delta_n\gamma'_n = A\varrho\theta_1 \ldots \theta_h = A\theta_1 \ldots \theta_h$ (since $A\varrho = A$). Hence $p(\bar{t})\theta_1 \ldots \theta_h = A\theta_1 \ldots \theta_h$ is an instance of $A\delta_n$. By Step 13 of the considered execution of s-process-goal-using-clause(J, φ), it follows that $p(\bar{t})\theta_1 \ldots \theta_h$ is an instance of a variant of some atom from ans_p. □

THEOREM A.3 (COMPLETENESS). *After a run of Algorithm 2 (using parameter l) on a query $(P, q(\overline{x}))$ and an extensional instance I, for every SLD-refutation of $P \cup I \cup \{\leftarrow q(\overline{x})\}$ that uses the leftmost selection function and does not contain any goal with term-depth greater than l, if $\theta_1, \ldots, \theta_h$ are the mgu's used in the refutation and the term-depth of the composition $\theta_1 \ldots \theta_h$ is not greater than l then there exists a tuple $\overline{t} \in ans\_q$ such that $\overline{x}\theta_1 \ldots \theta_h$ is an instance of a variant of $\overline{t}$.*

This theorem immediately follows from Lemma A.2. Together with Theorem 2.1 (on completeness of SLD-resolution) it makes a relationship between correct answers of $P \cup I \cup \{\leftarrow q(\overline{x})\}$ and the answers computed by Algorithm 2 for the query $(P, q(\overline{x}))$ on the extensional instance I.

For queries and extensional instances without function symbols, we take term-depth bound $l = 0$ and obtain the following completeness result, which immediately follows from the above theorem.

COROLLARY A.4. *After a run of Algorithm 2 using $l = 0$ on a query $(P, q(\overline{x}))$ and an extensional instance I that do not contain function symbols, for every computed answer θ of an SLD-refutation of $P \cup I \cup \{\leftarrow q(\overline{x})\}$ that uses the leftmost selection function, there exists a tuple $\overline{t} \in ans\_q$ such that $\overline{x}\theta$ is an instance of a variant of $\overline{t}$.*

ELECTRONIC APPENDIX

The electronic appendix for this article can be accessed in the ACM Digital Library.

ACKNOWLEDGMENTS

We would like to thank Mohamed Yahya for pointing out a mistake in the conference version [Madalińska-Bugaj and Nguyen 2008] of this paper. We are grateful to the reviewers for very helpful comments and suggestions.

REFERENCES

ABITEBOUL, S., HULL, R., AND VIANU, V. 1995. *Foundations of Databases*. Addison Wesley.

APT, K. 1997. *From Logic Programming to Prolog*. Prentice-Hall.

BANCILHON, F., MAIER, D., SAGIV, Y., AND ULLMAN, J. 1986. Magic sets and other strange ways to implement logic programs. In *Proceedings of PODS'1986*. ACM, 1–15.

BRODT, S., BRY, F., AND EISINGER, N. 2009. Search for more declarativity. In *Proceedings of Web Reasoning and Rule Systems 2009*, A. Polleres and T. Swift, Eds. LNCS Series, vol. 5837. Springer, 71–86.

BRY, F. 1990. Query evaluation in deductive databases: Bottom-up and top-down reconciled. *Data Knowl. Eng. 5*, 289–312.

CALÌ, A., GOTTLOB, G., AND LUKASIEWICZ, T. 2009. Datalog$^{\pm}$: a unified approach to ontologies and integrity constraints. In *Proceedings of ICDT 2009*, R. Fagin, Ed. ACM International Conference Proceeding Series Series, vol. 361. ACM, 14–30.

CAO, S., NGUYEN, L., AND SZALAS, A. 2011a. On the web ontology rule language OWL 2 RL. In *Proceedings of ICCCI'2011*, P. Jedrzejowicz, N.-T. Nguyen, and K. Hoang, Eds. LNCS Series, vol. 6922. Springer, 254–264.

CAO, S., NGUYEN, L., AND SZALAS, A. 2011b. WORL: A Web ontology rule language. In *Proceedings of KSE'2011*. IEEE, 32–39.

CLARK, K. 1979. Predicate logic as a computational formalism. Research Report DOC 79/59, Department of Computing, Imperial College.

FREIRE, J., SWIFT, T., AND WARREN, D. 1997. Taking I/O seriously: Resolution reconsidered for disk. In *Proc. of ICLP'1997*, L. Naish, Ed. MIT Press, 198–212.

LLOYD, J. 1987. *Foundations of Logic Programming, 2nd Edition*. Springer.

MADALIŃSKA-BUGAJ, E. AND NGUYEN, L. 2008. Generalizing the QSQR evaluation method for Horn knowledge bases. In *New Challenges in Applied Intelligence Technologies*, N.-T. Nguyen and R. Katarzyniak, Eds. Studies in Computational Intelligence Series, vol. 134. Springer, 145–154.

NEJDL, W. 1987. Recursive strategies for answering recursive queries - the RQA/FQI strategy. In *Proceedings of VLDB'87*, P. Stocker, W. Kent, and P. Hammersley, Eds. Morgan Kaufmann, 43–50.

NGUYEN, L. 2007. Foundations of modal deductive databases. *Fundamenta Informaticae 79,* 1-2, 85–135.

