Magic Sets and Stratified Databases

Yangjun Chen*
Technical Institute of Changsha, Hunan, China

This article considers the efficient bottom–up query evaluation for stratified databases. We investigate the applicability of magic-set method to stratified databases containing negative body literals and show that *culprit cycles* cause unstratification. Based on the analysis, we present a labeling algorithm to distinguish the context for constructing magic sets, which is simpler and more efficient than the algorithms proposed by Balbin et al. [*J. Logic Programming,* 295–344 (1991)].

I. INTRODUCTION

There has been much recent research into query evaluation procedures for deductive databases. Most of this research has concentrated on the class of definite queries and databases. A good survey of this research is given by Bancilhon and Ramakerishnan in Ref. 1. There they classified query evaluation procedures as either top–down or bottom–up, interpreted or compiled, and recursive or iterative.

In evaluating answers to a query on a database, a bottom–up computation can naturally employ the existing optimization techniques developed for relational databases, which operate on a set at a time and always terminate on finite problems. As a consequence, bottom–up methods are the focus of much research into deductive databases.[2–8] However, bottom–up methods do not use the query to restrict the computation in the same ''goal-driven'' way that a top–down computation does. Therefore, many irrelevant tuples may be generated during a bottom–up computation. Research is currently concerned with methods for transforming databases so that bottom–up evaluation of the resulting database does use the query to restrict the computation. Two examples of these methods are *magic sets*[2,3,8] and the *less general counting sets.*[2,4,8] In this article, we confine ourselves to the use of magic-set transformation on (function-free) deductive databases that contain negative body literals. The magic set algorithms on these databases are based on a *sideways information-passing strategy* (SIPS).

Two important properties that databases should have are *stratification* and

*Author's current address: chen@informatik.tu-chemnitz.de or Dept. of Computer Science, Technical University Chemnitz-Zwickau, 09107 Chemnitz, Germany.

INTERNATIONAL JOURNAL OF INTELLIGENT SYSTEMS, VOL. 12, 203–231 (1997)
 CCC 0884-8173/97/030203-29

domain independence, which enable answers to queries to be evaluated both correctly and efficiently. A stratified database contains no recursion ''through negation'' as in

$$p \leftarrow \neg q,$$

$$q \leftarrow p.$$

The perfect model semantics for a stratified database is given by Ref. 10, which gives a declarative description of the desired output for a query with respect to the database. A domain independent database is one for which the set of correct answers to an atomic query is independent of the domains of variables in the database statement, i.e., the semantics of the database does not change when new constants are added to its language. *Allowedness* is a sufficient syntactic characterization of domain independency.[11] In addition, it leads to certain efficiencies during the computation, and so we restrict attention here only to *allowed databases.*

Many researchers have tried to use the magic-set method for stratified databases.[12,13] However, an observation shows that when we apply the generalized magic sets[3] to a stratified database, the resultant database may be unstratified.[14] Therefore, a different approach to database transformation is required to underlie the bottom–up computation. Here, we present a method to solve the problem, which is simpler and more efficient than the algorithms proposed by Balbin et al.[14] In particular, our scheme differs from the published ones in the following respects:

(1) A concept of culprit cycles is introduced in this article to manifest the causes of unstratification. This concept provides a deep insight into the problem and allows us to derive new algorithms. In contrast, Balbin et al.'s method is based on an incomplete analysis of such abnormal behaviors and thus the corresponding algorithms are more complicated, and somewhat misleading as well.
(2) In Balbin et al.'s method, no magic rules for negative body literals are ''physically'' constructed. Instead, a program segment is constructed for each negative body literal, which is evaluated by a function call when the corresponding negative body literal is encountered during the main process. This means that the bottom–up evaluation is done in a structured approach and an extra control mechanism is needed. Our method does not distinguish between the magic rules for positive and negative body literals and the transform of a program is done in a uniform manner.
(3) Our algorithm requires less time. If a stratified program consisting of n levels contains e predicates, the time complexity of our algorithm is $O(n \cdot e)$, whereas the time complexity of Balbin et al.'s method is $O(n^2 \cdot e)$.

The remainder of the article is organized as follows. Notations and preliminary definitions are presented in Section II. In Section III, we briefly describe the magic-sets computation of a deductive database which does not contain any negative body literals. In Section IV, we analyze the factors that contribute to the unstratification when the generalized magic-set method is applied to a database which contains negative body literals. Further, we present a labeling algorithm that can be used to remove the causes of the unstratification. In Section V, we prove the correctness of the labeling algorithm. Section VI analyzes the time complexity. Section VII is a brief summary.

II. BASIC CONCEPTS

The language of a deductive database consists of the variables, constants, and predicate names in the database. We adopt some informal notational conventions for them. Variables will normally be denoted by the letters *u, v, x, y,* and *z* (possibly subscripted). Constants will normally be denoted by the letters *a, b,* and *c* (possibly subscripted). Predicate names will normally be denoted by the letters *p, q, r, s,* and *t* (possibly subscripted). In the absence of function symbols, a term is either a constant or a variable. Occasionally, it will be convenient not to apply these conventions rigorously. In such a case, possible confusion will be avoided by the context.

An *atom* is an *n*-ary predicate, $p(t_1, t_2, \ldots, t_n)$, $n \geq 0$, where p is a predicate name and $t_1, t_2, \ldots, t_n$ are terms. A *literal* is an atom or the negation of an atom. A positive literal is just an atom. A negative literal is the negation of an atom.

A *rule* is a first-order formula of the form

$$q \leftarrow p_1, p_2, \ldots, p_m, \qquad m \geq 0.$$

q is called the *head* and the conjunction $p_1, p_2, \ldots, p_m$ is called the *body* of the rule. Each p_i is a body literal. When $m = 0$, the rule is of the form

$$q \leftarrow$$

and is known as a *unit clause.*

An atom $p(t_1, t_2, \ldots, t_n)$, $n \geq 0$ is *ground* when all of its terms $t_1, t_2, \ldots, t_n$, are constants. A *ground rule* is one in which each atom in the rule is ground. A *fact* is a ground unit clause. The definition of a predicate p is the set of rules which have p as the head predicate. A *base predicate* is defined solely by facts. The set of facts in the database is also known as the *extensional database.* A rule that is not a fact is known as a *derivation rule.* A *derived predicate* is a predicate which is defined solely by derivation rules. A derived (base) literal is one whose predicate is derived (base). The set of derivation rules is also known as the *intensional database* or *program*.

A deductive database **D** is a finite set of rules consisting of a program **P** and a set of facts **F**, which may contain negative information. We differentiate between two types of databases: when the rules in the database do not contain any negative body literals, we refer to it as a *positive database* and to the program as a *positive program*; when this is not the case, we refer to a *normal database* and a *normal program.*

For a program **P**, we construct a dependency graph G[10] representing a *refers to* relationship between the predicates. This is a directed graph where there is a node for each predicate and an arc from node q to node p iff the predicate q occurs positively or negatively in the body of a rule whose head predicate is p. When this literal is negative, the arc is a *negative arc* and is marked with a "¬" sign; otherwise it is a *positive arc* and is unmarked. A predicate p *depends on* a predicate q if there is a path of length greater than or equal to one from q to p. We denote the relation p *depends on* q by $p \leftarrow q$, where *depends on* is the

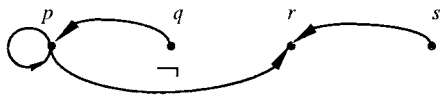

Figure 1. Dependency graph.

transitive closure of the *refers to* relation. If any arc in the path from q to p is negative, then we may also denote the dependency by $p \leftarrow\!\!\!\!\!\!\cdot\;\; q$. A predicate p is *recursive* if $p \leftarrow p$. Two predicates p and q are mutually *recursive* if $p \leftarrow q$ and $q \leftarrow p$.

Example 2.1. The dependency graph corresponding to the program:

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

is shown in Figure 1.

DEFINITION 2.1. *A path in G is a negative path if at least one arc in the path is negative. A cycle in G is a negative cycle if at least one arc in the cycle is negative.*

DEFINITION 2.2. *A logic program,* **P,** *is stratified if its predicates can be partitioned into levels so that, in every program clause,* $q \leftarrow p_1, p_2, \ldots, p_m$, *the level of every predicate in a positive literal in the body is less than or equal to the level of q and the level of every predicate in a negative literal is strictly less than the level of q.*

Throughout this article, we assume also, as in Ref. 15, that the levels of a stratified program are $0, 1, \ldots, n$ for some integer n, where n is the minimum number satisfying the above definition. In this case, **P** is said to have the maximum level n and is denoted $\mathbf{P} = \mathbf{L}_0 \cup \ldots \cup \mathbf{L}_n$, where $\mathbf{L}_i$ is a set of clauses whose head predicates have level i. The partitioning of **P** into the sets $\mathbf{L}_0, \ldots, \mathbf{L}_n$ is called *stratification,* and each set $\mathbf{L}_i$ is called a *stratum.*

PROPOSITION 2.1. **P** *is stratified if and only if there does not exist a negative cycle in G of* **P.**

Proof. See Ref. 10. ■

Consider any stratification $\mathbf{L}_0, \ldots, \mathbf{L}_n$ of a program **P**. Let M denote a set of ground atoms. Define T_i, $i = 0, \ldots, k$, to be the operator on M as follows. For every M and every ground atom p, $p \in T_i(M)$ if and only if $p \in M$ or for some rule $q \leftarrow p_1, p_2, \ldots, p_m$ in $\mathbf{L}_i$, there is a substitution θ of constants for variables such that $p = q\theta$ and for each body literal p_i, if $p_i\theta$ is positive then it

is in M, otherwise it is not in M. The sets $M_0, \ldots, M_k$ of ground atoms are defined by the equations

$$M_0 = \varnothing,$$
$$M_i = T_i^j(M_{i-1}) \qquad (i = 0, \ldots, k),$$

where T_i^j is T_i applied j times until the fixpoint is reached for some value of $j \geq 0$. Then, the semantics of **P** is M_k, which does not depend on the choice of stratification of **P** and is identical to the *perfect* model.[10]

III. MAGIC SETS METHOD

In this section, we consider the magic sets method for positive database. This method is based on the idea of the sideways information-passing strategy and improves efficiency by restricting the computation to tuples that are related to the query. Essentially, the magic-set transformation does two things:[3] it creates new magic rules, and it introduces new body literals into the original rules to form modified rules. The modified database enable a bottom–up computation to generate fewer irrelevant tuples.

