American Journal of Engineering Research (AJER)
e-ISSN: 2320-0847 p-ISSN : 2320-0936
Volume-6, Issue-1, pp-140-147
www.ajer.org
Research Paper Open Access

Bottom-Up Bisimilar Datalog Goals

Antoun Yaacoub, Zainab Assaghir
Applied Mathematics Department, Faculty of Science, Lebanese University, Lebanon

ABSTRACT: *We introduce the concept of bisimulation between Datalog goals using bottom-up evaluation technique: two Datalog goals are bisimilar with respect to a given Datalog program equipped with a bottom-up evaluation technique when their resolution trees are bisimilar. We define the notion of a resolution tree and we address the problem of deciding whether two given goals are bisimilar with respect to given programs. When the given program is a Datalog program equipped with a bottom-up evaluation technique, this problem is decidable in 2EXPTIME.*
Keywords: *Logic programming, Datalog, Bisimulation; Bottom-up, Decision method*

I. INTRODUCTION

Datalog bottom-up evaluation technique constitutes a trend in the deductive database evaluation strategy. In fact, deductive databases provide more expressive power than most relational databases, because they can naturally represent non-first-normal-form relations, and allow recursive view definitions.

It is easy to embed a programming interface to a deductive database in a logic programming language such as Prolog.

Search algorithms in deductive databases are generally divided in two parts: bottom-up algorithms and top-down algorithms.

Bottom-up algorithms generate logical consequences of the database until all answers to the goal are found. Top-down algorithms start with the goal and reduce it to subgoals. When we compare a typical Prolog top-down execution with the Datalog bottom-up execution we notice that the bottom-up evaluation reduces the overall cost due to constant per-join overheads, such as initialization costs, and per iteration overheads, such as updating the various predicate extensions and increases the degree of set-orientation. Bottom-up also guarantees termination for any pure Datalog program.

For this, and as is it was instructed by Antoun *et al.* [1,2], comparing two given Datalog programs and taking into account the shape of the SLD-trees they give rise to necessitates the comparison of infinitely many SLD-trees. Thus, in a first approach, they have restricted their study to the comparison of two given Datalog goals using top-down execution technique.

They showed that deciding whether two given goals are bisimilar with respect to a given general logic program is undecidable. Hence, a natural question was to restrict the language of logic programming. Thus, when the given logic program is hierarchical or restricted, the problem of deciding whether two given goals are bisimilar becomes decidable in 2EXPTIME using top-down execution technique.

In this paper, we extend the work done by Antoun *et al.* [1,2,3,4,5,6,7] and we consider Datalog programs equipped with bottom-up execution technique. We will say that, with respect to a fixed Datalog program P, two given goals are equivalent when their resolution trees are bisimilar. For this, we need to define what we mean by a resolution tree for a goal generated using a bottom-up execution technique. We examine the complexity of the following decision problem: given two Datalog goals F,G and a Datalog program P equipped with a bottom-up execution technqiue, determine if the resolution tree of $P \cup F$ and $P \cup G$ are bisimilar.

In section 2 of this paper, we will present some basic notions about Datalog programs, syntax and semantics. In section 3, we will refine the concept of bisimulation between Datalog goals. In section 4, we will address the problem of deciding whether two given goals are bisimilar with respect to a given Datalog program. At the end of the section, computational issues will be studied.

II. DATALOG PROGRAMS

Deductive database is like a relational database with a special query language (i.e. Datalog instead of SQL). It is also like an automated "theorem prover" which allows special kinds of formulas [8]. The relational database defines a number of relations called predicates "extensionally" (i.e. EDB) by enumerating the

mentioned tuples contained in the relation, and a logic program, which defines a number of predicates "intentionally" (i.e.. IDB). Since deductive database theory is based on logic programming, it borrows heavily from the field of logic programming [9]. It combines the benefits of representational and operational uniformity, recursion and declarative. Nonetheless, significant problems remain inherent in this synthesis.