NGUYEN, L. 2011. An implementation in Prolog of the generalized QSQR evaluation method for Horn knowledge bases. `http://www.mimuw.edu.pl/~nguyen/GQSQR-PL.zip`.

RAMAKRISHNAN, R., SRIVASTAVA, D., AND SUDARSHAN, S. 1992. Efficient bottom-up evaluation of logic programs. In J. Vandewalle, editor, The State of the Art in Computer Systems and Software Engineering. Kluwer Academic Publishers.

ROHMER, J., LESCOUER, R., AND KERISIT, J.-M. 1986. The Alexander method – a technique for the processing of recursive axioms in deductive databases. *New Generation Computing 4,* 3, 273–285.

ROSS, K. 1996. Tail recursion elimination in deductive databases. *ACM Trans. Database Syst. 21,* 2, 208–237.

SAGONAS, K. AND SWIFT, T. 1998. An abstract machine for tabled execution of fixed-order stratified logic programs. *ACM Trans. Program. Lang. Syst. 20,* 3, 586–634.

SAGONAS, K., SWIFT, T., AND WARREN, D. 1994. XSB as an efficient deductive database engine. In *Proceedings of the 1994 ACM SIGMOD Conference on Management of Data*, R. Snodgrass and M. Winslett, Eds. ACM Press, 442–453.

SHEN, Y.-D., YUAN, L.-Y., YOU, J.-H., AND ZHOU, N.-F. 2001. Linear tabulated resolution based on Prolog control strategy. *TPLP 1,* 1, 71–103.

STAAB, S. 2008. Completeness of the SLD-resolution. Slides of a course on Advanced Data Modeling, `http://www.uni-koblenz.de/FB4/Institutes/IFI/AGStaab/Teaching/SS08/adm08/DB2-SS08-Slides9.ppt`.

STÄRK, R. 1990. A direct proof for the completeness of SLD-resolution. In *Proceedings of CSL'89*, E. Börger, H. Büning, and M. Richter, Eds. LNCS Series, vol. 440. Springer, 382–383.

TAMAKI, H. AND SATO, T. 1986. OLD resolution with tabulation. In *Proceedings of ICLP'1986, LNCS 225*, E. Shapiro, Ed. Springer, 84–98.

VIEILLE, L. 1986. Recursive axioms in deductive databases: The query/subquery approach. In *Proceedings of Expert Database Conf.* 253–267.

VIEILLE, L. 1987. A database-complete proof procedure based on SLD-resolution. In *Proceedings of ICLP*. 74–103.

VIEILLE, L. 1989. Recursive query processing: The power of logic. *Theor. Comput. Sci. 69,* 1, 1–53.

ZHOU, N.-F. AND SATO, T. 2003. Efficient fixpoint computation in linear tabling. In *Proceedings of PPDP'2003*. ACM, 275–283.

Received May 2010; revised September 2011; accepted December 2011

Online Appendix to: A Generalized QSQR Evaluation Method for Horn Knowledge Bases

EWA MADALIŃSKA-BUGAJ, University of Warsaw
LINH ANH NGUYEN, University of Warsaw

A. ILLUSTRATIVE EXAMPLES

Example A.1. This example illustrates Algorithm 1. Consider the following query [Nejdl 1987], in which x, y, z, w denote variables, and a, b, c, d, e, i, o denote constant symbols:

— program P :

$$n(x,y) \leftarrow r(x,y)$$
$$n(x,y) \leftarrow p(x,z), n(z,w), q(w,y)$$
$$s(x) \leftarrow n(c,x)$$

— extensional instance I :

$p(c,d)$	$r(d,e)$	$q(e,a)$
$p(b,c)$		$q(a,i)$
$p(c,b)$		$q(i,o)$

— query: $s(x)$.

We give below a trace of a run of Algorithm 1 on this query (using term-depth bound $l = 0$). In this trace variables of the form δ or sup are local variables of procedure `process-goal-using-clause`. Figures 5 and 6 contain a proof-tree based presentation of this trace.

(1) $ans\_$ variables are set to empty relations
(2) $input\_$ variables are set to empty relations
(3) **calling** `process-goal` ($s(x)$)
(4) $input\_s := \{x\}$
(5) **calling** `process-goal-using-clause` ($s(x)$, ($s(x_1) \leftarrow n(c,x_1)$))
(6) $\delta_0 := \{x_1/x\}$, $sup_0 := \{\delta_0\}$
(7) $sup_1 := \emptyset$
(8) **calling** `process-goal` ($n(c,x)$)
(9) $input\_n := \{(c,x)\}$
(10) **calling** `process-goal-using-clause` ($n(c,x)$, ($n(x_2,y_2) \leftarrow r(x_2,y_2)$))
(11) $\delta_0 := \{x_2/c, y_2/x\}$, $sup_0 := \{\delta_0\}$
(12) $sup_1 := \emptyset$
(13) **exiting** the call 10
(14) **calling** `process-goal-using-clause` ($n(c,x)$,
($n(x_3,y_3) \leftarrow p(x_3,z_3), n(z_3,w_3), q(w_3,y_3)$))
(15) $\delta_0 := \{x_3/c, y_3/x\}$, $sup_0 := \{\delta_0\}$
(16) $sup_1 := \emptyset$
(17) $sup_1 := \{\{x_3/c, y_3/x, z_3/d\}, \{x_3/c, y_3/x, z_3/b\}\}$
(18) $sup_2 := \emptyset$
(19) $\delta_1 := \{x_3/c, y_3/x, z_3/d\}$