A. Sideways Information-Passing Strategy

A sideways information-passing strategy (SIPS) is an inherent component of any query evaluation strategy. Informally, for a rule of a program, a SIPS represents a decision about the order in which the predicates of the rule will be evaluated, and the variables for which the values are passed from predicates to other predicates during evaluation. Intuitively, a SIPS describes how bindings passed to a rule's head by unification are used to evaluate the predicates in the rule's body. Thus, a SIPS describes how we evaluate a rule when a given set of head arguments are bound to constants. Consider, for example, the familiar *ancestor* predicate where *ancestor*(x, y) is *true* if y is an ancestor of x, and where *parent* is a base predicate, such that *parent*(x, y) is *true* if y is a parent of x:

(1) *ancestor*$(x, y) \leftarrow$ *parent*(x, y),
(2) *ancestor*$(x, y) \leftarrow$ *parent*(x, z), *ancestor*(z, y).

The query $\leftarrow$ *ancestor*$(john, y)$ retrieves all the ancestors of *john*. By unification, the variable x in the second rule is bound to *john*. We can evaluate *parent*(x, z) using this binding, and obtain a set of bindings for z. These are passed to *ancestor* to generate subgoals, which in this case have the same binding pattern. The values for z can then be said to be passed sideways from *parent* to *ancestor*.

Generalizing from this example, we may say that the basic step of sideways information passing is to evaluate a set of predicates (possibly with some arguments bound to constants), and to use the results to bind variables appearing in

another predicate. It is important to stress that SIPS do not say how this information is passed. Indeed, there may be more than one way to pass the information for given SIPS. For example, SIPS do not specify whether the information is passed on a tuple-at-a-time basis, or as a set of tuples. SIPS describe only the flow of information with respect to a rule with a given set of head arguments bound to constants, which will be generated when a top–down strategy like Prolog evaluates the rule. In general, SIPS are associated with a rule according to the query form. Different query forms, such as $ancestor^{bf}(x, y)$ and $ancestor^{fb}(x, y)$ (see below), usually have different SIPS for the same set of defining rules. The choice of one SIPS over another is guided by factors such as the current and expected size of different relations and the indexing mechanism employed.

The following definition of allowed SIPS is borrowed from.[14] It is the refining of the definition given by Ref. 3 and preserves the allowedness under the magic-set transformation.[14] In a given rule, two literals are called *connected* if they share a common argument. This is extended in the obvious way to connection through a chain of predicates, where each adjacent pair shares an argument.

DEFINITION 3.1 *Let $B(\mathbf{R})$ denote the set of body literals for a rule $\mathbf{R}$, and let p^a be a special literal, denoting head literal restricted to its bound arguments. An allowed SIPS for a rule $\mathbf{R}$ is a labeled bipartite graph $G(V_1, V_2)$, where V_1 is the set of subsets of $B(\mathbf{R}) \cup \{p^a\}$ and $V_2 = B(\mathbf{R})$, and which satisfies the following two conditions:*

(1) *Each arc is of the form $N \rightarrow_\chi p$, where $N \in V_1$, $p \in V_2$. The label χ stands for a nonempty set of variables which satisfies the following restrictions:*
   (i) *each variable in χ appears in a member (a predicate) of N and in p;*
   (ii) *each literal in N is connected to p;*
   (iii) *each variable appearing in N appears in a positive literal in N, or in a bound argument position of p^a in N.*

(2) *There exists a total order of $B(\mathbf{R}) \cup \{p^a\}$ in which:*
   (i) *p^a precedes all member of $B(\mathbf{R})$;*
   (ii) *any literal which is not in the graph follows every literal that is in the graph; and*
   (iii) *for every arc $N \rightarrow_\chi p$, if the literal $p' \in N$, then p' precedes p.*

Example 3.1. Consider the above program defining *ancester*

(1) $ancester(x, y) \leftarrow parent(x, y)$,
(2) $ancestor(x, y) \leftarrow parent(x, z), ancestor(z, y)$.

Let the query be $\leftarrow ancestor(john, y)$ and $ancestor_1$ be a special predicate, denoting $ancestor(x, y)$ restricted to its first bound argument. An arc for the first rule might be

$$\{ancestor_1(x)\} \rightarrow_{\{x\}} parent(x, y).$$

The SIPS for the second rule is

$$\{ancestor_1(x)\} \rightarrow_{\{x\}} parent(x, y),$$
$$\{ancestor_1(x), parent(x, z)\} \rightarrow_{\{z\}} ancestor(z, y).$$

B. The Adorned Program

In terms of the SIPS for a program, we can adorn the program. This is done by annotating predicates with a character string, which is called *adornment.* An adornment for an m-ary predicate $p(t_1, t_2, \ldots, t_m)$ is a string of length m made up of the letters b and f, where b stands for *bound* and f stands for *free.* We obtain an adornment for a predicate as follows. During a computation, each argument, t_i, $1 \leq i \leq m$, of the literal $p(t_1, t_2, \ldots, t_m)$ is expected to be bound or free, depending on the information flow (SIPS). If t_i is expected to be bound (free), it acquires a $b(f)$ annotation, and so the length of the adornment string is m. Note that the adornment is attached to the predicate and becomes part of it.

Example 3.2. The following is the adorned rule set corresponding to the familiar *ancestor* predicate for the SIPS of Example 3.1:

(1) $ancestor^{bf}(x, y) \leftarrow parent(x, y)$,
(2) $ancestor^{bf}(x, y) \leftarrow parent(x, z), ancestor^{bf}(z, y)$.

C. The Magic-Set Algorithm

Now we consider the magic-set transformation of a program which does not contain any negative body literals. Magic-set algorithms are program transformations that take an initial adorned program and query and return a modified program which gives the same answers for a particular query as the initial program. Using the bottom–up method, the transformed program generates fewer irrelevant tuples than the initial program. There have been several magic-set algorithms reported in the literature.[2,3,16]

A common trait among these algorithms is that, based on the adornment of the head and body literals, some new positive literals are introduced into the body of rules, and new rules are added to the program which define these literals. The new literals are called magic literals and are related to the existing literals of the program as follows. For a positive adorned predicate p^a with l bound argument positions where $l > 0$, define the magic predicate of p^a to be the predicate whose name is the predicate name of p^a prefixed with *''Magic–''* and whose arity is l. The new rules defining the magic predicates are called magic rules.

The following is a magic-set algorithm.[14] The input of the algorithm consists of the adorned query, $q^a(\bar{v})$, the adorned program, $\mathbf{P}^a$, and the corresponding set of SIPS, $\mathbf{S}^a$. The output of the algorithm is the modified version of the adorned program plus the magic rules, $\mathbf{P}^{am}$, and the seed, $magic\text{–}q^a(\bar{v}_d)$, where $\bar{v}_d$ is the vector of arguments which are bound in the adornment a of q. In the algorithm, the following definitions are used:

bodyLit(N) denotes the conjunction of body literals of a rule $\mathbf{R}^a$ in N, where N is the tail of an arc N $\rightarrow_\chi p$ in the SIPS for $\mathbf{R}^a$;

magic($r^a(\bar{v})$) returns a literal *magic–*$r^a(\bar{v}_d)$.

```
function magic-set-transformation (q^a(v̄), P^a, S^a)
   P^am := ∅
   for each rule R^a in P^a of the form p ← p_1, p_2, . . . , p_m do
    add the rule p ← magic(p), p_1, p_2, . . . , p_m to P^am
    for each arc N →_χ r in the SIPS associated with R^a do
      if p is in N then
         add the rule magic(r) ← magic(p), bodyLit(N) to P^am
      else add the rule magic(r) ← bodyLit(N) to P^am
   return (magic_q^a(v̄_d), P^am)
```

Example 3.2. Consider the adorned query $\leftarrow ancestor^{bj}(john, y)$ to the program

$$ancestor(x, y) \leftarrow parent(x, y),$$
$$ancestor(x, y) \leftarrow parent(x, z), ancestor(z, y).$$

If the chosen SIPS is like that of Example 3.1, we will have the following adorned program:

$$ancestor^{bf}(x, y) \leftarrow parent(x, y),$$
$$ancestor^{bf}(x, y) \leftarrow parent(x, z), ancestor^{bf}(z, y).$$

Then, the magic rules and modified rules are:

$$ancester^{bf}(x, y) \leftarrow magic\text{–}ancestor^{bf}(x), parent(x, y),$$
$$ancestor^{bf}(x, y) \leftarrow magic\text{–}ancestor^{bf}(x), parent(x, z), ancestor^{bf}(z, y),$$
$$magic\text{–}ancestor^{bf}(z) \leftarrow magic\text{–}ancestor^{bf}(x), parent(x, z),$$
$$magic\text{–}ancestor^{bf}(john).$$

IV. A LABELING ALGORITHM

The magic-set transformation does not always preserve stratification and allowedness when we use it for a stratified and allowed normal database. Therefore, it is necessary to develop a different approach to database transformation. To solve these problems, Balbin et al.[14] proposed a structured bottom–up method and a modification of SIPS (which is called allowed SIPS). The method employs a labeling algorithm to distinguish the context for constructing magic sets and preserves stratification when magic sets constructed for negative literals are not taken into account. For the allowed SIPS the magic-set transformation preserves allowedness. Since the magic rules for negative literals are not constructed in the structured bottom–up method, a duplication of rules is required to deal with the negative literals. Here, we present a new labeling algorithm which can distinguish the context for the magic sets constructed for both positive and

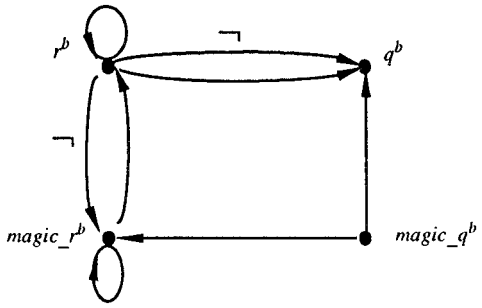

Figure 2. Dependency graph of transformed program of Example 4.1.

negative literals. We assume that our SIPS are allowed. We have already given the extract definition of allowed SIPS in the last section.

A. Causes of Unstratification

We differentiate among three cases of unstratification:

(1) A literal occurs both positively and negatively in the body of a rule.
(2) A negative literal occurs multiply in the body of a rule.
(3) A negative literal occurs in a recursive rule.

Case 1. Mixing occurrence of a literal. In the following, we omit explicit SIPS for simplicity of exposition, and assume default SIPS where the tail of the arc for each body literal p includes all literals to the left of p in the rule (including the head).

