CLASSIFICATION OF RECURSIVE FORMULAS IN DEDUCTIVE DATABASES

Cheong Youn, Lawrence J. Henschen
Department of EE/CS
Northwestern University, Evanston
Illinois 60201 U.S.A

Jiawei Han
School of Computing Science
Simon Fraser University
Burnaby, British Columbia
V5A 1S6 CANADA

ABSTRACT

In this paper, we present results on the classification of linear recursive formulas in deductive databases and apply those results to the compilation and optimization of recursive queries. We also introduce compiled formulas and query evaluation plans for a representative query for each of these classes.

To explain general recursive formulas, we use a graph model that shows the connectivity between variables. The connectivity between variables is the most critical part in processing recursive formulas. We demonstrate that based on such a graph model all the linear recursive formulas can be classified into several classes and each class shares some common characteristics in compilation and query processing. The compiled formulas and the corresponding query evaluation plans can be derived based on the study of the compilation of each class.

1 INTRODUCTION

Recursion is one of the most discussed techniques in deductive database systems. Classifying recursive formulas is known to be a hard problem. Therefore researchers have considered certain specific patterns of recursive formulas that are easily recognizable and compilable [Han 85a][Ioan 85]. In this paper, we will use a graph model as a tool to represent recursive formulas. Our goal is to show a general and uniform planning mechanism for each of several important classes of linear recursive formulas which can map an arbitrary query of that class to a compiled formula. This will illustrate the power and utility of the graph approach.

At query evaluation, we will consistently use the evaluation principle that join operations will be performed only after selection operations to optimize data retrievals. If we can't apply selection or join operations any further, we will retrieve the exit relation first if the exit relation is not evaluated. Generally, in this situation, answers can be derived by a Cartesian product operation or existance checking.

In section 3, we introduce a graph model that can be used to explain recursive formulas. The I-graph will be used to expand the recursive predicate on successive iterations. In section 4, we will define the terms used in this paper. We also overview the classification of recursive formulas. In section 5, formulas with one-directional cycles will be discussed. Formulas with bounded cycles will be considered in section 6. Formulas with unbounded cycles (independent cycles with non-zero weight) will be discussed in section 7. Components with no non-trivial cycle are mentioned in section 8. Dependent cycles are discussed in section 9. Mixed formulas are discussed in section 10. Further studies and conclusions are in section 11.

2 THE GRAPH MODEL

We consider function-free Horn clauses with only one occurrence of the recursive predicate in the antecedent and with no occurrence of equality. We also do not allow constants in the statement and do not allow a variable to appear more than once under the recursive predicate.

We also assume that there is only one recursive rule (single recursion) which has one or more non-recursive rules (exit rules). The exit rule has the form $P :- E$. Since these rules play a role in the compiled form and not in the graph analysis, we will use E as a generic exit expression and not bother to write the exit rule in the examples.

Suppose we are given a recursive statement F of the form $P(x,...) :- A(u,v) \wedge ...P(y,...)$.... We will associate a labeled, weighted, hybrid graph $G = (V, E_u, E_d, W, \mathcal{L})$ to F. The graph construction was originally introduced by Ioannidis [Ioan 85], and from now we will call this graph an **I-graph**.

Each variable (V) is a vertex. Variables like u, v occuring in a non-recursive predicate $A(u,v)$ are connected

by an undirected edge (E_u). For each pair of variables in corresponding positions of the recursive predicate P in the consequent and the antecedent (e.g. x and y), there is a directed edge (E_d) from x to y. There is a weight (W) for each edge. The directed edge (x $\longrightarrow$ y) has weight 1. We assume that there is an implicit reverse directed edge (y $\longrightarrow$ x), and it has weight -1. Each undirected edge (u — v) has weight 0. There is a label ($\mathcal{L}$) attached to each edge. An undirected edge has a label Q if two nodes of the undirected edge are in the non-recursive predicate Q. Each directed edge is labeled with the recursive predicate.

Definition : The **weight** of a path (cycle) in the graph is defined as the sum of the weights of the edges along the path (cycle). Regarding undirected edges, they can be traversed in both directions. Traversing a directed edge in the opposite direction of the arrow is the same as traversing the implicit reverse directed edge and contributes -1 to the weight.