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

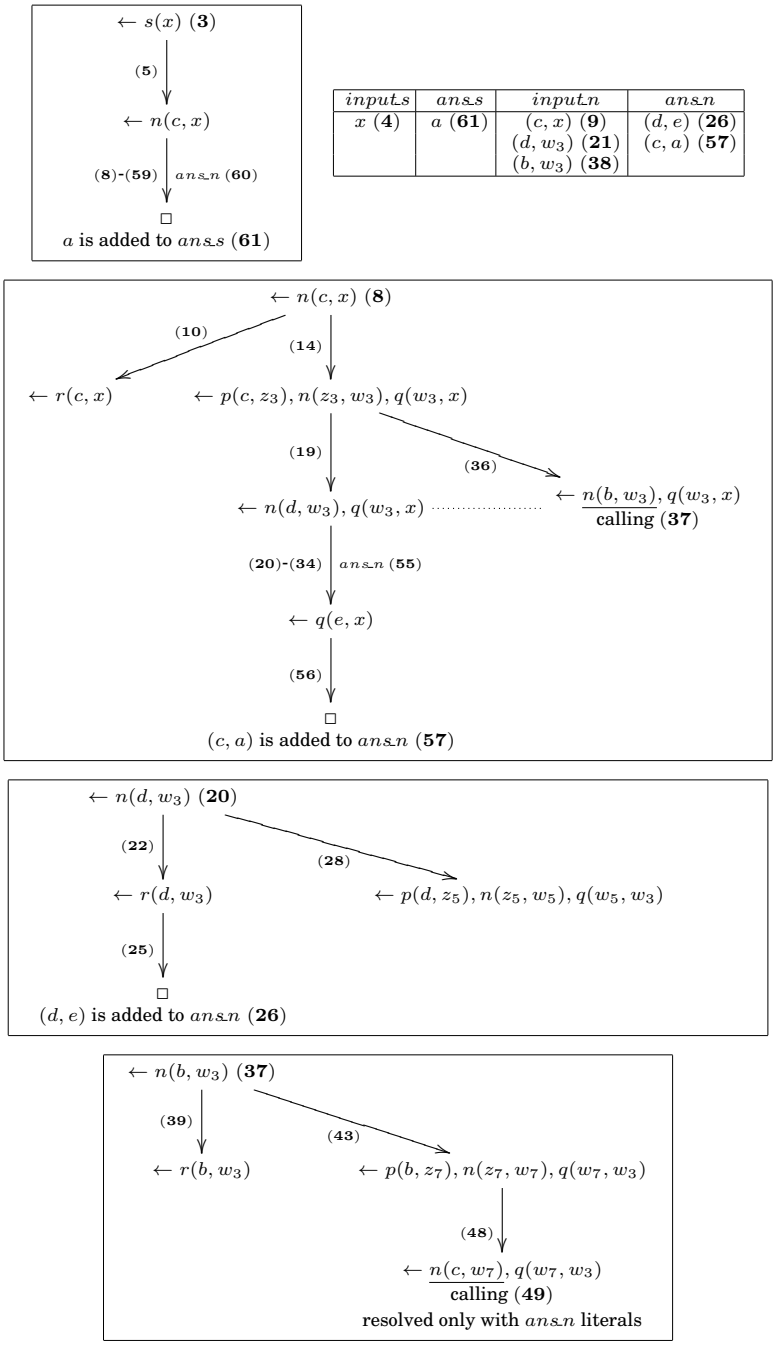
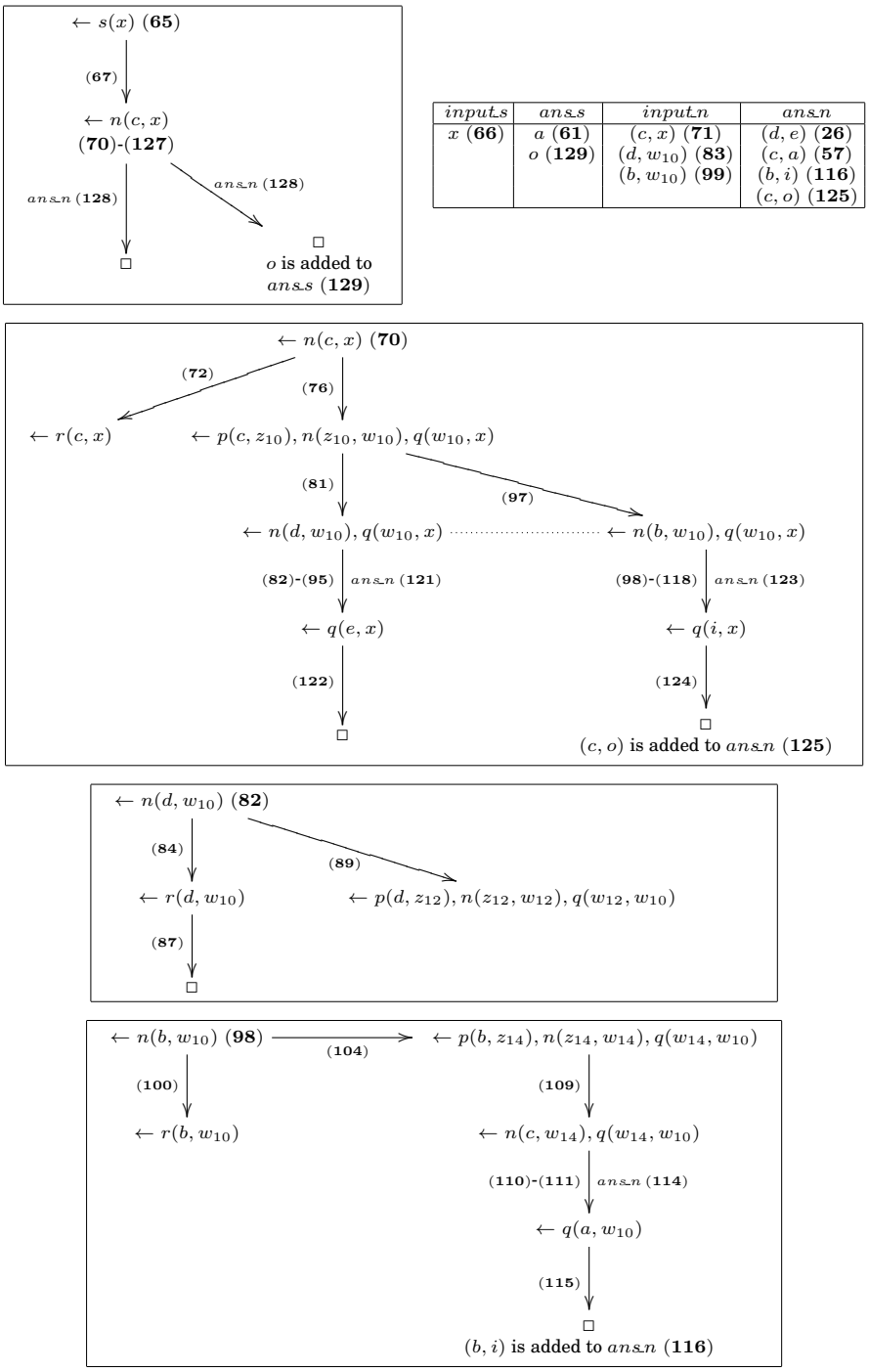