The fact is that deductive databases provide more expressive power than most relational databases. It is easy to embed a programming interface to a deductive database in a general-purpose, or imperative, logic programming language such as Prolog [10].

Datalog is a non-procedural query language based on first-order logic that consists of a finite set of rules and a query literal. In Datalog we define two types of relations:

(1) base relations - physically stored in the database and
(2) derived relations - usually temporary relations that hold intermediate results.

Datalog was specifically designed to be used as a database language. Syntactically, Datalog is very similar to Prolog.

Another characteristic of Datalog is that each relation has a unique name and fixed arity (or number of attributes). In practice, we do not associate a "name" with an attribute, but rather its argument position. Consider the following example: [11]

r1(A1, A2, A3, A4)

Here r1 is the relation name and R1 is of arity 4.

All of the rules are built from literals: p(A1, A2,…,An), where p is the relation name and Ai are the variables or constants. Names that start with an upper case letter are variables.

The general form of a rule is as follows:

$p(x_1,x_2,\ldots,x_n)$:– $q_1(x_{11},x_{12},\ldots,x_{m1}),\ldots,q_k(x_{k1},\ldots,x_{mk})$, e.

In this form q_i are base or derived relation names, e is an arithmetic predicate (any number) and each x_i appearing in p appears in at least one of the q_i's.

The Datalog rule can be interpreted as:

p(...) is true if $q_1(\ldots)$ and $q_2(\ldots)$ and ... and $q_k(\ldots)$ and e is true.

Because Datalog often uses the bottom-up evaluation, the termination behavior is much better than that of Prolog. Normally we can assure that termination be guaranteed for a query evaluation. This is not possible for arbitrary programs.

Termination is a major issue in databases, and large subsets of queries/programs have been defined for which termination can be guaranteed.

Besides predicates and constants, Datalog-programs contain also variables, (i.e. place holders for domain elements). In Datalog an answer to an atomic query, say Q(X), is a set of constants that satisfy the query [9].

When it comes to query, search algorithms in deductive databases are generally divided in two parts: bottom-up algorithms and top-down algorithms [10].

The two approaches have been compared and contrasted mainly on strong (terminating) completeness over programs without function symbols, weak (non-terminating) completeness, coverage and efficiency [8].

The top-down approach is goal directed, often a good feature for efficiency. However, it has been shown that strict top-down approaches cannot be strongly complete over programs without function symbols [12]. Vieille has described top-down approaches in which lemmas are used to obtain termination over programs without function symbols, and there is a completeness proof for one of his methods [13].

The main disadvantage of top-down algorithms in databases is that the computations are performed a tuple at a time. Reduction of a goal to subgoals involves only a small amount o data. This usually results in a loss of efficiency [14].

The bottom-up approach is complete and terminating in programs without function symbols. But in certain situations it computes the entire deductive closure of a program in order to answer any question. Various methods based on "magic sets" have been proposed to focus bottom-up deduction. Sometimes a very large number of auxiliary "magic rules" is needed [15].

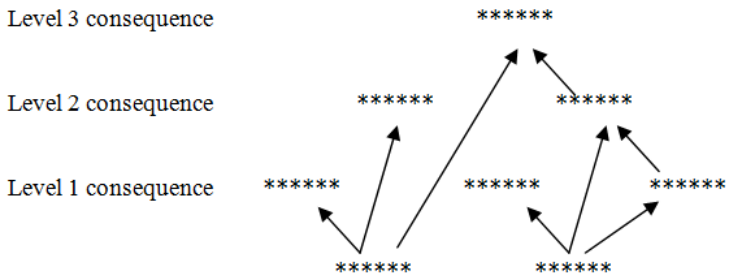


Figure:1 Bottom-up algorithms generate logical consequences of the database until all answers to the goal are found.

In Datalog bottom-up approach we assign rules in certain order and then evaluate them in that order.