Example 4.1. Consider the following program:

$$q(x) \leftarrow p_1(x), \neg r(x), p_2(x, y), r(y),$$
$$r(x) \leftarrow p(x, y), r(y),$$

where p, p_1, and p_2 are base predicates. For the query $\leftarrow q^b(\mathrm{a})$ the magic rules and modified rules are

$$q^b(x) \leftarrow magic\text{–}q^b(x), p_1(x), \neg r^b(x), p_2(x, y), r^b(y),$$
$$magic\text{–}q^b(\mathrm{a}),$$
$$r^b(x) \leftarrow magic\text{–}r^b(x), p(x, y), r^b(y),$$
$$magic\text{–}r^b(x) \leftarrow magic\text{–}q^b(x), p_1(x),$$
$$magic\text{–}r^b(y) \leftarrow magic\text{–}q^b(x), p_1(x), \neg r^b(x), p_2(x, y),$$
$$magic\text{–}r^b(y) \leftarrow magic\text{–}r^b(x), p(x, y).$$

The dependency graph is shown in Figure 2 and contains the negative cycle:

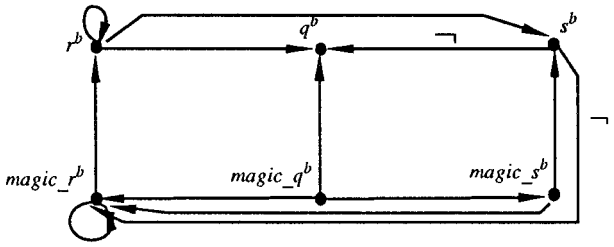

Figure 3. Dependency graph of transformed program of the modification of Example 4.1.

$$r^b \leftarrow magic\text{–}r^b \Leftarrow r^b.$$

Let us examine the source of the unstratification. This can be done by simulating a top–down computation of the rules. Consider, for example, the query $\leftarrow q^b(x)$. A top–down computation will evaluate the subquery $r^b(x)$ with the value for x which satisfies $p_1(x)$. The negative literal $\neg r^b(x)$ succeeds if this subquery fails. This value for x satisfies q provided that $p_2(x, y)$ is proven and a value for y satisfies the query $r^b(y)$. It is important to note that the two queries to *r,* the negative subquery and the positive subquery, are independent of each other. That is, they are evaluated in two different contexts. At the time that $r^b(y)$ is asked, $\neg r^b(x)$ has already been satisfied. However, the construction of the magic rules for r does not separate the two contexts. The magic rules for a positive literal and a negative literal are treated as one, when they should be separated and used in accordance with the corresponding contexts. The mixing of contexts causes the unstratification problem.

Now we consider a modification of Example 4.1:

$$q(x) \leftarrow p_1(x), \neg s(x), p_2(x, y), r(y),$$
$$s(x) \leftarrow p(x, y), r(y),$$
$$r(x) \leftarrow p(x, y), r(y),$$

and give the same query as above. We have the transformed rules as follows:

$$q^b(x) \leftarrow magic\text{–}q^b(x), p_1(x), \neg s^b(x), p_2(x, y), r^b(y),$$
$$magic\text{–}q^b(\mathrm{a}),$$
$$s^b(x) \leftarrow magic\text{–}s^b(x), p(x, y), r^b(y),$$
$$r^b(x) \leftarrow magic\text{–}r^b(x), p(x, y), r^b(y),$$
$$magic\text{–}s^b(x) \leftarrow magic\text{–}q^b(x), p_1(x),$$
$$magic\text{–}r^b(y) \leftarrow magic\text{–}q^b(x), p_1(x), \neg s^b(x), p_2(x, y),$$
$$magic\text{–}r^b(y) \leftarrow magic\text{–}s^b(x), p(x, y),$$
$$magic\text{–}r^b(y) \leftarrow magic\text{–}r^b(x), p(x, y).$$

From the dependency graph of the transformed rules (shown in Fig. 3), we can see that

$$s^b \leftarrow r^b \leftarrow magic\text{–}r^b \leftarrow s^b$$

constructs a negative cycle.

The cause of the unstratification is the same as that of Example 4.1, which can be seen, if we evaluate the query $\leftarrow q^b(x)$ top–down and expand the subquery $\neg s$ by substituting s with the body of the rule defining s. In such a way, we will have the following subgoal during the process:

$$\leftarrow \neg(p(x, y_0) \wedge r(y_0)), p_2(x, y), r(y),$$

which is logically equivalent to

$$\leftarrow \neg\mathbf{r}(y_0), \mathbf{p}_2(x, y), \mathbf{r}(y),$$

$$\leftarrow \neg p(x, y_0), p_2(x, y), r(y).$$

As analyzed above, $\neg r(y_0)$ and $r(y)$ in the first rule (in bold) are independent of each other. However, when we transform the above rules and construct magic rules for r, we will lose the separation of the contexts for r, which underlies the unstratification.

Case 2. Multi-occurrence of a negative literal.

Example 4.2. We modify the program of Example 4.1 by negating the second occurrence of r

$$q(x) \leftarrow p_1(x), \neg r(x), p_2(x, y), \neg r(y),$$

$$r(x) \leftarrow p(x, y), r(y),$$

and give the query $\leftarrow q^b(\mathrm{a})$. The transformed rules are

$$q^b(x) \leftarrow magic\text{–}q^b(x), p_1(x), \neg r^b(x), p_2(x, y), \neg r^b(y),$$

$$magic\text{–}q^b(\mathrm{a}),$$

$$r^b(x) \leftarrow magic\text{–}r^b(x), p(x, y), r^b(y),$$

$$\mathrm{magic}\text{–}r^b(x) \leftarrow magic\text{–}q^b(x), p_1(x)$$

$$magic\text{–}r^b(y) \leftarrow magic\text{–}q^b(x), p_1(x), \neg r^b(x), p_2(x, y),$$

$$magic\text{–}r^b(y) \leftarrow magic\text{–}r^b(x), p(x, y).$$

There is a negative cycle:

$$r^b \leftarrow magic\text{–}r^b \leftarrow r^b.$$

This negative cycle is the same as that of Example 4.1. The source of the unstratification is also the losing of the separation of contexts. If the program is evaluated top–down, the query to $\neg r(x)$ and the query to $\neg r(y)$ are independent of each other. That is, at the time that $\neg r(y)$ is asked, $\neg r(x)$ has already been satisfied. Note that in answering a negative literal $\neg r$ we first compute the entire extension for r and then compare the extension of r with the extension of the remaining body literals and perform the negation operation using set difference. Therefore,

it is necessary, by a bottom–up computation, to separate the contexts for $\neg r(x)$ and $\neg r(y)$ considering the way of evaluating a negative literal. However, the magic-set transformation mixes the contexts and gives rise to the unstratification problem.

Case 3. Occurrence of negative literals in a recursive rule.

Example 4.3. Consider the following program:

$$q(x) \leftarrow \neg r(x), p_1(x, y), q(y),$$
$$q(x) \leftarrow p_2(x),$$
$$r(x) \leftarrow p_3(x),$$

where p_1, p_2, and p_3 are base predicates. For the query $\leftarrow q^b(\mathrm{a})$ the magic rules and modified rules are

$$q^b(x) \leftarrow magic\text{–}q^b(x), \neg r^b(x), p_1(x, y), q^b(y),$$
$$magic\text{–}q^b(\mathrm{a}),$$
$$q^b(x) \leftarrow magic\text{–}q^b(x), p_2(x),$$
$$r^b(x) \leftarrow magic\text{–}r^b(x), p_3(x),$$
$$magic\text{–}r^b(x) \leftarrow magic\text{–}q^b(x),$$
$$magic\text{–}q^b(x) \leftarrow magic\text{–}q^b(x), \neg r^b(x), p_1(x, y).$$

The negative cycle is

$$r^b \leftarrow magic\text{–}r^b \leftarrow magic\text{–}q^b \leftarrow r^b.$$

To show the source of unstratification, we expand the recursive predicate $q(y)$ in the first rule of the program. Then we have

$$q(x) \leftarrow \neg r(x), p_1(x, y), \neg r(y), p_1(y, y_0), q(y_0).$$

From this we can see that the source of the unstratification is the same as that of Case 2.

B. The Labeling Algorithm

The idea behind the labeling algorithm is to distinguish the context for constructing magic sets. In terms of the three cases of unstratification, three different labeling strategies are employed. For the first case, we explicitly label p when it appears after a negative body literal $\neg q$ in a rule (here we assume that the information passing strategy is from left to right.) and simultaneously appears in the defining rules for q or there exists a path in the dependency graph from p to q. For the second case, we number the different occurrence of a negative body literal in a rule. For the third case, we give each negative body

literal in a recursive rule a dynamic subscript which will be changed with each application step of the T operation.[17]

The input to the labeling algorithm consists of $\mathbf{P}^a$ and the corresponding set of SIPS $\mathbf{S}^a$. The program $\mathbf{P}^a$ is first arranged into a stratification so that $\mathbf{P}^a = \cup_{i=1}\mathbf{L}_i$ and associated SIPS are $\mathbf{S}^a = \cup_{i=1}\mathbf{S}_i$, where each set of $\mathbf{L}_i$ is a stratum and $\mathbf{S}_i$ is associated with the rules in $\mathbf{L}_i$. The output of the algorithm is the labeled program $\mathbf{P}^l = \cup_{i=1}\{\mathbf{L}_i' \cup \mathbf{I}_i\}$, where $\mathbf{L}_i'$ is the labeled version of $\mathbf{L}_i$ and $\mathbf{I}_i$ is an associated set of newly constructed rules, and associated SIPS $\mathbf{S}^l$.

In the algorithm, there are three stages with each handling a different case of unstratification. In the first stage, we number the different occurrence for each negative body literal in a rule. In the second stage, we give each negative body literal a dynamic subscript when it appears in a recursive rule. Finally, we label each body literal p when there exists a sequence of paths connecting it to a negative body literal $\neg q$ in the same rule, or there exists a sequence of paths with at least one path being negative connecting it to a positive body literal q in the same rule and there is an arc of the form $\mathrm{N} \rightarrow r$ in the SIPS such that $q \in \mathrm{N}$ and $p = r$. (Note that these paths are not necessarily in the same direction; see below.)

```
procedure negnumber()
   begin
    for j := 1 to n do
    for each q ∈ negBodyList(j) do
        number the different occurrence of q in a rule of L_j;
    for each k such that q-k is a new numbered predicate do
        let m be the number of the stratum in which q is defined;
        copy the rules defining q to I_m;
        copy the corresponding SIPS to S_m^l;
        replace each q in I_m and S_m^l with q-k;
end
```

The first stage of the algorithm calls *negnumber*. The procedure performs two actions. First, it examines each stratum and numbers the different occurrence of each negative body literal. Secondly, for each numbered predicate q–k, it creates a new version of the rules defining q–k. In the algorithm, we make use of the following set:

> *negBodyList*(j) is the set of predicates that appear as negative body literals in the straum $\mathbf{L}_j$.