Fig. 5. A proof-tree based presentation of a trace of the first iteration of the outer loop of a run of Algorithm 1 on the query given in Example A.1. The numbers in bold font indicate the corresponding steps of that trace, which are listed in the example.



input_s	*ans_s*	*input_n*	*ans_n*
x (**66**)	a (**61**)	(c, x) (**71**)	(d, e) (**26**)
	o (**129**)	(d, w_{10}) (**83**)	(c, a) (**57**)
		(b, w_{10}) (**99**)	(b, i) (**116**)
			(c, o) (**125**)

Fig. 6. A proof-tree based presentation of a trace of the second iteration of the outer loop of a run of Algorithm 1 on the query given in Example A.1 (after clearing *input_* variables). The numbers in bold font indicate the corresponding steps of that trace, which are listed in the example.

(20) **calling** `process-goal` ($n(d, w_3)$)
(21) $input\_n := \{(c, x), (d, w_3)\}$
(22) **calling** `process-goal-using-clause` ($n(d, w_3)$, ($n(x_4, y_4) \leftarrow r(x_4, y_4)$))
(23) $\delta_0 := \{x_4/d, y_4/w_3\}$, $sup_0 := \{\delta_0\}$
(24) $sup_1 := \emptyset$
(25) $sup_1 := \{\{x_4/d, y_4/e, w_3/e\}\}$
(26) $ans\_n := \{(d, e)\}$
(27) **exiting** the call 22
(28) **calling** `process-goal-using-clause` ($n(d, w_3)$, ($n(x_5, y_5) \leftarrow p(x_5, z_5), n(z_5, w_5), q(w_5, y_5)$))
(29) $\delta_0 := \{x_5/d, y_5/w_3\}$, $sup_0 := \{\delta_0\}$
(30) $sup_1 := \emptyset$
(31) $sup_2 := \emptyset$
(32) $sup_3 := \emptyset$
(33) **exiting** the call 28
(34) **exiting** the call 20
(35) $sup_2 := \{\{x_3/c, y_3/x, z_3/d, w_3/e\}\}$
(36) $\delta_1 := \{x_3/c, y_3/x, z_3/b\}$
(37) **calling** `process-goal` ($n(b, w_3)$)
(38) $input\_n := \{(c, x), (d, w_3), (b, w_3)\}$
(39) **calling** `process-goal-using-clause` ($n(b, w_3)$, ($n(x_6, y_6) \leftarrow r(x_6, y_6)$))
(40) $\delta_0 := \{x_6/b, y_6/w_3\}$, $sup_0 := \{\delta_0\}$
(41) $sup_1 := \emptyset$
(42) **exiting** the call 39
(43) **calling** `process-goal-using-clause` ($n(b, w_3)$, ($n(x_7, y_7) \leftarrow p(x_7, z_7), n(z_7, w_7), q(w_7, y_7)$))
(44) $\delta_0 := \{x_7/b, y_7/w_3\}$, $sup_0 := \{\delta_0\}$
(45) $sup_1 := \emptyset$
(46) $sup_1 := \{\{x_7/b, y_7/w_3, z_7/c\}\}$
(47) $sup_2 := \emptyset$
(48) $\delta_1 := \{x_7/b, y_7/w_3, z_7/c\}$
(49) **calling** `process-goal` ($n(c, w_7)$)
(50) **exiting** the call 49
(51) $sup_3 := \emptyset$
(52) **exiting** the call 43
(53) **exiting** the call 37
(54) $sup_3 := 0$
(55) $\delta_2 := \{x_3/c, y_3/x, z_3/d, w_3/e\}$
(56) $sup_3 := \{\{x_3/c, y_3/x, z_3/d, w_3/e, x/a\}\}$
(57) $ans\_n := \{(d, e), (c, a)\}$
(58) **exiting** the call 14
(59) **exiting** the call 8
(60) $sup_1 := \{\{x_1/a, x/a\}\}$
(61) $ans\_s := \{a\}$
(62) **exiting** the call 5
(63) **exiting** the call 3