Bottom-up evaluation of queries on deductive databases has many advantages over an evaluation scheme such as Prolog (top-down). In particular, it is able to avoid infinite loops by detecting repeated (possibly cyclic) subgoals. Bottom-up evaluation works by applying the rules to the given facts, thereby deriving new facts, and repeating this process with the new facts until no more facts are derivable. The query is considered only at the end, when the facts matching the query are selected [8].

The bottom-up method offers three significant advantages over the traditional top-down model:

(1) The declarative least Herbrand model semantics is guaranteed for positive Horn clause programs;
(2) Redundant derivations are avoided through memoing, leading to measurable gains in efficiency for programs in which goals or facts can be derived in many ways;
(3) As a consequence of the first advantage, no operational guarantees need to be made, and a number of semantic optimizations are possible. One example of a powerful optimization made possible by the declarative semantics is factoring [8].

Another advantage of bottom-up evaluation of logic programs is the increased degree of set-oriented computation. When the number of inferences made by two different evaluation techniques is identical, the technique that performs more set-oriented computation is expected to perform better in terms of the number of I/O operations. Ramakrishnan's theoretical analysis and performance results show that rule ordering can greatly reduce the number of rule applications, and therefore the number of joins in bottom-up evaluation, without making additional inferences.

This has two benefits:

(1) It reduces the overall cost due to constant per-join overheads such as initialization costs, and per iteration overheads such as updating the various predicate extensions and
(2) It increases the degree of set-orientation. The fewer the number of rule applications performed, the greater the number of inferences made in a single rule application. Again, this increases the degree of set-orientation, and hence decreases the number of I/O operations [16]. The reduction in cost due to ordering of rules depends on other techniques such as efficient join and indexing strategies, and duplicate elimination techniques.

The main disadvantage of bottom-up algorithms is that bottom-up algorithms are not goal-oriented. Thus, the search can involve a lot of irrelevant computation.

In the literature, three main algorithms [17] were extensively used and applied to the bottom up approach: Very Naive, Semi Naive and Magic Sets algorithms. In the following, we will not be concerned by which algorithm we will use, but instead we will be concerned by all the generated subgoals/atoms to draw our tree.

For example, the Very Naive algorithm starts from scratch i.e it sets the values of all IDB predicates to empty. Then, it computes all the possible facts round by round. Note that the query is considered only at the end of the algorithm. For example, if we have a goal: ?p(a,y), the answer of this goal is returned as soon as the algorithm terminates (i.e. after computing all the possible facts).

Example 1.

Let P be a Datalog program with the following input:

q(a,b).
q(b,c).
q(c,d).
p(x,y):- q(x,y).
p(x,y):- q(x,z), p(z,y).
Let the goal be: ?p(a;y).

In this example, we have 3 facts q(a,b), q(b,c) and q(c,d). Also, q is the only EDB predicate since it appears only in the body of the rules, whereas p is the only IDB predicate. Moreover, we have two rules as shown earlier.

Now let us execute this program using the Very-Naïve Bottom-up evaluation technique.

Initially, set all the IDB's to empty, here we only have p.

So P = empty, and Q = {(a;b); (b;c); (c;d)}

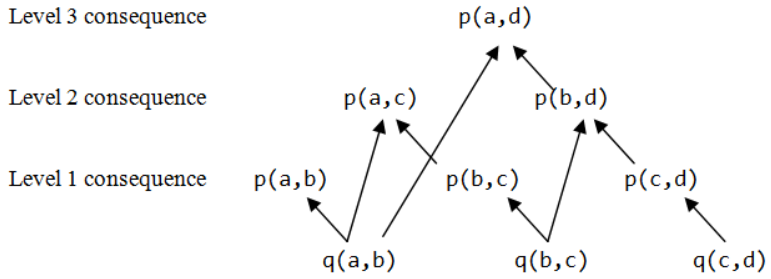
In round 1: P = {(a,b); (b,c); (c,d)}. Clearly, only the first rule works in the first round because P is empty, so it takes all the tuples from Q.