After *negnumber* has been executed we call the *dynlabel* procedure. The purpose of *dynlabel* is to subscript those negative body literals which appear in recursive rules. In addition, for each subscripted predicate p_i a new version of the rules defining p_i is created. Because the subscript will be changed with each application step of the T operation, we remove the unstratification caused by these negative literals. In the algorithm, *negBodyList*$^n(j)$ is the set of predicates that appear as numbered and unnumbered negative body literals in the stratum $\mathbf{L}_j$ and $\mathbf{I}_j$.

```
procedure dynlabel()
   begin
    for j := 1 to n do
      for each p ∈ negBodyList^n(j) do
       if p appears in a recursive rule then
         begin
           replace each p in all L_m, I_m and S_m^l with p_i;
         end
end
```

At last we call the *label* procedure, the third stage of the algorithm. In the procedure, each body literal p is labeled when there is an arc of the form $\mathrm{N} \rightarrow r$ in the SIPS such that $p = r$ and $q \in \mathrm{N}$ for some q and either of the following conditions holds:

(1) There exists a sequence of paths

$$\mu = (\mu_1, \mu_2, \ldots, \mu_h)$$

connecting it to a negative body literal $\neg q$ in the same rule.

(2) There exists a sequence of paths with at least one path being negative connecting it to a positive body literal q in the same rule.

Note that the paths in μ are not necessarily in the same direction. Each path μ_k, where $1 < k < h$, has one endpoint in common with the perceding path μ_{k-1}, and the other endpoint in common with the succeeding path μ_{k+1}. Assume μ_k and μ_{k+1} are not in the same direction and have the same node as the terminal node. If the last edges of μ_k and μ_{k+1} are $\alpha \rightarrow \beta$ and $\alpha' \rightarrow \beta$, respectively, and in the SIPS there is an arc of the form $\mathrm{N} \rightarrow r$ such that $\alpha' \in \mathrm{N}$ and $\alpha = r$, in terms of the algorithm of magic-set transformation, we will have a path consisting of only magic predicates in the dependency graph of the transformed version, which has one endpoint in common with μ_{k+1} and in the same direction as μ_{k+1}. The same analysis applies to the case where μ_k and μ_{k+1} are not in the same direction and have the same node as the initial node. Therefore, for the paths whose directions are not the same, we need to check the existence of the associated SIPS. But in the description of the algorithm, we omit the detail for simplicity. The following new functions are used by *label.*

pathConn(r, p) returns *true* if r and p are both body literals in the same rule, r is negative and there exists a sequence of paths connecting r and p in the dependency graph, or r is positive but at least one path in the sequence of the paths is negative; otherwise it returns *false.*

labeled(p) returns the labeled counterpart of p.

```
procedure label()
   begin
     Plabel := ∅
     for j := n downto 1 do
       for each unlabeled derived literal p appears in L_j
         if pathConn(r, p) and N → γ such that r ∈ N and p = γ then
```

```
If p ∈ Plabel then replace p with labeled(p)
   else
      begin
       replace p by label-p-j;
       let m be the number of the stratum in which p is defined;
       copy the rules defining p to I_m;
       copy the corresponding SIPS to S_m^l;
       replace each p in I_m and S_m^l with label-p-j;
       Plabel:= Plabe ∪ {p}
      end
end
```

C. Sample Trace

In this subsection, we trace the steps of the algorithm for a particular example. We omit variables and SIPS for simplicity.

Example 4.4. Consider the following program:

$$\mathbf{L}_3 \quad p \leftarrow s, \neg r, t, \neg r, q, p,$$

$$\mathbf{L}_2 \quad r \leftarrow q,$$

$$\mathbf{L}_1 \quad q \leftarrow q_0.$$

Here s, t, and q_0 are base predicates. The $\mathbf{L}_i$ form the stratification. After the magic transformation the rules are

$$p \leftarrow magic\text{–}p, s, \neg r, t, \neg r, q, p,$$

$$r \leftarrow magic\text{–}r, q,$$

$$q \leftarrow magic\text{–}q, q_0,$$

$$magic\text{–}r \leftarrow magic\text{–}p, s,$$

$$magic\text{–}r \leftarrow magic\text{–}p, s, \neg r, t,$$

$$magic\text{–}q \leftarrow magic\text{–}p, s, \neg r, t, \neg r,$$

$$magic\text{–}p \leftarrow magic\text{–}p, s, \neg r, t, \neg r, q,$$

$$magic\text{–}q \leftarrow magic\text{–}r.$$

The dependency graph with three negative cycles is shown in Figure 4. The three negative cycles are: $r \leftarrow magic\text{–}r \leftarrow magic\text{–}p \overset{-}{\leftarrow} r$, $r \leftarrow q \leftarrow magic\text{–}q \overset{-}{\leftarrow} r$ and $r \leftarrow magic\text{–}r \overset{-}{\leftarrow} r$.

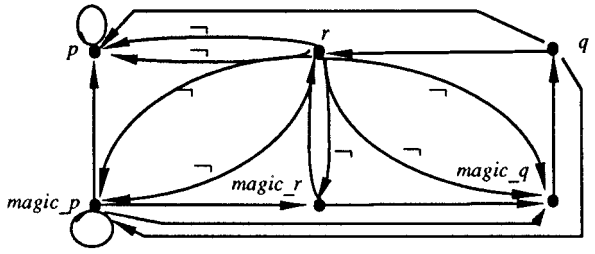

Figure 4. Dependency graph of transformed program of Example 4.4.

First, we execute *negnumber.* The resultant strata are:

$\mathbf{L}_3$ $p \leftarrow s, \neg r, t, \neg r, q, p;$ $\quad$ $\mathbf{I}_3$ $p \leftarrow s, \neg r\text{–}1, t, \neg r\text{–}2, q, p;$

$\mathbf{L}_2$ $r \leftarrow q;$ $\quad$ $\mathbf{I}_2$ $r\text{–}1 \leftarrow q, r\text{–}2 \leftarrow q;$

$\mathbf{L}_1$ $q \leftarrow q_0;$ $\quad$ $\mathbf{I}_1$ $\varnothing;$

Before we perform *dynlabel,* note that the dependency graph of the relevant magic transformed program of the resultant strata still contains the cycles: $r\text{–}1 \leftarrow magic\text{–}r\text{–}1 \leftarrow magic\text{–}p \Leftarrow r\text{–}1$, $r\text{–}2 \leftarrow magic\text{–}r\text{–}2 \leftarrow magic\text{–}p \Leftarrow r\text{–}2$, $r\text{–}1 \leftarrow q \leftarrow magic\text{–}q \Leftarrow r\text{–}1$, and $r\text{–}2 \leftarrow q \leftarrow magic\text{–}q \Leftarrow r\text{–}2$.

After the application of *dynlabel,* the resultant program is

$\mathbf{L}_3$ $p \leftarrow s, \neg r, t, \neg r, q, p;$ $\quad$ $\mathbf{I}_3$ $p \leftarrow s, \neg r\text{–}1_i, t, \neg r\text{–}2_i, q, p;$

$\mathbf{L}_2$ $r \leftarrow q;$ $\quad$ $\mathbf{I}_2$ $r\text{–}1_i \leftarrow q, r\text{–}2_i \leftarrow q;$

$\mathbf{L}_1$ $q \leftarrow q_0;$ $\quad$ $\mathbf{I}_1$ $\varnothing;$

The dependency graph of the resultant strata contains the negative cycles: $r\text{–}1_i \leftarrow q \leftarrow magic\text{–}q \Leftarrow r\text{–}1_i$ and $r\text{–}2_i \leftarrow q \leftarrow magic\text{–}q \Leftarrow r\text{–}2_i$.

At this point we execute the *label* procedure. That is, we label a literal p when there exists a sequence of paths connecting it to another body literal q in the same rule, with at least one path being negative and there is an arc of the form $\mathrm{N} \rightarrow r$ in the SIPS such that $q \in \mathrm{N}$ and $p = r$. The resultant program is

$\mathbf{L}_3$ $p \leftarrow s, \neg r, t, \neg r, q, p;$ $\quad$ $\mathbf{I}_3$ $p \leftarrow s, \neg r\text{–}1_i, t, \neg r\text{–}2_i, label\text{–}q\text{–}3, p;$

$\mathbf{L}_2$ $r \leftarrow q;$ $\quad$ $\mathbf{I}_2$ $r\text{–}1_i \leftarrow q, r\text{–}2_i \leftarrow q;$

$\mathbf{L}_1$ $q \leftarrow q_0;$ $\quad$ $\mathbf{I}_1$ $label\text{–}q\text{–}3 \leftarrow q_0;$

The relevant program is

$\mathbf{L}_1$ $q \leftarrow q_0$ $\quad$ $\mathbf{I}_3$ $p \leftarrow s, \neg r\text{–}1_i, t, \neg r\text{–}2_i, label\text{–}q\text{–}3, p;$

$\mathbf{I}_2$ $r\text{–}1_i \leftarrow q, r\text{–}2_i \leftarrow q;$

$\mathbf{I}_1$ $label\text{–}q\text{–}3 \leftarrow q_0;$

The magic-transformed program is

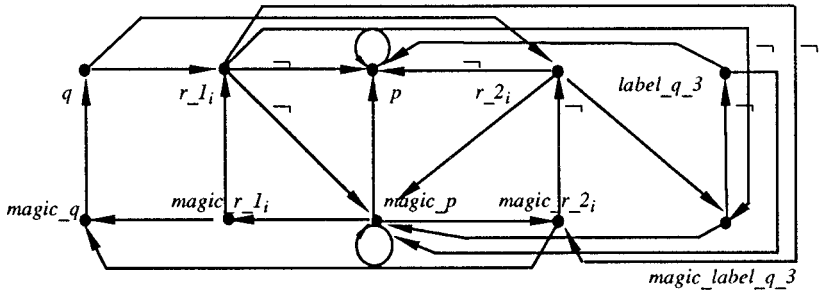

Figure 5. Dependency graph of labeled and transformed program of Example 4.4.