(64) $input\_$ variables are reset to empty relations

(65) **calling** `process-goal` ($s(x)$)
(66) $input\_s := \{x\}$
(67) **calling** `process-goal-using-clause` ($s(x)$, ($s(x_8) \leftarrow n(c, x_8)$))
(68) $\delta_0 := \{x_8/x\}$, $sup_0 := \{\delta_0\}$

(69) $sup_1 := \emptyset$
(70) **calling** `process-goal` $(n(c, x))$
(71) $input\_n := \{(c, x)\}$
(72) **calling** `process-goal-using-clause` $(n(c, x), (n(x_9, y_9) \leftarrow r(x_9, y_9)))$
(73) $\delta_0 := \{x_9/c, y_9/x\}$, $sup_0 := \{\delta_0\}$
(74) $sup_1 := \emptyset$
(75) **exiting** the call 72
(76) **calling** `process-goal-using-clause` $(n(c, x),$
$(n(x_{10}, y_{10}) \leftarrow p(x_{10}, z_{10}), n(z_{10}, w_{10}), q(w_{10}, y_{10})))$
(77) $\delta_0 := \{x_{10}/c, y_{10}/x\}$, $sup_0 := \{\delta_0\}$
(78) $sup_1 := \emptyset$
(79) $sup_1 := \{\{x_{10}/c, y_{10}/x, z_{10}/d\}, \{x_{10}/c, y_{10}/x, z_{10}/b\}\}$
(80) $sup_2 := \emptyset$
(81) $\delta_1 := \{x_{10}/c, y_{10}/x, z_{10}/d\}$
(82) **calling** `process-goal` $(n(d, w_{10}))$
(83) $input\_n := \{(c, x), (d, w_{10})\}$
(84) **calling** `process-goal-using-clause` $(n(d, w_{10}), (n(x_{11}, y_{11}) \leftarrow r(x_{11}, y_{11})))$
(85) $\delta_0 := \{x_{11}/d, y_{11}/w_{10}\}$, $sup_0 := \{\delta_0\}$
(86) $sup_1 := \emptyset$
(87) $sup_1 := \{\{x_{11}/d, y_{11}/e, w_{10}/e\}\}$
(88) **exiting** the call 84
(89) **calling** `process-goal-using-clause` $(n(d, w_{10}),$
$(n(x_{12}, y_{12}) \leftarrow p(x_{12}, z_{12}), n(z_{12}, w_{12}), q(w_{12}, y_{12})))$
(90) $\delta_0 := \{x_{12}/d, y_{12}/w_{10}\}$, $sup_0 := \{\delta_0\}$
(91) $sup_1 := \emptyset$
(92) $sup_2 := \emptyset$
(93) $sup_3 := \emptyset$
(94) **exiting** the call 89
(95) **exiting** the call 82
(96) $sup_2 := \{\{x_{10}/c, y_{10}/x, z_{10}/d, w_{10}/e\}\}$
(97) $\delta_1 := \{x_{10}/c, y_{10}/x, z_{10}/b\}$
(98) **calling** `process-goal` $(n(b, w_{10}))$
(99) $input\_n := \{(c, x), (d, w_{10}), (b, w_{10})\}$
(100) **calling** `process-goal-using-clause` $(\leftarrow n(b, w_{10}), (n(x_{13}, y_{13}) \leftarrow r(x_{13}, y_{13})))$
(101) $\delta_0 := \{x_{13}/b, y_{13}/w_{10}\}$, $sup_0 := \{\delta_0\}$
(102) $sup_1 := \emptyset$
(103) **exiting** the call 100
(104) **calling** `process-goal-using-clause` $(n(b, w_{10}),$
$(n(x_{14}, y_{14}) \leftarrow p(x_{14}, z_{14}), n(z_{14}, w_{14}), q(w_{14}, y_{14})))$
(105) $\delta_0 := \{x_{14}/b, y_{14}/w_{10}\}$, $sup_0 := \{\delta_0\}$
(106) $sup_1 := \emptyset$
(107) $sup_1 := \{\{x_{14}/b, y_{14}/w_{10}, z_{14}/c\}\}$
(108) $sup_2 := \emptyset$
(109) $\delta_1 := \{x_{14}/b, y_{14}/w_{10}, z_{14}/c\}$
(110) **calling** `process-goal` $(n(c, w_{14}))$
(111) **exiting** the call 110
(112) $sup_2 := \{\{x_{14}/b, y_{14}/w_{10}, z_{14}/c, w_{14}/a\}\}$
(113) $sup_3 := \emptyset$
(114) $\delta_2 := \{x_{14}/b, y_{14}/w_{10}, z_{14}/c, w_{14}/a\}$
(115) $sup_3 := \{\{x_{14}/b, y_{14}/i, z_{14}/c, w_{14}/a, w_{10}/i\}\}$
(116) $ans\_n := \{(d, e), (c, a), (b, i)\}$
(117) **exiting** the call 104