In round 2: P = {(a,b); (b,c); (c,d); (a,c); (b,d)}. Here, the second rule also works. Applying the join between p and q, we get the newly added tuples. The algorithm continues because the values of P are changed.

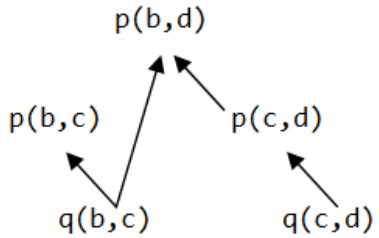
In round 3: P = {(a,b); (b,c); (c,d); (a,c); (b,d); (a,d)}. New facts are generated, so the algorithm continues.

In round 4: P = {(a,b); (b,c); (c,d); (a,c); (b,d); (a,d)}. No new facts are generated, the algorithm stops. The answer for the query is y = {b;c;d}.

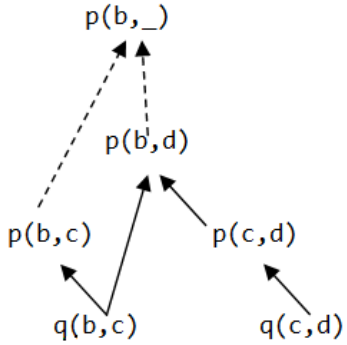
Let us now show the resolution tree of Datalog programs using the bottom-up evaluation algorithm.



Note that the above resolution tree is always finite. We call, for example, q(a,b) an atom without antecedent and that q(a,b) is an antecedent of p(a,b). We define the tree of a goal ?p(b,y) as the set of all sub-tree(s) rooted by ?p(b,_) where _ could be any constant. In the above example, the tree of the goal ?p(b,y) is the following:



In the case where, the resulting tree contains more than one root, we add a fictive root as in the following:



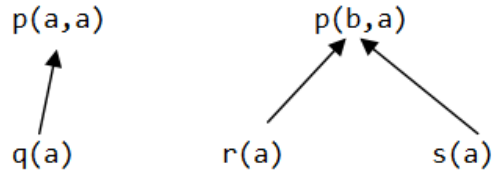
III. BISIMULATION

A bisimulation is a binary relation between goals such that related goals have "equivalent" trees. Let *P* be a Datalog program. A binary relation *Z* between Datalog goals is said to be a P-bisimulation iff it satisfies the following conditions for all Datalog goals *F*1,*G*1 such that *F*1*ZG*1:

- *F*1 is without antecedent iff *G*1 is without antecedent ,
- For each antecedent *F*2 of *F*1 and a clause in *P*, there exists an antecedent *G*2 of *G*1 and a clause in *P* such that *F*2*ZG*2,
- For each antecedent *G*2 of *G*1 and a clause in *P*, there exists an antecedent *F*2 of *F*1 and a clause in *P* such that *F*2*ZG*2.

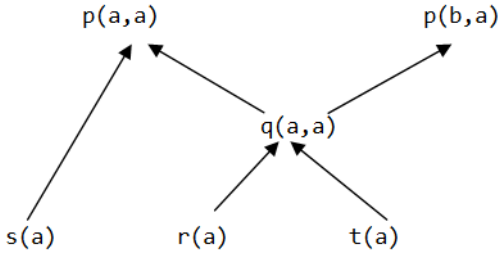
One can show that the set of all P-bisimulations is closed under taking arbitrary unions. This shows that:
Proposition 1. *There exists a maximal P-bisimulation, namely the binary relation Zmax between Datalog goals defined as follows: F*1*ZmaxG*1 *iff there exists a P-bisimulation Z such that F*1*ZG*1.
It follows immediately that:
Proposition 2. *Zmax is an equivalence relation on the set of all Datalog goals.*

Example 2. Let *P* be the following program:
q(a).
r(a).
s(a).
p(a,y) :-q(y).
p(b,y) :-r(y).
p(b,y) :s(y).
and let *F, G* be respectively the following goals ?p(a,y) and ?p(b,y).
The resulting resolution tree from the program gives the following:



Let *Z* be the binary relation between goals such that:
?p(a,a) Z ?p(b,a),
?q(a) Z ?r(a),
?q(a) Z ?s(a).
Obviously, *Z* is a *P-bisimulation*. Since *F Z G*, then *F Zmax G*.