Example 1.

$$(s1a)\ P(x,y) \ :-\ A(x,z) \wedge P(z,y)$$
$$(s1b)\ P(x,y,z) \ :-\ A(x,y) \wedge P(u,z,v) \wedge B(u,v)$$

The corresponding I-graphs are in Figure 1(a) and 1(b) respectively. We do not write down the label P for directed edges because there is only one recursive predicate for each formula and all directed edges have the same label P.

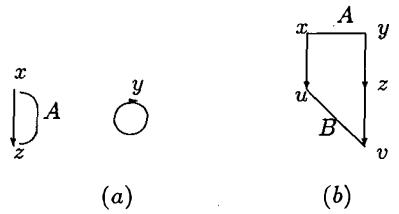


Figure 1

Definition : Let's consider a graph for the k-th expansion of a formula F. The **k-th resolution graph**, G_k = (V, E_u, E_d, W, $\mathcal{L}$) of F is defined recursively in the following manner.

- The I-graph of F is the first resolution graph.
- The k-th resolution graph, G_k ($k \geq 2$) of F is obtained from the (k-1)st resolution graph, G_{k-1}, by the following process.
  1. Form a k-th I-graph by renumbering variables in the original formula and then unifying with the (k-1)st expansion of F.
  2. Append the k-th I-graph to the (k-1)st resolution graph using common variables.

The k-th resolution graph retains all the arrows from the (k-1)st I-graph. Further, the k-th resolution graph is formed directly from the (k-1)st resolution graph without the need to actually form a resolvent. These retained directed edges give a better picture of the derivation.

Example 2.

$$(s2a)\ P(x,y) \ :-\ A(x,z) \wedge P(z,u) \wedge B(u,y)$$

The I-graph is shown in Figure 2(a). If we renumber the variables, we have:

$$(s2b)\ P(x_1,y_1) \ :-\ A(x_1,z_1) \wedge P(z_1,u_1) \wedge B(u_1,y_1)$$

By unification of $P(x_1,y_1)$ in (s2b) with P(z,u) in (s2a), we obtain:

$$(s2b')\ P(z,u) \ :-\ A(z,z_1) \wedge P(z_1,u_1) \wedge B(u_1,u)$$

The 2nd expansion of (s2a) is (shown in Figure 2(c)):

$$(s2c)\ P(x,y) \ :-\ A(x,z) \wedge A(z,z_1) \wedge P(z_1,u_1) \wedge B(u_1,u) \wedge B(u,y)$$

The second I-graph for (s2b') is in Figure 2(b). The second resolution graph G_2 in Figure 2(c) can be drawn by appending Figure 2(b) to 2(a). In Figure 2(c), the weight from x to z_1 is two. That means, in the second expansion of (s2a), x appears under the recursive predicate P in the consequent and z_1 appears in the corresponding position of the recursive predicate in the antecedent. We have Figure 2(d) by considering (s2c) as a formula by itself as opposed to a second resolution graph.

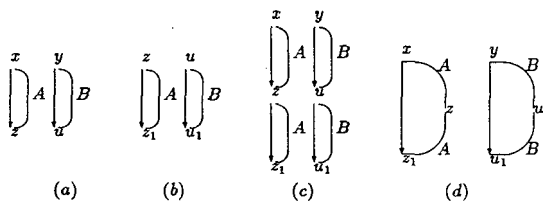


Figure 2

This graph model is a powerful tool for explaining and formalizing recursive formulas. In the next section, we will introduce classes of recursive formulas based on this graph model, and will show compiled formulas and query evaluation plans for a representative query for each of these classes.

3 CLASSIFICATION OF RECURSIVE FORMULAS

Definition : The **dimension (D)** of a recursive formula is the number of variables in the recursive predicate. If there are n variables in the recursive predicate, we call the formula an n-D recursive statement.

Definition : A variable in the recursive formula F (after k-th expansion) is a **determined variable** for a given query if the value of the variable is given in the query

or derivable from a query constant by selection and join operations over only the non-recursive predicates in the (k-th resolution) graph. If x is a determined variable and L(...x..y..) is a non-recursive predicate, then y is also a determined variable [Hens 84].

Definition : Two variables v_1 and v_2 are **connected** if there is a path consisting of all undirected edges.

Definition : A **non-trivial component** (cycle) is a component (cycle) with at least one directed edge. Otherwise, the component (cycle) is **trivial**.