(118) **exiting** the call 98
(119) $sup_2 := \{\{x_{10}/c, y_{10}/x, z_{10}/d, w_{10}/e\}, \{x_{10}/c, y_{10}/x, z_{10}/b, w_{10}/i\}\}$
(120) $sup_3 := \emptyset$
(121) $\delta_2 := \{x_{10}/c, y_{10}/x, z_{10}/d, w_{10}/e\}$
(122) $sup_3 := \{\{x_{10}/c, y_{10}/a, z_{10}/d, w_{10}/e, x/a\}\}$
(123) $\delta_2 := \{x_{10}/c, y_{10}/x, z_{10}/b, w_{10}/i\}$
(124) $sup_3 := \{\{x_{10}/c, y_{10}/a, z_{10}/d, w_{10}/e, x/a\}, \{x_{10}/c, y_{10}/o, z_{10}/b, w_{10}/i, x/o\}\}$
(125) $ans\_n := \{(d, e), (c, a), (b, i), (c, o)\}$
(126) **exiting** the call 76
(127) **exiting** the call 70
(128) $sup_1 := \{\{x_8/a, x/a\}, \{x_8/o, x/o\}\}$
(129) $ans\_s := \{a, o\}$
(130) **exiting** the call 67
(131) **exiting** the call 65
(132) the main loop is repeated once more, without affecting $ans\_$ relations
(133) the returned result is $\{a, o\}$.

Example A.2. We illustrate Algorithm 2 by tracing it on the query given in Example A.1 (using term-depth bound $l = 0$). Here, we assume that the instruction $K' := K' \cup \{(A, (\delta_{i-1}\gamma_i)_{|X})\}$ in Step 6 of function `resolve-using-body-atom` is optimized to $K' := K' \cup \{(A, (\delta_{i-1}\gamma_i)_{|Var((A,B_{i+1},\ldots,B_n))})\}$, where $B_{i+1}, \ldots, B_n$ are the remaining atoms of the body of the processed program clause. In the presented trace variables K and i are local variables of procedure `s-process-goal-using-clause`. Figures 7 and 8 contain a proof-tree based presentation of this trace.

(1) $ans\_$ variables are set to empty relations
(2) $input\_$ variables are set to empty relations
(3) **calling** `s-process-goal` $(\{s(x)\})$
(4) $input\_s := \{x\}$
(5) **calling** `s-process-goal-using-clause` $(\{s(x)\}, (s(x_1) \leftarrow n(c, x_1)))$
(6) $K := \{(s(x), \{x_1/x\})\}$
(7) $i := 1$
(8) **calling** `s-process-goal` $(\{n(c, x)\})$
(9) $input\_n := \{(c, x)\}$
(10) **calling** `s-process-goal-using-clause` $(\{n(c, x)\}, (n(x_2, y_2) \leftarrow r(x_2, y_2)))$
(11) $K := \{(n(c, x), \{x_2/c, y_2/x\})\}$
(12) $i := 1$
(13) $K := \emptyset$
(14) **exiting** the call 10
(15) **calling** `s-process-goal-using-clause` $(\{n(c, x)\},$
$(n(x_3, y_3) \leftarrow p(x_3, z_3), n(z_3, w_3), q(w_3, y_3)))$
(16) $K := \{(n(c, x), \{x_3/c, y_3/x\})\}$
(17) $i := 1$
(18) $K := \{(n(c, x), \{y_3/x, z_3/d\}), (n(c, x), \{y_3/x, z_3/b\})\}$
(19) $i := 2$
(20) **calling** `s-process-goal` $(\{n(d, w_3), n(b, w_3)\})$
(21) $input\_n := \{(c, x), (d, w_3), (b, w_3)\}$
(22) **calling** `s-process-goal-using-clause` $(\{n(d, w_3), n(b, w_3)\},$
$(n(x_4, y_4) \leftarrow r(x_4, y_4)))$
(23) $K := \{(n(d, w_3), \{x_4/d, y_4/w_3\}), (n(b, w_3), \{x_4/b, y_4/w_3\})\}$
(24) $i := 1$