Example 3. Let *P* be the following program:
s(a).
r(a).
t(a).
q(a,y) :-r(y),t(y).
p(b,y) :-q(a,y).
p(a,y) :s(a),q(a,y).
and let *F, G* be respectively the following goals ?p(a,y) and ?p(b,y).
The resulting resolution tree from the program gives the following:



Obviously, *Z* is not *P-bisimulation*. Since *p(a,a)* has an antecedent *s(a)* and *s(a)* has no antecedent, whereas *p(b,a)* has *q(a,a)* as antecedent and *r(a)* (or *t(a)*) as antecedent for *q(a,a).*
In this paper, we address the following decision problem :
(π) given a Datalog program *P* equipped with a bottom-up evaluation technique and Datalog goals *F*1,*G*1, determine whether *F*1*ZmaxG*1.

IV. BISIMULATION DECIDABILITY FOR DATALOG PROGRAMS EQUIPPED WITH BOTTOM-UP EVALUATION TECHNIQUE

We now study the computational complexity of the following decision problem:
(π) given a Datalog program *P* equipped with a bottom-up evaluation technique and Datalog goals *F*1,*G*1, determine whether *F*1*ZmaxG*1. In this respect, let *P* be a Datalog program. In the following algorithm, bothWithoutAntecedent(*F*1,*G*1) is a Boolean function returning true iff *F*1 is without antecedent and *G*1 is without antecedent. Moreover, antecedent(.) is a function returning the set of all antecedents of its argument with a clause of *P* whereas get-element(.) is a function removing one element from the set of elements given as

input and returning it. In order to demonstrate the decidability of (π), we need to prove the following lemmas for all Datalog goals $F1$,$G1$:

Lemma 1 (*Termination*). bisim($F1$,$G1$) *terminates.*
Lemma 2 (*Completeness*). *If* $F1ZmaxG1$, *then* bisim($F1$,$G1$) *returns* true.
Lemma 3 (*Soundness*). *If* bisim($F1$,$G1$) *returns* true, *then* $F1ZmaxG1$.
Let < be the binary relation on the set of all pairs of Datalog goals defined by: ($F2$,$G2$) < ($F1$,$G1$) iff
– the tree for $F1$ is deeper than the tree for $F2$,
– the tree for $G1$ is deeper than the tree for $G2$.
The depth of a tree is the depth of its longest branch. Remark that in this section, all trees are finite.
Obviously, < is a partial order on the set of all pairs of goals, and thus < is well-founded.

```
Algorithm  function bisim(F1,G1)
begin
 if bothWithoutAntecedent(F1,G1) then
   return true
 else
   SF ←− antecedent(F1)
   SG ←− antecedent(G1)
   if SF≠∅ and SG≠∅ then
    SF' ←− SF
    while SF'≠∅ do
     F2 ←− get-element(SF')
     found-bisim ←− false
     SG'←− SG
     while SG'≠∅ and found-bisim = false do
      G2 ←− get-element(SG')
      found-bisim ←− bisim(F2,G2)
          end while
     if found-bisim = false then
      return false
     end if
    end while
    SG'←− SG
    while SG'≠∅ do
     G2 ←− get-element(SG')
     found-bisim ←− false
     SF'←− SF
     while SF'≠∅ found-bisim = false do
      F2 ←− get-element(SF')
      found-bisim ←− bisim(G2, F2)
          end while
     if found-bisim = false then
      return false
     end if
     return true
    else
     return false
    end if
end
```

Proof of Lemma 1. The proof is done by <-induction on ($F1$,$G1$). Let ($F1$,$G1$) be such that for all ($F2$,$G2$), if ($F2$,$G2$) < ($F1$,$G1$) then bisim($F2$,$G2$) terminates. Since every recursive call to bisim that is performed along the execution of bisim($F1$,$G1$) is done with respect to a pair ($F2$,$G2$) of goals such that ($F2$,$G2$) < ($F1$,$G1$), then bisim($F1$,$G1$) terminates.