Definition : A non-trivial cycle is **independent** if it is not connected to any other non-trivial cycles nor to any other directed edges. Otherwise, the cycle is **dependent**.

Remark : We can compress several undirected edges into one edge in many cases. For example,

$$P(x,y) :- A(x,u) \wedge B(x,z) \wedge C(z,u) \wedge P(u,y)$$

can be simplified to

$$P(x,y) :- ABC(x,u) \wedge P(u,y)$$

and the formula has two independent cycles. For more details and examples, see [Youn 88].

Definition : An independent cycle is **one-directional** if all the directed edges along the cycle have the same direction. Otherwise, the cycle is **multi-directional**.

Definition : A one-directional cycle is **rotational** if there is at least one undirected edge as a part of the non-trivial cycle. Otherwise, the cycle is **permutational**.

Definition : A one-directional cycle is a **unit cycle** if the weight of the cycle is 1. A formula is a unit-cycle formula if there are only disjoint unit cycles in the corresponding I-graph.

We will restrict the formulas so that any variable appearing in the consequent also appears in the antecedent. Such formulas are called **range restricted** [Gall 84]. In the I-graph, if a variable used as a tail of a directed edge is not used as a head of any other directed edge nor is connected to undirected edges, the formula is not a range restricted formula.

We can classify recursive formulas as follows:

(A) One-directional cycles

- (A1) Unit, rotational cycles
- (A2) Unit, permutational cycles
- (A3) Non-unit, rotational cycles
- (A4) Non-unit, permutational cycles
- (A5) Disjoint combination of different Ai's

(B) Bounded cycles

(C) Unbounded cycles

(D) No non-trivial cycles

(E) Dependent cycles

(F) Mixed cycles: Disjoint combination of different classes

Each class will be discussed in the following sections.

4 ONE–DIRECTIONAL CYCLES

In this section we view recursive formulas from two differnt approaches. One is from the graph model we described before (syntactic view) and the other one is from the information passing and query evaluation point of view (semantic view).

4.1 UNIT CYCLES

There are two differnt kinds of unit cycles, one is the class of unit, rotational cycles (class A1) and the other one is the class of unit, permutational cycles (class A2). A unit, rotational cycle is a unit cycle with at least one undirected edge and by recursive expansions, new variables will be generated for the recursive predicate. A unit, permutational cycle is a self directed loop and by recursive expansions, no new variables will be generated for the recursive predicate.

Definition : A recursive formula is **strongly stable** if the determined variables of the recursive predicate in the consequent and in the antecedent occur in the same positions for any query.

Theorem 1: A recursive formula is strongly stable if and only if there are only disjoint unit cycles in the corresponding I-graph.

pf : ($\longleftarrow$) A unit cycle can be a self directed loop (permutational) or a cycle with undirected edges (rotational). Recall, such a cycle has only one directed edge, and, since it is a cycle, must therefore have at least one undirected edge or be a self loop. In both cases, if a variable in the consequent is determined, then the variable in the same position in the antecedent will be determined, and no variables in other positions in the antecedent will be determined because all cycles are disjoint. By induction on the number of expansions, we can easily see that stability will be preserved for arbitrary numbers of expansions.

($\longrightarrow$) Suppose the graph is not stable because of a uniform cycle of length two, say $P(x,y) :- A(x,z) \wedge P(y,z)$. A query in which only x is determined gives a determined variable z in a different position in the antecedent. A similar non-satisfactory query form can be found if the cycle is not one-directional or fails the stability condition in any other way. Thus, if the determined variables of the recursive predicate in the consequent and in the antecedent occur in the same position on arbitrary queries, then there are n disjoint connections and each connection is made between variables in the same position of the recursive predicate in the consequent and in the antecedent. Therefore, there are n disjoint unit cycles. □

Thus we have equivalent syntactic and semantic **characterizations** for strongly stable formulas.

Remark : Note that this definition of strongly stable is stronger than that given in [Hens 84]. Here, the condition on determined variables holds for **all** query forms. From now on, we call a strongly stable formula simply a "stable formula".

Example 3.