$p \leftarrow magic\text{–}p, s, \neg r\text{–}1_i, t, \neg r\text{–}2_i, label\text{–}q\text{–}3, p,$

$r\text{–}1_i \leftarrow magic\text{–}r\text{–}1_i, q,$

$q \leftarrow magic\text{–}q, q_0,$

$magic\text{–}r\text{–}1_i \leftarrow magic\text{–}p, s,$

$magic\text{–}label\text{–}q\text{–}3 \leftarrow magic\text{–}p, s, \neg r\text{–}1_i, t, \neg r\text{–}2_i,$

$magic\text{–}p \leftarrow magic\text{–}p, s, \neg r\text{–}1_i, t, \neg r\text{–}2_i, label\text{–}q\text{–}3,$

$magic\text{–}q \leftarrow magic\text{–}r\text{–}1_i,$

$r\text{–}2_i \leftarrow magic\text{–}r\text{–}2_i, q,$

$label\text{–}q\text{–}3 \leftarrow magic\text{–}label\text{–}q\text{–}3, q_0,$

$magic\text{–}r\text{–}2_i \leftarrow magic\text{–}p, s, \neg r\text{–}1_i, t,$

$magic\text{–}q \leftarrow magic\text{–}r\text{–}2_i.$

The dependency graph of the relevant predicates is shown in Figure 5 and all negative cycles are removed.

V. CORRECTNESS OF THE LABELING ALGORITHM

In this section, we prove the preserving of stratification of the magic-set-based transformation on a labeled database. For this purpose, the following definition is necessary.

DEFINITION 5.1. *A directed graph is called a culprit cycle if its underlying undirected graph* (*The underlying undirected graph of a directed graph is defined as the undirected graph resulting from ignoring the directions in the edges*) *is a cycle and at least one edge in the graph is negative.*

As an example, consider the graph shown in Figure 6. In the graph, there are two negative edges: from q to p and from t to s. Especially, the graph shown in

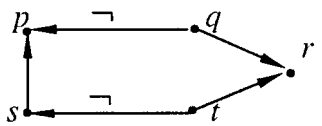

Figure 6. Culprit cycle.

Figure 7. Culprit cycles.

Figure 7(a) is also a culprit cycle since we can take it as the graph shown in Figure 7(b) and there is a negative edge: $p \leftarrow q$ in the graph.

PROPOSITION 5.1. *Let $\mathbf{P}^a$ be a stratified adorned program and $\mathbf{P}^{am}$ the resultant program after applying the algorithm of magic-set transformation to $\mathbf{P}^a$. If $\mathbf{P}^{am}$ is unstratified, then the dependency graph corresponding to $\mathbf{P}^a$ contains necessarily culprit cycles.*

Proof. Suppose that $\mathbf{P}^a$ is stratified but $\mathbf{P}^{am}$ is unstratified. Let p and q be two nodes (not the nodes representing magic predicates) in a negative cycle of $\mathbf{P}^{am}$ and Γ be the negative path from p to q. Without loss of generality, we assume that on Γ there is no node which represents a magic predicate. Thus, Γ is also a negative path in the directed graph corresponding to $\mathbf{P}^a$. Consider the path Γ' in the negative cycle from q to p. Because $\mathbf{P}^a$ is stratified, we know that on Γ' there is at least one node which represents a magic predicate. Let λ_0 be the first node representing a magic predicate, say, *magic–*r_0. Now consider the succeeding node of λ_0, λ_1. In terms of the algorithm of the magic-set transformation, λ_1 is a node which represents either r_0 or another magic predicate. If λ_1 represents a magic predicate, we further examine its succeeding node. In this way, we can finally find a sequence

$$\lambda_0, \lambda_1, \ldots, \lambda_m$$

with λ_i representing *magic–*r_i $(0 \le i < m)$ and λ_m representing r_{m-1}. If r_{m-1} is p, then

$$p = r_{m-1} \rightarrow r_{m-2} \rightarrow \cdots \rightarrow r_1 \rightarrow r_0 \rightarrow q'$$

is a path from p to q' in the dependency graph corresponding to $\mathbf{P}^a$. Therefore, the graph consisting of the path from p to q, the path from q to q', and the path from p to q' constitutes a culprit cycle in the dependency graph corresponding to $\mathbf{P}^a$. The proposition is proved. If r_{m-1} is not p, we consider r_{m-1}'s successor. If this successor is neither p nor the node representing a magic predicate, we further check its successor. In this way, we can find p or another γ_0 which is the first node, after r_{m-1}, representing a magic predicate, say, *magic–*s_0. As above, we can find another sequence $\gamma_0, \gamma_1, \ldots, \gamma_n$ with γ_i representing *magic–*s_i $(0 \le i < n)$ and γ_n representing s_{n-1} such that

$$s_{n-1} \rightarrow \cdots \rightarrow s_0$$

is a path in the dependency graph corresponding to $\mathbf{P}^a$. Because the dependency graph is finite, we can finally find a node t in a finite number of steps such that

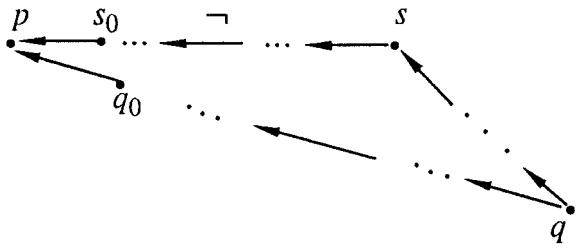

Figure 8. Culprit cycle.

$t = p$. Hence, the graph consisting of the sequence of paths which we have found is a culprit cycle in the dependency graph corresponding to $\mathbf{P}^a$. ■

However, the culprit cycle is not sufficient for unstratification. As a counter example, consider the graph shown in Figure 8. If in the SIPS for the program there is no arc being of the form $\mathrm{N} \rightarrow r$ such that $s_0 \in \mathrm{N}$ and $q_0 = r$, then in the dependency graph corresponding to the transformed version of the program there is no negative cycle which is introduced from the culprit cycle and the newly produced magic rules. The reason for that is no magic rule of the form $magic\text{–}q_0 \leftarrow \cdot \cdot \cdot s_0 \cdot \cdot \cdot$ is derived. Therefore, we lack an edge for a negative cycle.

PROPOSITION 5.2. *For a stratified adorned program* $\mathbf{P}^a$, *if* $\mathbf{P}^{al}$ *is the resultant program after applying the labeling algorithm to* $\mathbf{P}^a$, *and* $\mathbf{P}^{alm}$ *is the resultant program after applying the magic-set transformation to* $\mathbf{P}^{al}$ *then* $\mathbf{P}^{alm}$ *is stratified.*

Proof. From Proposition 5.1, we know that if a transformed program is unstratified, the dependency graph corresponding to the original program contains at least one culprit cycle. Therefore, we need only to prove that the dependency graph corresponding to the labeled version of a stratified program contains no culprit cycle or only those from which no negative cycles can be introduced. After applying the first procedure of the labeling algorithm, the graph as shown in Figure 9(a) will be changed into the graph as shown in Figure 9(b). Therefore, the unstratification caused by the graph will be eliminated.

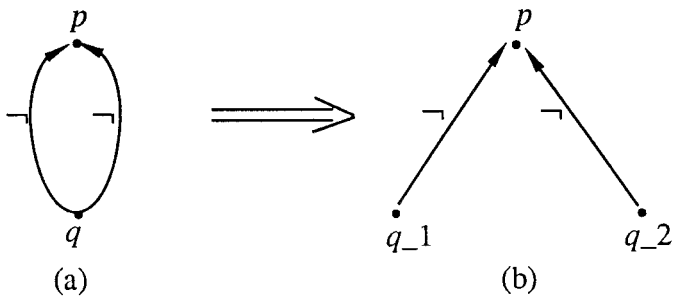

Figure 9. Removing culprit cycles by using *negnumber*.

Figure 10. Culprit cycles.

After applying the second procedure of the labeling algorithm, any culprit cycles like that of Figure 10(a) becomes the graph shown in Figure 10(b). Since the subscript i will be changed with each iteration of the T operation,[17] this subgraph is in effect identical to the graph shown in Figure 11. Thus, the unstratification caused by the graph shown in Figure 10(a) will also be removed.

After applying the third procedure of the labeling algorithm, any culprit cycle like that of Figure 12(a) will be changed into the graph shown in Figure 12(b) if in the SIPS for the program there is an arc being of the form $\mathrm{N} \rightarrow r$ such that $s_0 \in \mathrm{N}$ and $q_0 = r$. Only those culprit cycles, which do not have the associated SIPS, are left unlabeled because they cannot introduce negative cycles at all. By Proposition 5.1, the $\mathbf{P}^{alm}$ is stratified. ■

PROPOSITION 5.3. *Let* $\mathbf{D} = \mathbf{P} \cup \mathbf{F}$ *be a stratified database and* $\mathbf{D}^{alm} = \mathbf{P}^{alm} \cup \mathbf{F}$ *be the resultant database after applying the magic set transformation to the labeled database* $\mathbf{P}^{al}$. *Let* $\mathbf{A}$ *be the set of answers to a query on* $\mathbf{D}^{alm}$. *If* $\mathbf{A}$ *is evaluated using a bottom–up computation, then* $\mathbf{A}$ *is sound and complete with respect to the perfect model.*

Proof. See Appendix. ■

VI. TIME COMPLEXITY

In order to analyze the time complexity of the labeling algorithms, we distinguish two kinds of elementary costs: the cost of labeling a rule and the cost of copying a rule. Since in practice a rule contains only a limited number of literals, both the time spent for labeling a rule and the time for copying a rule

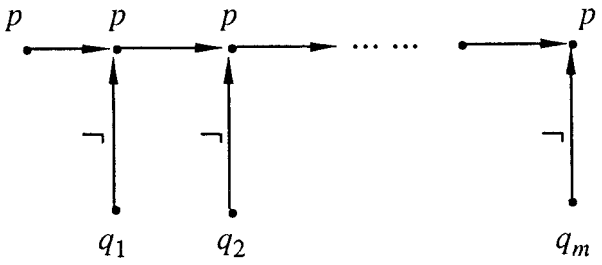

Figure 11. Expansion of graph of Figure 10(b).

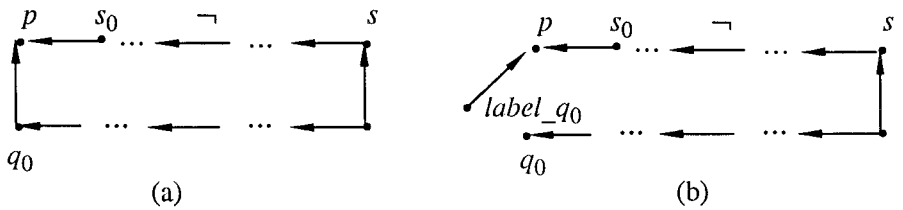

Figure 12. Removing culprit cycles by using *label*.

can be taken as a constant. We denote them by c_{label} and c_{copy}, respectively. In addition, we assume that the rule set contained in the system is partitioned into n strata such that

(1) each stratum $\mathbf{L}_i$, $1 \leq i \leq n$, contains exactly the rules defining a predicate p if p is not recursive, or
(2) $\mathbf{L}_i$ contains exactly the rules defining p and (any) other predicates in the same strongly connected component (in the associated dependency graph) as p if p is recursive; and
(3) $\mathbf{L}_0$ contains all the base facts.