Proof of Lemma 2. Let us consider the following property: (*Prop*1($F1$,$G1$)): if $F1ZmaxG1$ then bisim($F1$,$G1$) returns true. Again, we proceed by <-induction. Suppose ($F1$,$G1$) is such that for all ($F2$,$G2$), if ($F2$,$G2$) < ($F1$,$G1$) then *Prop*1($F2$,$G2$).

Let us show that *Prop*1(*F*1,*G*1).

Suppose *F*1*ZmaxG*1. Hence, for all antecedents *F*2 of *F*1, there exists an antecedent *G*2 of *G*1 such that *F*2*ZmaxG*2, and conversely. Seeing that the tree is finite, then (*F*2,*G*2) < (*F*1,*G*1).

By induction hypothesis, *Prop*1(*F*2,*G*2). Since *F*2*ZmaxG*2, then bisim(*F*2,*G*2) returns true. As a result, one sees that bisim(*F*1,*G*1) returns true.

Proof of Lemma 3. It suffices to demonstrate that the binary relation *Z* defined as follows between Datalog goals is a bisimulation: *F*1*ZG*1 iff bisim(*F*1,*G*1) returns true. Let *F*1,*G*1 be Datalog goals such that *F*1*ZG*1.

Hence, bisim(*F*1,*G*1) returns true. Thus, obviously, *F*1 is without antecedent iff *G*1 is without antecedent, and the first condition characterizing bisimulations holds for *Z*. Now, suppose that *F*2 is a descendant of *F*1. Since bisim(*F*1,*G*1) returns true, then there exists a descendent *G*2 of *G*1 such that bisim(*F*2,*G*2) returns true, i.e. *F*2*ZG*2. As a result, the second condition characterizing bisimulations holds for *Z*. The third condition characterizing bisimulations holds for *Z* too, as the reader can quickly check. Thus *Z* is a bisimulation. _

As a consequence of lemmas 1 – 3, we have:

Theorem 1. Algorithm *is a sound and complete decision procedure for* (π).

It follows that (π) is decidable. Moreover,

Theorem 2. (π) *is in 2EXPTIME.*

Proof. Let *P* be a restricted Datalog program and *G* be a goal. Let *n* be the maximal number of atoms in the clauses of *P* or in *G*, *p* be the number of predicate symbols in *P*, *a* be the maximal arity of the predicate symbols in *P*, and *c* be the number of constants in *P*. Thus, the number of variables in *P* is about $n \times a$, the number of ground atoms is bounded by $p \times c^a$ and the total number of atoms is bounded by $p \times (c+n \times a)^a$. Moreover, as the number of goals of size *n* is bounded by $p^n \times (c + n \times a)^{n \times a}$, the number of pairs of goals of size *n* is bounded by $p^{2 \times n} \times (c + n \times a)^{2 \times a \times n}$. We conclude that the maximal depth *D* of a branch in the tree cannot exceed $p^n \times (c + n \times a)^{n \times a}$. Remark that our algorithm uses twice two nested loops. In fact, for a maximal depth *D*, the time complexity of the algorithm is approximately equal to 2 × (the time complexity of the algorithm for a depth $D - 1$) which is in turn equal to 4 × (the time complexity of the algorithm for a depth $D - 2$). Thus, by iterating the same operation until depth 1, one can show that for a depth *D*, the time complexity of our algorithm is about $2D \leq 2^{p^n \times (c + n \times a)^{n \times a}}$.