$(s3)\ P(x,y,z)\ :-\ A(x,u) \wedge B(y,v) \wedge P(u,v,w) \wedge C(w,z)$

The corresponding I-graph shows that there are three disjoint unit cycles. Therefore, (s3) is a stable formula. The compiled formula and the evaluation plan for the query P(a,b,Z) are the following respectively:

$$\bigcup_{k=1}^{\infty} \quad A^k - \begin{array}{c} B^k \\ / \\ E \\ \backslash \\ C^k \end{array} , \qquad \bigcup_{k=1}^{\infty} \quad \begin{array}{c} \sigma A^k \\ \backslash \\ E - C^k \\ / \\ \sigma B^k \end{array}$$

A branch such as $\begin{array}{c} \sigma A^k \\ > \\ \sigma B^k \end{array}$ means that A^k and B^k are evaluated independently (we use "-" for the join operation because of the difficulty to use the symbol "$\bowtie$"), and the results are combined with E. We can find evaluation plans for other possible queries, e.g. P(X,b,c) and P(X,a,Z) in a symmetric way.

Query evaluation can be done in many different ways, but we are dealing on the higher level logical form so that the global query plan can be optimized.

For formulas of this class, the compiled formulas are easily obtained and query evaluation plans for all possible queries are also easily found. For more details and examples, see [Youn 88].

4.2 TRANSFORMATIONS OF NONUNIT CYCLES INTO UNIT CYCLES

Classes A3 and A4 consist of formulas with non-unit, one-directional cycles, either **rotational** or **permutational**. We call these cycles "transformable to stable cycles".

If a formula is not stable itself, but can be transformed to an equivalent set of formulas which does represent stable recursion, the techniques of section 5.1 can still be applied. To that end, we show general methods by which (A3) and (A4) formulas can be so transformed.

For the moment, we assume that there is only one component in the corresponding I-graph for a given formula. Formulas with 2 or more components in the I-graph will be discussed later.

Theorem 2: If there is an independent, one-directional cycle of weight n in the I-graph for an n-D recursive formula, F, then

1. The formula becomes stable after each n expansions.
2. The formula can be transformed into an equivalent stable formula with multiple exits by unfolding exactly n times.

Pf (1) : If two directed edges share a variable, we can assume there is a non-recursive predicate "EQUAL" between the shared variable (this is only for the theoretical development). If there is a trivial cycle or if there is more than one undirected edge shared by the same variables, we can collapse them into a single undirected edge. Therefore, without loss of generality, we can assume that there is exactly one undirected edge (non-recursive predicate) between directed edges in the cycle of weight n. Let's call the variables in the consequent $x_1, x_2, ...x_n$ (by the traversal order of the cycle) and the corresponding variables in the antecedent $y_1, y_2, ...y_n$. In the first expansion, x_n is connected to y_1 and no other y. For the following expansion, new variables $z_1, z_2, ...z_n$ will be produced and x_n is connected with z_2 and no other z. By induction on the number of expansions, we can easily find that after n expansions, x_n will be connected to the variable in same position and will not be connected to the variables in any other positions.

pf (2) : From property (1), we find the cyclic behaviour of the formula. Generate the first (n-1) expansions of F and replace the recursive predicate in the antecedents by the exit relation and leave the n-th expansion of F as a new recursive formula. The new recursive formula with n exit relations is stable and produces the same answers as the original formulas. It is, in fact, logically equivalent to the original set. □

We now know that a formula, F, with an independent cycle of weight n become a stable formula after n expansions, and we can consider that the n-th resolution graph of F has n disjoint unit cycles. Sample rules are mentioned in the following sections.

4.3 NON-UNIT, ROTATIONAL CYCLES

From the theorem 2, we know that a formula with a non-unit, rotational cycle can be transformed to an equivalent unit-cycle formula.