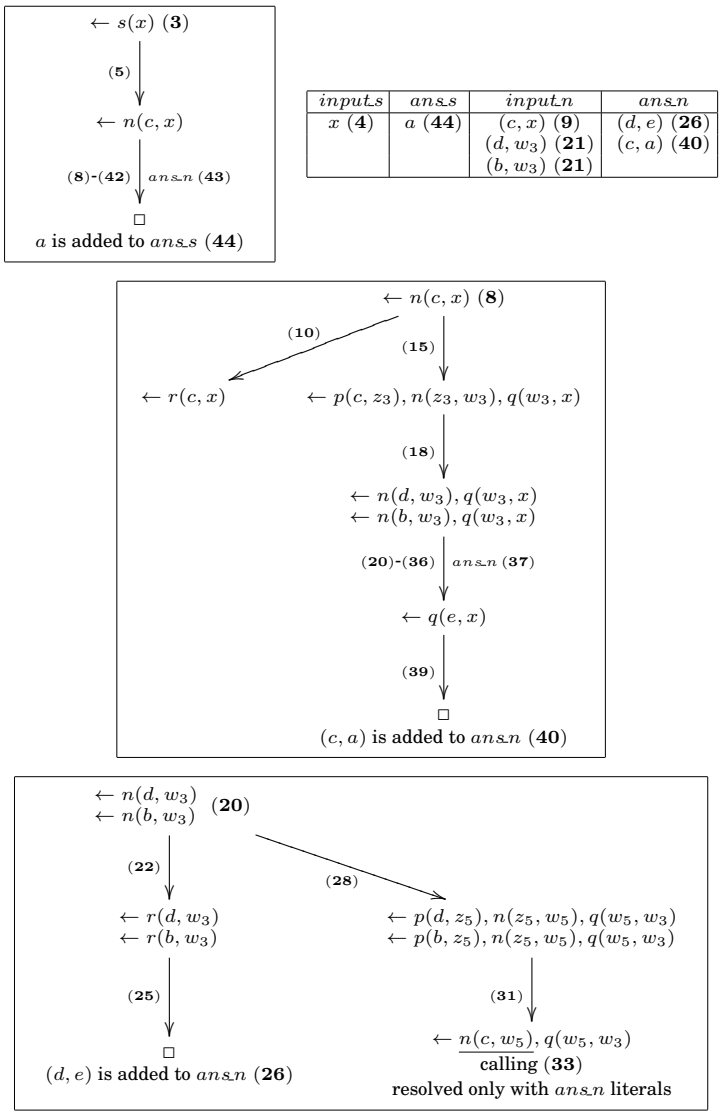


Fig. 7. A proof-tree based presentation of a trace of the first iteration of the outer loop of a run of Algorithm 2 on the query given in Example A.2. The numbers in bold font indicate the corresponding steps of that trace, which are listed in the example.

(25) $K := \{(n(d, w_3), \{w_3/e\})\}$
(26) $ans\_n := \{(d, e)\}$
(27) **exiting** the call 22
(28) **calling** `s-process-goal-using-clause` $(\{n(d, w_3), n(b, w_3)\},$
$(n(x_5, y_5) \leftarrow p(x_5, z_5), n(z_5, w_5), q(w_5, y_5)))$
(29) $K := \{(n(d, w_3), \{x_5/d, y_5/w_3\}), (n(b, w_3), \{x_5/b, y_5/w_3\})\}$
(30) $i := 1$
(31) $K := \{(n(b, w_3), \{y_5/w_3, z_5/c\})\}$
(32) $i := 2$