Using the above assumption, we know that the time requirement of *negnumber*() is $k_1 \cdot c_{label} + k_2 . c_{copy}$ for some integers $k_1, k_2 \leq n$, since in each stratum there are only a constant number of negated predicates. For the same reason, the cost of *dynlabel*() is $k \cdot c_{label}$ for some $k \leq n$. In order to show that the (third) algorithm *label*() requires $O(n \cdot e)$ time, where e represents the number of the predicates involved in the rule set, we demonstrate that for each pair of the form (r, p), where p is an unlabeled (positive) body literal and r is a body literal in the same rule, the function *pathConn*(r, p) can be evaluated in linear time. To this end, we consider the underlying undirected graph G' of the corresponding dependency graph G. Then, the computation of *pathConn*(r, p) is reduced to the problem of checking whether there exists an (undirected) path connecting r and p in G'. Obviously, this can be done in $O(e)$ time. Considering that there is only one **for** loop in *label*() and every iteration step has cost $O(e)$ in the worst case, we know that *label*() requires only $O(n \cdot e)$ time.

In contrast, the algorithm proposed in Ref. 14 requires $O(n^2 \cdot e)$ time, because its "positive labeling procedure" consists of two loops. On the one hand, the outer loop will be performed n times. On the other hand, for every iteration of the outer loop (say for the ith iteration), a function *depends*(i, j), $i > j$, which is used to determine whether there exists a path in G from a predicate defined in $\mathbf{L}_j$ to a predicate defined in $\mathbf{L}_i$, will be computed for i strata through the inner loop. In addition, two elementary operations: labeling a rule and copying a rule, will be carried out for these i strata in the worst case. Since the cost of *depends*(i, j) is $O(e)$, the total cost of the algorithm proposed in Ref. 14 is

$$O\left(\sum_{i=1}^{n} i \cdot (c_{label} + c_{copy} + e)\right) = O(n^2 \cdot e).$$

VII. SUMMARY

In this article, a labeling algorithm for stratified databases is presented. The algorithm which is performed prior to the magic-set algorithm can be used to distinguish the context for constructing magic sets. We have shown that the culprit cycles give rise to the unstratification of a database. Based on the analysis, three subprocedures are developed to remove the different kinds of culprit cycles. The *negnumber* procedure numbers the different occurrences of a negative literal in a rule. The *dynlabel* procedure gives each negative body literal a dynamic subscript when it appears in a recursive rule. Finally, the *label* procedure labels each body literal p when there exists a sequence of paths connecting it to a negative body literal $\neg q$ in the same rule, or a sequence of paths with at least one path being negative connecting it to a positive body literal q in the same rule and there is an arc of the form $\mathrm{N} \rightarrow r$ in the SIPS such that $q \in \mathrm{N}$ and $p = r$.

VIII. APPENDIX

Here we prove Proposition 5.3. To this end, we have to first clarify that the labeling algorithm given in Section IV will not change the semantics of the original program. That is, we need to show that if $\mathbf{A}$ is the set of answers to a query on $\mathbf{D} = \mathbf{P} \cup \mathbf{F}$ and $\mathbf{A}'$ is the set of answers to the query on $\mathbf{D}^l = \mathbf{P}^l \cup \mathbf{F}^l$, where $\mathbf{P}^l$ and $\mathbf{F}^l$ are the labeled versions of $\mathbf{P}$ and $\mathbf{F}$, respectively, then we will have $\mathbf{A} = \mathbf{A}'$. For this purpose, we prove five lemmas to confirm this claim. Then we show a sixth lemma which completes the total proof.

Let $\mathbf{F}$ be a set of facts. A binding θ is a set of pairs $\{x_1/t_1, \ldots, x_n/t_n\}$, where $x_1, \ldots, x_n$ are variables, and $t_1, \ldots, t_n$ are elements in $\mathbf{F}$. If p is a predicate with variables $x_1, \ldots, x_n$ and θ a binding, then $p\theta$ denotes the simultaneous replacement of all the variables of p by the corresponding elements in θ. Let r be a rule:

$$q \leftarrow p_1, \ldots, p_m.$$

A binding θ is *applicable* w.r.t. r and the fact set $\mathbf{F}$ if for each body literal p_i,

(1) $p_i\theta \in \mathbf{F}$ if p_i is positive, or
(2) $p_i\theta \notin \mathbf{F}$ if p_i is negative.

The *application of r to* $\mathbf{F}$ denoted $r(\mathbf{F})$, is defined as:

$$r(\mathbf{F}) = \{q\theta | \theta \text{ is applicable w.r.t. } r \text{ and } \mathbf{F} \text{ and } q \text{ is the head of rule } r\}.$$

For a set R of rules $R(\mathbf{F})$, the *application of R to* $\mathbf{F}$, is defined as:

$$R_0(\mathbf{F}) = \mathbf{F},$$

$$R_{i+1}(\mathbf{F}) = \bigcup_{r \in R} r(R_i(\mathbf{F})) \bigcup R_i(\mathbf{F}),$$

$$R(\mathbf{F}) = \bigcup_{i=0}^{\infty} R_i(\mathbf{F}).$$

Interpretation I' is a *subinterpretation* of an interpretation I, denoted $I' \subseteq I$, if I and I' are identical except perhaps in some predicates p such that $I'(p) \subset I(p)$. (For an interpretation I, $I(p)$ denotes the interpretation of p under I.) We say that an interpretation G is a *minimal model of R w.r.t.* a set of facts $\mathbf{F}$ if

- $\mathbf{F} \subseteq G$, and
- G is a model of R and
- There does not exist a model G' of R different from G such that $G \supset G'$.

Note that the above description is identical in spirit to the definition of the perfect model. For the purpose of proofs, however, we put stress on the application of a single rule, which may facilitate the clarification of the main idea of the following lemmas.

LEMMA 1. *Let* $\mathbf{P}$ *be a program* (*i.e., a set of rules*) *with a stratification consisting of a single level and let* $\mathbf{F}$ *be a set of facts. Then* $\mathbf{P}(\mathbf{F})$ *is a minimal model of* $\mathbf{P}$ *w.r.t.* $\mathbf{F}$.

Proof. The proof is identical to Lemma 3.2.2 in Ref. 18. ■

In the remainder of the appendix, we will use $\mathbf{D}_1 = \mathbf{P}_{negn} \cup \mathbf{F}_{negn}$, $\mathbf{D}_2 = \mathbf{P}_{dyn} \cup \mathbf{F}_{dyn}$ and $\mathbf{D}_3 = \mathbf{P}_{lab} \cup \mathbf{F}_{lab}$ to denote the databases obtained by applying the labeling procedures *"negnumber," "dynlabel,"* and *"label"* to $\mathbf{D} = \mathbf{P} \cup \mathbf{F}$, respectively.

LEMMA 2. *Let* $\mathbf{P}$ *be a program with a stratification consisting of a single level and* $\mathbf{F}$ *be the set of facts. Let* Ω_{negn} *be the set of those facts with the newly generated predicate symbols when the labeling procedure "negnumber" is applied to* $\mathbf{P} \cup \mathbf{F}$. *Then we have* $\mathbf{P}(\mathbf{F}) = \mathbf{P}_{negn}(\mathbf{F}_{negn})/\Omega_{negn}$.

Proof. Without loss of generality, assume that only one rule r in $\mathbf{P}$ contains multiple appearance of a negative literal in its body. In terms of the procedure *"negnumber,"* the different occurrence of this negative literal will be numbered. Thus, the proof of the lemma is reduced to the proof of the equation $r(\mathbf{F}) = r_{negn}(\mathbf{F}_{negn})/\Omega_{negn}$, where r_{negn} is the labeled rule obtained by applying the procedure *"negnumber"* to r.

Assume $r(\mathbf{F}) \not\subset r_{negn}(\mathbf{F}_{negn})$. Then there exists at least an element $e \in r(\mathbf{F})$ but $e \notin r_{negn}(\mathbf{F}_{negn})$. Suppose that r is of the form:

$$q \leftarrow p_1, \ldots \neg s, \ldots, \neg s, \ldots, p_m,$$

and there is a binding θ such that $q\theta = e$. Then $p_1\theta, \ldots, p_m\theta \in \mathbf{F}$ and all $s\theta \notin \mathbf{F}$. Let $S \subset \mathbf{F}$ be the set of all s facts in $\mathbf{F}$. In terms of the algorithm *"negnumber,"* r will be transformed to r_{negn}:

$$q \leftarrow p_1, \ldots \neg s_1, \ldots, \neg s_i, \ldots, p_m,$$

and accordingly $\mathbf{F}$ will be augmented to $\mathbf{F}_{negn} = \mathbf{F} \cup \bigcup_{j=1}^{i} S_j$, where each $S_j (j = 1, \ldots, i)$ is a copy of S and each element in S_j has the predicate name s_j. Since all $s\theta \notin \mathbf{F}$ and thus all $s\theta \in S$, we have $s_j \notin S_j (j = 1, \ldots, i)$. Therefore, $s_j \notin \mathbf{F}_{negn}$ $(j = 1, \ldots, i)$. Obviously, $p_1\theta, \ldots, p_m\theta \in \mathbf{F}_{negn}$. Thus, $q\theta = e \in r_{negn}(F_{negn})$. Contradiction. Then, we have $r(\mathbf{F}) \subseteq r_{negn}(\mathbf{F}_{negn})$. Trivially, we have $r(\mathbf{F}) \subseteq r_{negn}(\mathbf{F}_{negn})/\Omega_{negn}$. In a similar way, we can prove $r(\mathbf{F}) \supseteq r_{negn}(\mathbf{F}_{negn})/\Omega_{negn}$. This completes the proof of the lemma. ■

Note that in a practical implementation, we will not label a base literal because no magic rule will be constructed for it. Therefore, in practice, $\neg s$ in the above rule will never be labeled. However, for the theoretical purpose, we prove this situation to provide the basic step for an induction proof.

LEMMA 3. *Let* $\mathbf{P}$ *be a program with a stratification consisting of a single level and* $\mathbf{F}$ *be the set of facts. Let* Ω_{dyn} *be the set of those facts with the newly generated predicate symbols when the labeling algorithms "dynlabel" is applied to* $\mathbf{P} \cup \mathbf{F}$. *Then we have* $\mathbf{P}(\mathbf{F}) = \mathbf{P}_{dyn}(\mathbf{F}_{dyn})/\Omega_{dyn}$.