V. CONCLUSION

In this paper, we have introduced the concept of bisimulation between datalog goals relatively to a Datalog program equipped with a bottom-up evaluation technique: two Datalog goals are bisimilar with respect to a given program when their trees are bisimilar. As proved in Section III, deciding whether two given goals are bisimilar with respect to a given general Datalog program is decidable in 2EXPTIME. Future work can be dedicated to the study of the flow detection using the resulting resolution tree from a Datalog program using a bottom up evaluation technique.

REFERENCES

[1]. P. Balbiani, and A. Yaacoub, Deciding the bisimilarity relation between Datalog goals. In *European Conference on Logics in Artificial Intelligence* (JELIA), Toulouse, 26/09/2012- 28/09/2012 , L. Fari nas del Cerro, A. Herzig, and J. Mengin, Eds., Springer, p. 67–79.

[2]. A. Yaacoub, Towards an information flow in logic programming. International *Journal of Computer Science Issues (IJCSI) 9, 2*, 2012.

[3]. A. Yaacoub, and A. Awada, Inference Control On Information Flow In Logic Programming. *International Journal of Computer Science: Theory and Application 3(1)*, 2015, p. 13-22.

[4]. A. Yaacoub, Flux de l'information en programmation logique, doctoral diss., Universite Paul Sabatier - Toulouse III, 2012.

[5]. A. Yaacoub, A. Awada, and H. Kobeissi, Information Flow in Concurrent Logic Programming. *British Journal of Mathematics & Computer Science, 5(3)*, 2015, p. 367-382.

[6]. A. Yaacoub, and A. Awada, Information Flow in Datalog using Very Naive and Semi Naive Bottom-Up Evaluation Techiques. *International Journal of Computer Science: Theory and Application 5(2)*, 2016, p. 35-48.

[7]. A. Yaacoub, Bisimilarity, Datalog and Negation. *International Journal of Computer Science: Theory and Application 5(2)*, 2016, p. 28-34.

[8]. S. Brass, Bottom-Up Query Evaluation in Extended Deductive Databases. Dem Habilitation dissertation for Fachbereich Mathematik der Universitat at Hannover, 1996.

[9]. J.A. Fernandez, J. Gryz, and J. Minker, Disjunctive deductive databases: semantics, updates, and architecture. Institute for Advanced Computer Studies University of Maryland, College Park, MD, 1997.

[10]. D. Kemp, and P. Stuckey, Magic Sets and Bottom-up Evaluation of Well-Founded Models. *Bull Information Systems, Phoenix*, AZ, 1991.

[11]. P. Dasiewicz, Materials for ECE456, Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, CDN, 1997.
[12]. C. Beeri, S. Naqvi, R. Ramakrishnan, C. Shmueli, and S. Tsur, Sets and negation in a logic database language. In Proceedings of the *ACM Symposium on Principles of Database Systems*, San Diego, California, CA, 1987.
[13]. L. Vieille, A database-complete proof procedure based on SLD-resolution. *International Conference on Logic Programming, Melbourne*, Australia, 74-103, 1987.
[14]. J. Kuittinen, O. Nurmil, S. Sippu, and E. Soisalon-Soininen, Efficient Implementation of Loops in Bottom-Up Evaluation of Logic Queries. Department of Computer Science, University of Helsinki, SF, 1994
[15]. I. Balbin, and K. A. Meenakshi, Query Independent Method for Magic Set Computation on Stratified Databases. *International Conference on Fifth Generation Computer Systems*, Tokyo, Japan, p. 711-718, 1988.
[16]. R. Ramakrishnan, D. Srivastava, and S. Sudarshan, Controlling the Search in Bottom-Up Evaluation, University ofWisconsin, Madison, WI, 1999
[17]. J. Ullman, Principles of Databases and Knowledge base Systems, Volume I and II. Computer Science Press, 1988.