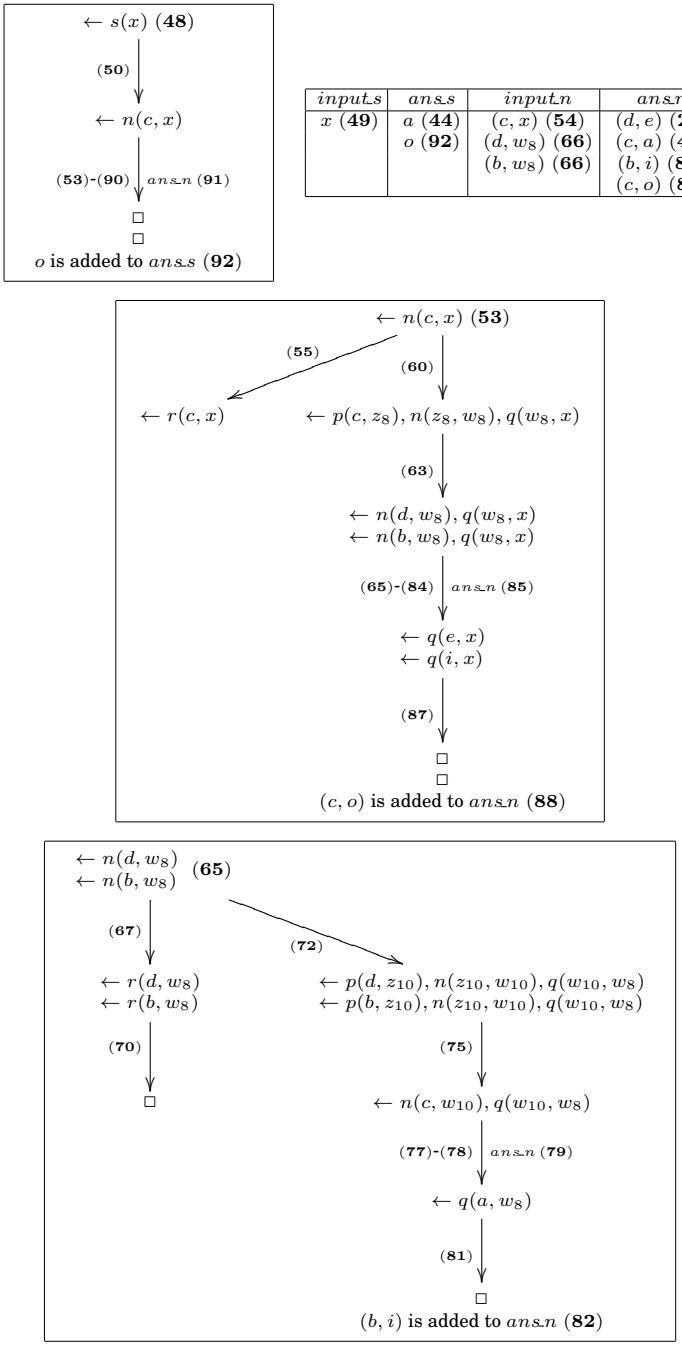


Fig. 8. A proof-tree based presentation of a trace of the second iteration of the outer loop of a run of Algorithm 2 on the query given in Example A.2 (after clearing $input\_$ variables). The numbers in bold font indicate the corresponding steps of that trace, which are listed in the example.

(33) **calling** `s-process-goal` ($\{n(c, w_5)\}$)
(34) **exiting** the call 33
(35) **exiting** the call 28
(36) **exiting** the call 20
(37) $K := \{(n(c, x), \{y_3/x, w_3/e\})\}$
(38) $i := 3$
(39) $K := \{(n(c, x), \{x/a\})\}$
(40) $ans\_n := \{(d, e), (c, a)\}$
(41) **exiting** the call 15
(42) **exiting** the call 8
(43) $K := \{(s(x), \{x/a\})\}$
(44) $ans\_s := \{a\}$
(45) **exiting** the call 5
(46) **exiting** the call 3

(47) $input\_$ variables are reset to empty relations

(48) **calling** `s-process-goal` ($\{s(x)\}$)
(49) $input\_s := \{x\}$
(50) **calling** `s-process-goal-using-clause` ($\{s(x)\}$, $(s(x_6) \leftarrow n(c, x_6))$)
(51) $K := \{(s(x), \{x_6/x\})\}$
(52) $i := 1$
(53) **calling** `s-process-goal` ($\{n(c, x)\}$)
(54) $input\_n := \{(c, x)\}$
(55) **calling** `s-process-goal-using-clause` ($\{n(c, x)\}$, $(n(x_7, y_7) \leftarrow r(x_7, y_7))$)
(56) $K := \{(n(c, x), \{x_7/c, y_7/x\})\}$
(57) $i := 1$
(58) $K := \emptyset$
(59) **exiting** the call 55
(60) **calling** `s-process-goal-using-clause` ($\{n(c, x)\}$,
$(n(x_8, y_8) \leftarrow p(x_8, z_8), n(z_8, w_8), q(w_8, y_8))$)
(61) $K := \{(n(c, x), \{x_8/c, y_8/x\})\}$
(62) $i := 1$
(63) $K := \{(n(c, x), \{y_8/x, z_8/d\}), (n(c, x), \{y_8/x, z_8/b\})\}$
(64) $i := 2$
(65) **calling** `s-process-goal` ($\{n(d, w_8), n(b, w_8)\}$)
(66) $input\_n := \{(c, x), (d, w_8), (b, w_8)\}$
(67) **calling** `s-process-goal-using-clause` ($\{n(d, w_8), n(b, w_8)\}$,
$(n(x_9, y_9) \leftarrow r(x_9, y_9))$)
(68) $K := \{(n(d, w_8), \{x_9/d, y_9/w_8\}), (n(b, w_8), \{x_9/b, y_9/w_8\})\}$
(69) $i := 1$
(70) $K := \{(n(d, w_8), \{w_8/e\})\}$
(71) **exiting** the call 67
(72) **calling** `s-process-goal-using-clause` ($\{n(d, w_8), n(b, w_8)\}$,
$(n(x_{10}, y_{10}) \leftarrow p(x_{10}, z_{10}), n(z_{10}, w_{10}), q(w_{10}, y_{10}))$)
(73) $K := \{(n(d, w_8), \{x_{10}/d, y_{10}/w_8\}), (n(b, w_8), \{x_{10}/b, y_{10}/w_8\})\}$
(74) $i := 1$
(75) $K := \{(n(b, w_8), \{y_{10}/w_8, z_{10}/c\})\}$
(76) $i := 2$
(77) **calling** `s-process-goal` ($\{n(c, w_{10})\}$)
(78) **exiting** the call 77
(79) $K := \{(n(b, w_8), \{y_{10}/w_8, w_{10}/a\})\}$
(80) $i := 3$

(81) $K := \{(n(b, w_8), \{w_8/i\})\}$
(82) $ans\_n := \{(d, e), (c, a), (b, i)\}$
(83) **exiting** the call 72
(84) **exiting** the call 65
(85) $K := \{(n(c, x), \{y_8/x, w_8/e\}), (n(c, x), \{y_8/x, w_8/i\})\}$
(86) $i := 3$
(87) $K := \{(n(c, x), \{x/a\}), (n(c, x), \{x/o\})\}$
(88) $ans\_n := \{(d, e), (c, a), (b, i), (c, o)\}$
(89) **exiting** the call 60
(90) **exiting** the call 53
(91) $K := \{(s(x), \{x/a\}), (s(x), \{x/o\})\}$
(92) $ans\_s := \{a, o\}$
(93) **exiting** the call 50
(94) **exiting** the call 48

(95) the main loop is repeated once more, without affecting $ans\_$ relations
(96) the returned result is $\{a, o\}$.