Example 4.

$$(s4a)\ P(x_1,x_2,x_3)\ :-\ A(x_1,y_3) \wedge B(x_2,y_1) \wedge C(y_2,x_3) \wedge P(y_1,y_2,y_3)$$
$$(s4b)\ P(x_1,x_2,x_3)\ :-\ E(x_1,x_2,x_3)$$

There is an independent, one-directional cycle of weight 3 in the I-graph. The 2nd and 3rd expanded formulas are the following:

$(s4c)\ P(x_1,x_2,x_3)\ :-\ A(x_1,y_3) \wedge C(z2,y_3) \wedge B(x_2,y_1)$
$\wedge\ A(y_1,z_3)\ \wedge\ C(y_2,x_3)\ \wedge\ B(y_2,z_1)\ \wedge\ P(z_1,z_2,z_3)$

$(s4d)\ P(x_1,x_2,x_3)\ :-\ A(x_1,y_3) \wedge C(z2,y_3) \wedge B(z2,u_1)$
$\wedge\ B(x_2,y_1)\ \wedge\ A(y_1,z_3)\ \wedge\ C(u_2,z_3)\ \wedge\ C(y_2,x_3)$
$\wedge\ B(y_2,z_1)\ \wedge\ A(z_1,u_3)\ \wedge\ P(u_1,u_2,u_3)$

For the transformation to the equivalent stable formula, we need two more exit relations (s4a') and (s4c'). These are found by replacing the recursive predicates in the antecedent of (s4a) and (s4c) with the exit relation E. With (s4d) as a new recursive formula and (s4b), (s4a') and (s4c') as exit relations, we have an equivalent stable formula that produces the same results as (s4a) and (s4b). The compiled formula is:

$$\cup_{k=1}^{\infty}\ (ACB)^k - [\,E \cup A - E \cup (AC) - E\,] - \begin{matrix} B \\ / \\ C \end{matrix}\ \begin{matrix} (AB) \\ / \\ (BC) \end{matrix}\ \begin{matrix} (CAB)^k \\ / \\ \backslash \\ (ABC)^k \end{matrix}$$

and the evaluation plan for the query P(a,b,Z) is the following:

$$\cup_{k=1}^{\infty}\ \begin{matrix} \sigma(ACB)^k \\ \backslash \\ \sigma(BAC)^k \end{matrix}\ \begin{matrix} A \\ \backslash \\ B \end{matrix}\ [\,E \cup E - C \cup E - BC\,] - \begin{matrix} (AC) \\ \backslash \\ (BA) \end{matrix}\ - (ABC)^k$$

In the compiled formula, the notation "AB" indicates the connectivity of two relations and doesn't indicate any particular order. In a query evaluation plan, however, order of the predicates is the actual order to be used in the evaluation process.

4.4 NON-UNIT, PERMUTATIONAL CYCLES

From the theorem 2, we can easily see that a formula with a non-unit, permutational cycle can be transformed to an equivalent unit-cycle formula.

Example 5.

$(s5)\ P(x,y,z)\ :-\ P(y,z,x)$

The correponding I-graph shows that there is a cycle of weight three. The formula can be transformed into a stable formula. But there is no non-recursive relation involved in the expansions, and after three expansions, the formula can not produce any new values (or tuples). We call such formulas bounded [Ioan 85]. Bounded formulas will not produce any new tuples (values) after certain expansions regardless of the contents of the database. The above formula has no new variables in the antecedent; all the variables are from the consequent. This is called a "permutational pattern" to distinguish it from rotational formulas.

Theorem 3: A formula with a permutational cycle or disjoint combinations of such cycles is also permutational.

pf : A disjoint combination of permutational cycles is also permutational because there are no new variables in the recursive predicates. As soon as the formula becomes stable, it is in fact in its original form. □

Example 6.

$(s6)\ P(x,y,z,u,v,w)\ :-\ P(z,y,u,x,w,v)$

Statement (s6) is a permutational formula, and there are three permutational cycles in the I-graph with weights 3, 1, and 2 respectively. We can easily see that the formula becomes stable (comes back to the original formula) after 6 expansions and will not produce new tuples by further expansions, therefore further expansions are meaningless.

4.5 GENERAL PROPERTIES

Theorem 4: A formula constructed by a disjoint combination of one or more one-directional cycles can be transformed to an equivalent unit-cycle formula.

pf : If there are k disjoint independent, one-directional cycles $G_1, G_2, \ldots G_k$ from a recursive formula, and the weight of the cycle for each G_i is c_i, the formula can be transformed to an equivalent stable formula by unfolding exactly L times, where L is the least common multiple of $c_1, c_2, \ldots c_k$. □

Example 7.

$(s7)\ P(x,y,z,u,w,s,v)\ :-\ A(x,t) \wedge P(t,z,y,w,s,r,v)$
$\wedge\ B(u,r)$

The corresponding I-graph has 4 one-directional disjoint cycles of weights 1, 2, 3, and 1 respectively. We can easily see that the formula becomes stable after 6 expansions.