Proof. The proof is similar to Lemma 2. ■

LEMMA 4. *Let* $\mathbf{P}$ *be a program with a stratification consisting of a single level and* $\mathbf{F}$ *be the set of facts. Let* Ω_{lab} *be the set of those facts with the newly generated predicate symbols when the labeling algorithms "label" is applied to* $\mathbf{P} \cup \mathbf{F}$. *Then we have* $\mathbf{P}(\mathbf{F}) = \mathbf{P}_{lab}(\mathbf{F}_{lab})/\Omega_{lab}$.

Proof. The proof is similar to Lemma 2. ■

Based on the above lemmas, we can demonstrate another important claim that the labeling algorithm will not change the semantics of the original (stratified) databases.

Given a stratified database $\mathbf{D} = \mathbf{P} \cup \mathbf{F}$ with $\mathbf{P}$ being partitioned into levels $L_1, \ldots, L_n$. Let $\mathbf{F}_1 = L_1(\mathbf{F}), \mathbf{F}_2 = L_2(\mathbf{F}_1), \ldots, \mathbf{F}_i = L_2(\mathbf{F}_{i-1}), \ldots, \mathbf{F}_n = L_n(\mathbf{F}_{n-1})$. Then $\mathbf{F}_n$ is the perfect model for $\mathbf{D}$. If the labeled version of $\mathbf{D}$ is denoted as $\mathrm{D}^l = \mathbf{P}^l \cup \mathbf{F}^l$, then $\mathbf{P}^l$ can also be partitioned into n levels $L_1^l = L_1' \cup I_1, \ldots, L_n^l = L_n' \cup I_n$, where each L_i' $(i = 1, \ldots, n)$ is the labeled version of L_i and $\mathbf{F}_n^l$ each I_i is an associated set of newly constructed rules in terms of L_i. Let $\mathbf{F}_1^l = L_i^l(\mathbf{F}^l), \ldots, F_i^l = L_i^l(F_{i-1}^l), \ldots, \mathbf{F}_n^l = L_n^l(\mathbf{F}_{n-1}^l)$. Then $\mathbf{F}_n^l$ is the perfect model for $\mathbf{D}^l$.

LEMMA 5. *Let* Ω_i $(i = 1, \ldots, n)$ *be the set of those facts with the newly generated predicate symbols in* L_i^l. *Then* $\mathbf{F}_n = \mathbf{F}_n^l/\Omega_n$.

Proof. We show by induction that $\mathbf{F}_n = \mathbf{F}_n^l/\Omega_n$.

Basis: In terms of Lemma 2, Lemma 3, and Lemma 4, we have

(1) $L_1(\mathbf{F}) = L_{1.negn}(\mathbf{F}_{negn})/\Omega_{negn}$,
(2) $L_1(\mathbf{F}) = L_{1.dyn}(\mathbf{F}_{dyn})/\Omega_{dyn}$,
(3) $L_1(\mathbf{F}) = L_{1.lab}(\mathbf{F}_{lab})/\Omega_{lab}$.

Replace L_1 and $\mathbf{F}$ in (2) with $L_{1.negn}$ and $\mathbf{F}_{negn}$, respectively. We then have

$$L_{1.negn}(\mathbf{F}_{negn}) = L_{1.negn.dyn}(\mathbf{F}_{negn.dyn})/\Omega_{dyn}.$$

Similarly, by replacing L_1 and $\mathbf{F}$ in (3) with $L_{1.negn.dyn}$ and $\mathbf{F}_{negn.dyn}$, respectively, we will have

$$L_{1.negn.dyn}(\mathbf{F}_{negn.dyn}) = L_{1.negn.dyn.lab}(\mathbf{F}_{negn.dyn.lab})/\Omega_{lab}.$$

Therefore, the following equations hold:

$$L_1(\mathbf{F}) = L_{1.negn}(\mathbf{F}_{negn})/\Omega_{negn}$$

$$L_{1.negn}(\mathbf{F}_{negn})/\Omega_{negn} = L_{1.negn.dyn}(\mathbf{F}_{negn.dyn})/\Omega_{negn}/\Omega_{dyn}$$

$$L_{1.negn.dyn}(\mathbf{F}_{negn.dyn})/\Omega_{negn}/\Omega_{dyn} = L_{1.negn.dyn.lab}(\mathbf{F}_{negn.dyn.lab})/\Omega_{negn}/\Omega_{dyn}/\Omega_{lab}.$$

By the definitions, $(\Omega_{negn} \cup \Omega_{dyn} \cup \Omega_{lab}) = \Omega_1$ and $L_{1.negn.dyn.lab}(F_{negn.dyn.lab}) = L_1^l(\mathbf{F}^l)$. Thus, $L_1(\mathbf{F}) = L_1^l(\mathbf{F}^l)/\Omega_1$ holds.

Induction step: Assume that, for $k \leq i$, $\mathbf{F}_k = \mathbf{F}_k^l/\Omega_k$ holds. We show that $\mathbf{F}_{i+1} = \mathbf{F}_{i+1}^l$. Assume that only one rule r in the $(i + 1)$th level L_{i+1} contains multiple appearance of a negative literals in its body and is of the form:

$$q \leftarrow p_1, \ldots \neg s, \ldots, \neg s, \ldots, p_m,$$

then the proof of $\mathbf{F}_{i+1} = \mathbf{F}_{i+1}^l/\Omega_{i+1}$ is reduced to the proof of $r(\mathbf{F}_i) = r_{negn}(\mathbf{F}_k^l)/\Omega_i$. By the definition of strata, there exists some $j \leq i$ such that s predicate is defined in the jth level. In terms of the algorithm *"negnumber,"* r_{negn} is of the form:

$$q \leftarrow p_1, \ldots, \neg s_1, \ldots, \neg s_h, \ldots, p_m,$$

and $s_1, \ldots, s_h$ are defined in L_j^l. Let $S \subset \mathbf{F}_j$ be the set of s facts and S_u $(u = 1, \ldots, h)$ be the set of s_u facts. Then each $S_u \subset \Omega_j (u = 1, \ldots, h)$. If there exists a binding θ such that $q\theta \in r(\mathbf{F}_i)$, then $p_1\theta, \ldots, p_m\theta \in \mathbf{F}_i$ and all $s\theta \notin \mathbf{F}_i$. By induction hypothesis, $\mathbf{F}_i^l = \mathbf{F}_i \cup \Omega_i$. Then $p_1\theta, \ldots, p_m\theta \in \mathbf{F}_i^l$. Since all $s\theta \notin \mathbf{F}_i$ and thus $s\theta \notin S$, we know that $s_u \notin S_u$ and thus $s_u \notin \Omega_j$. This explains that $q\theta \in r_{negn}(\mathbf{F}_i^l)$. Therefore, $r(\mathbf{F}_i) \subseteq r_{negn}(\mathbf{F}_i^l)$. Trivially, we have $r(\mathbf{F}_i) \subseteq r_{negn}(\mathbf{F}_j^l)/\Omega_i$. In a similar way, we can show that $r_{negn}(\mathbf{F}_i^l)/\Omega_i \subseteq r(\mathbf{F}_i)$. Therefore, $r(\mathbf{F}_i) = r_{negn}(\mathbf{F}_i^l)/\Omega_i$.

If there exists a copy of r in the $(i + 1)$th level L_{i+1}^l:

$$q' \leftarrow p_1, \ldots \neg s, \ldots, \neg s, \ldots, p_m,$$

all facts of the form $q'\theta$ will not be in $\mathbf{F}_{i+1}$ but in $\mathbf{F}^{l}_{i+1}$. Denote all such facts as δ_{i+1}. Then $\Omega_{i+1} = \Omega_i \cup \delta_{i+1}$ and $\mathbf{F}_{i+1} = \mathbf{F}^{l}_{i+1}/\Omega_{i+1}$.

In a similar way, we can prove the other two cases. The correctness of the consecutive applications of *"negnumber," "dynlabel,"* and *"label"* to a stratified database can be derived using the technique for showing the basic step. ■

From Lemma 5, we see that the labeling algorithms do not change the semantics of a stratified database and the labeled version remains stratified. Therefore, to complete the proof of Proposition 5.3, we need only to show that for a query q issued to a stratified database P, the answers obtained by running P is the same as that by running its magic version.

LEMMA 6. *For an adorned stratified database* $\mathbf{D}^a = \mathbf{P}^a \cup \mathbf{F}$ *and adorned query* $?\text{-}q^a(\bar{t})$, *let* $\mathbf{P}^{am}$ *be the program transformed by magic-set transformation, so that* $\mathbf{D}^{am} = \mathrm{P}^{am} \cup \{magic(q^a)\} \cup \mathbf{F}$. *Then* $\mathbf{P}^a$ *and* $\mathbf{P}^{am}$ *are equivalent with respects to* $?\text{-}q^a(\bar{t})$. *That is, an instance* $q^a(\bar{c}) \in \mathbf{F}_i$ *for some i if and only if there exists some j such that* $q^a(\bar{c}) \in L^{am}_j(\mathbf{F}^{m}_{j-1})$, *where* L^{am}_j *represents the jth level of* P^{am} *and* $\mathbf{F}^{am}_{j-1} = L^{am}_{j-1}(L^{am}_{j-2}(\ldots(L^{am}_1(\mathbf{F}))\ldots))$.

Proof. We prove this lemma by induction on the level of q^a. (The level of a query $?\text{-}q^a$ is defined to be the level of those rules defining q^a in the adorned program.)

Basis. Assume $l(q^a) = 1$. In this case, only the first level of $\mathbf{D}^a$ is involved in the computation. For a predicate symbol $p \in L_1$ such that $\neg p(\ldots)$ occurs in the body of some rule of L_1, introduce a new predicate symbol $\overline{p}$. Let

$$\Delta = \{\overline{p}(t_1, t_2, \ldots, t_n) | p \text{ is a } n\text{-any predicate symbol in } L_1, \text{ and } p(t_1, t_2, \ldots, t_n) \notin \mathbf{F}\}.$$

Replacing each $\neg p$ with the corresponding $\overline{p}$, we obtain a positive subdatabase which contains the rules corresponding to the first level of $\mathbf{P}^a$ and all facts. Then the proof is identical to Proposition 4.2 of Ref. 3.

Induction step. Suppose that for some n, for all $i < n$, and for any query q^a of level i, the lemma holds. We prove that an instance $q^a(\bar{c})$ (of $q^a(\bar{t})$) $\in \mathbf{F}_n$ if and only if there exists some integer u such that $q^a(\bar{c}) \in L^{am}_u(\mathbf{F}^{am}_{u-1})$. Since $q^a(\bar{t}) \in L_n$ and $n > 1$, there must be a rule r in L_n with head q and body

$$p_1, \ldots \neg s_1, \ldots \neg s_h, \ldots, p_k,$$

and a binding θ such that $q\theta = q^a(\bar{c})$. In terms of the magic-set transformation, there is a rule of the following form in L^{am}_u for some u:

(1) $q^a \leftarrow magic\text{–}q^a, p_1, \ldots \neg s_1, \ldots \neg s_h, \ldots, p_k.$

For $q^a(\bar{c})$ to be in $L^{am}_u(\mathbf{F}^{am}_{u-1})$,

(2) $q^a\theta \leftarrow magic\text{–}q^a\theta, p_1\theta, \ldots \neg s_1\theta, \ldots \neg s_h\theta, \ldots, p_k\theta$

must be a ground instance of the above rule. Since $magic\text{–}q^a\theta$ is generated by the magic-set transformation, all we need to show is that all $s_g\theta$ ($g = 1, \ldots, h$) are not in $\mathbf{F}^{am}_{u-1}$ and thus not in $L^{am}_u(\mathbf{F}^{am}_{u-1})$ and all $p_j(j = 1, \ldots, k)$ are in $L^{am}_u(\mathbf{F}^{am}_{u-1})$. First, we prove that $s_g\theta$ are not in $\mathbf{F}^{am}_{u-1}$. By the definition of levels, each s_g is of level i such that $i < n - 1$. If $s_g\theta$ is in $\mathbf{F}^{am}_{u-1}$ and thus $s_g\theta$ is in F_{i+1}, by the induction hypothesis, we know that $s_g\theta$ is in F_{i+1} and thus it is also in F_{n-1}. From this, we have $q^a(\bar{c}) \notin L_n$. Contradiction. Second, we prove that all $p_j(j = 1, \ldots, k)$ are in $L^{am}_u(\mathbf{F}^{am}_{u-1})$. To this end, we require to prove that magic(p_j) ($j = 1, \ldots, k$) is also in $L^{am}_u(\mathbf{F}^{am}_{u-1})$, if p_j is a derived predicate. Consider the derived body literals in the total order induced by the SIPS associated with r. Let p_1 be the first derived literal in the body. There is an arc $N \rightarrow p_1$ from the SIPS, in which N consists of base literals and possibly q. So there is a magic rule with head $magic\text{–}p_1$ and body consisting of the base literal in N and possibly $magic(q^a)$. Since the facts defining the base predicate in $\mathbf{D}^a$ are included in $\mathbf{F}^{am}_{u-1}$ and $magic(q^a)$ is generated by the magic transformation, then $magic(p_1)\theta$ is in $L^{am}_u(\mathbf{F}^{am}_{u-1})$. Consider the next derived literal p_2 in the SIPS-induced order. The head of the SIPS arc N entering p_2 may include p_1, and the corresponding magic rule would include p_1 in the body. Since $p_1\theta$ is in $L^{am}_u(\mathbf{F}^{am}_{u-1})$, using a similar argument to the above we can show that $magic(p_2)\theta$ is in $L^{am}_u(\mathbf{F}^{am}_{u-1})$. Repeating this for all such derived body literals, we see that each $magic(p_j)\theta$ ($j = 1, \ldots, k$) is in $L^{am}_u(\mathbf{F}^{am}_{u-1})$. If p_1 is defined in the vth level of P^a and $\nu < n$, then by induction hypothesis $p_1\theta$ is in $L^{am}_t(\mathbf{F}^{am}_{t-1})$ for some $t < u$, and thus $p_1\theta$ is in $L^{am}_u(\mathbf{F}^{am}_{u-1})$. If p_1 is defined in the nth level of P^a with a rule of the form:

$$(3)\quad p_1 \leftarrow p_{11}, \ldots, p_{1w},$$

there exists some binding γ such that

$$(4)\quad p_1\gamma \leftarrow p_{11}\gamma, \ldots, p_{1w}\gamma,$$

is an instance of (3) and $p_1\gamma = p_1\theta$. (This is due to the fact that $q^a\theta \leftarrow p_1\theta, \ldots \neg s_1\theta, \ldots \neg s_h\theta, \ldots, p_k\theta$ is an instance of r in L_n.) From this, we know that $magic(p_1)\gamma = magic(p_1)\theta$. In terms of the magic set transformation, the rule:

$$(5)\quad p_1 \leftarrow magic(p_1), p_{11}, \ldots, p_{1w},$$

is in L^{am}_d for some $d \leq u$. Due to the equation $magic(p_1)\gamma = magic(p_1)\theta$, $p_1\gamma \leftarrow magic(p_1)\gamma$, $p_{11}\gamma, \ldots, p_{1w}\gamma$ is an instance of (5). Therefore, $p_1\theta = p_1\gamma$ is in $L^{am}_u(\mathbf{F}^{am}_{u-1})$. In the same way, we can show that each $p_j\theta$ ($j = 1, \ldots, k$) is in $L^{am}_u(\mathbf{F}^{am}_{u-1})$ and thus $q^a(\bar{c})$ is in $L^{am}_u(\mathbf{F}^{am}_{u-1})$.

In order to complete the induction step, we have to show that if $q^a(\bar{c}) \in L^{am}_u(\mathbf{F}^{am}_{u-1})$ for some integer u, then $q^a(\bar{c}) \in \mathbf{F}_n$ for some n. The proof in this direction simply follows from the fact that rules in L^{am}_u, which are derived from rules in L_n, are more restrictive in that an extra positive body literal (magic predicate) is inserted into the body. ■

PROPOSITION 5.4. *Let* $\mathbf{D} = \mathbf{P} \cup \mathbf{F}$ *be a stratified database and* $\mathbf{D}^{alm} = \mathbf{P}^{alm} \cup \mathbf{F}$ *be the resultant database after applying the magic set transformation to the labeled database* $\mathbf{P}^{a}$. *Let* $\mathbf{A}$ *be the set of answers to a query on* $\mathbf{D}^{alm}$. *If* $\mathbf{A}$ *is evaluated using a bottom–up computation, then* $\mathbf{A}$ *is sound and complete with respect to the perfect model.*

Proof. By Lemma 5, we know that a labeling algorithm will not change the semantics of the original program. By Lemma 6, we know that a transformed labeled program is equivalent to the original labeled program in the sense that the set of answers (to the corresponding adorned query) obtained by running the former is the same as that by running the latter. This completes the proof. ■

References

1. F. Bancilhon, and R. Ramakrishnan, ''Performance evaluation of data intensive logic programs,'' in *Foundations of Deductive Databases and Logic Programming,* J. Minker, Ed., Morgan Kaufman, Los Altos, CA, 1988, pp. 439–518.
2. F. Bancihon, D. Maier, Y. Sagiv, and J. Ullman, ''Magic sets and other strange ways to implement logic programs,'' in *Proceedings of the 5th ACM SIGMOD–SIGACT Symposium on Principles of Database Systems,* Washington, 1986, pp. 1–15.
3. C. Beeri, and R. Ramakrishnan, ''On the power of magic,'' in *Proceedings of the 6th ACM SIGACT–SIGART Symposium on Principles of Database Systems,* San Diego, CA, 1987, pp. 269–283.
4. Y. Chen, ''Processing of recursive rules in knowledge-based systems—Algorithms for handling recursive rules and negative information and performance measurements,'' Ph.D. Thesis, Computer Science Department, University of Kaiserslautern, Germany, Feb. 1995.
5. Y. Chen, ''On the Bottom–up evaluation of recursive queries,'' *Int. J. Intell. Syst.,* 1995, accepted.
6. D.B. Kemp, K. Ramamohanarao, I. Balbin, and K. Meenakshi, ''Propagating constraints in recursive deductive databases,'' in *Proceedings of the North American Conference on Logic Programming,* Cleveland, OH, 1989.
7. K. Morris, J.F. Naughton, Y. Saraiya, J.D. Ullman, and A. van Gelder, ''Yawn! (Yet another window on nail!),'' *IEEE Data Engrg.*, **10**(4), 28–43 (Dec. 1987).
8. D. Sacca and C. Zaniolo, ''Implementation of a simple class of logic queries for databases,'' in *Proceedings of the 5th ACM Symposium on Principles of Database, Systems,* WA, 1986, pp. 16–23.
9. J. Rohmer, R. Lescoeur, and J.-M. Kerisit, ''The Alexander method—A technique for the proceeding of recursive axioms in deductive databases,'' *New Generation Comput.,* **4**(3), 273–286 (1986).
10. K.R. Apt, H.A. Blair, and A. Walker, ''Towards a theory of declarative knowledge,'' in *Foundations of Deductive Databases and Logic Programming,* J. Minker, Ed., Morgan Kaufman, Los Altos, CA, 1988, pp. 89–148.
11. R. Topor and E. Sonenberg, ''On domain independent database,'' in *Foundations of Deductive Databases and Logic Programming,* J. Minker, Ed., Morgan Kaufman, Los Altos, CA, 1988, 217–240.
12. K. Ross, ''On negation in HILOG,'' *J. Logic Programming,* **18,** 27–53 (1994).
13. C. Zaniolo, N. Arni, and K. Ong, ''Negation and aggregates in recursive rules: The LDL++ approach,'' in *Proc. of the 3th Int. Conf. on Deductive and Object-Oriented Databases,* Phoenix, AZ, Dec. 1993.
14. L. Balbin, G.S. Port, K. Ramamohannarao, and K. Meenakshi, ''Efficient bottom–up

computation of queries on stratified databases,'' *J. Logic Programming,* 295–344 (Nov. 1991).
15. J.W. Lloyd, E.A. Sonenberg, and R.W. Topor, *Integrity Constraint Checking in Stratified Databases, Technical Report 86/5,* Dept. of Computer Science, Univ. of Melbourne, 1986.
16. A. Chandra and D. Harel, ''Horn clause queries and generalization,'' *J. Logic Programming,* **2**(1), 1–15 (1985).
17. M. van Emden and R. Kowalski, ''The semantics of predicate logic as a programming language,'' *J. Assoc. Comput. Mach.,* **23**(4), 733–742 (1976).
18. C. Beeri, S. Naqvi, R. Ramakrishnan, O. Shmuelo, and S. Tsur, ''Sets and negation in a logic database language (LDL1),'' in *Proc. of the 6th ACM SIGACT–SIGMOD–SIGART Symposium on Principles of Database Systems,* San Diego, CA, 1987, pp. 21–37.