We should point out that there are other recursive formulas which may stabilize for *particular* queries but are not strongly stable and are not equivalent to a strongly stable formula.

5 BOUNDED CYCLES

In this section and the following section, we will discuss formulas that have multi-directional cycles. As we discussed above, to be transformed into a stable one, a formula should have disjoint unit cycles after some expansions. As we can see by the following theorem, multi-directional cycles can't be transformed into unit cycles. Therefore, formulas with multi-directional cycles can't be transformed into stable formulas.

Theorem 5: An independent, multi-directional cycle can not be transformed to an equivalent unit-cycle formula.

pf: An independent, multi-directional cycle has at least one (possibly compressed) undirected edge whose two nodes

each are used as the tail of directed edges. By the expansion of the resolution graph, the two nodes can never be split, and the resolution graph can't be expressed as disjoint unit cycles (refer to the first theorem of section 5). Therefore the formula can't be transformed to a stable formula. □

Corollary 1: An independent cycle can be transformed to an equivalent unit-cycle formula (or is unit-cycle formula) if and only if it is one–directional.

pf: From the theorem 5. □

Definition : An independent cycle is called a **bounded cycle** if the weight of the cycle is 0.

Definition : The **rank** of a recursive formula is defined to be the smallest i such that the (i+1)st expansion and all succeeding expansions do not produce any tuple not found in the first i expansions.

Definition : A recursive formula is called **bounded** if and only if there exits a finite upper bound on its rank independent of the contents of the relations involved in the formula [Ioan 85].

Ioannidis's Theorem : Let F be a recursive formula with no permutational patterns. Then F is bounded if and only if the corresponding I-graph contains no cycle of non-zero weight. In that case a tight upper bound on the rank of the recursive formula is given by the maximum weight of any path in the I-graph [Ioan 85].

Example 8.

$(s8)\ P(x,y,z,u) :- A(x,y) \wedge B(y_1,u) \wedge C(z_1,u_1) \wedge P(z,y_1,z_1,u_1)$

We call bounded formulas, "pseudo recursion". If a recursive formula is bounded, there is an equivalent finite set of nonrecursive formulas, e.g. (s8) has the upper bound 2 (from Figure 3), and will be expressed as nonrecursive formula(s) by replacing relation P in the antecedent by the exit relation E: Therfore, the formula (s8) can be expressed by equivalent nonrecursive formulas:

$(s8a')\ P(x,y,z,u) :- A(x,y) \wedge B(y_1,u) \wedge C(z_1,u_1) \wedge E(z,y_1,z_1,u_1)$

$(s8b')\ P(x,y,z,u) :- A(x,y) \wedge B(y_1,u) \wedge C(z_1,u_1) \wedge A(z,y_1) \wedge B(y_2,u_1) \wedge C(z_2,u_2) \wedge E(z_1,y_2,z_2,u_2)$

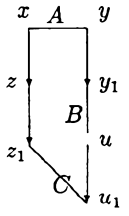


Figure 3

Theorem 6: A formula constructed by a disjoint combination of bounded cycles is bounded.

pf : All the disjoint components will be expanded independently, and all the components are bounded. Therefore, the formula is bounded. □

Bounded formulas have been considered by many researchers for optimization [Naug 86][Ioan 85] because after certain expansions we do not need to generate further expansions of the formula nor do further query processing. General solutions providing the upper bound can be found in [Ioan 85][Naug 86].

6 UNBOUNDED CYCLES

Definition : An independent, multi–directional cycle of non–zero weight is called an **unbounded cycle.**

Query evaluation plans for unbounded cycles are more complicated than the previous cases.

Example 9.

$(s9)\ P(x,y,z) :- A(x,y) \wedge B(u,v) \wedge P(u,z,v)$

The 1st and 2nd resolution graphs are in Figure 4. Let's consider a query of the form P(d,v,v), where v stands for non-determined position (variables) and d for a determined position (constant), either from the query or from recursive predicates after some expansions. The evaluation process will be the following:

The first expansion (Figure 4(a)): The value of x is given and we can apply the selection operation to the relation A and derive values of y. There is no more selection or join operation possible. If there is no information available, we will select the exit relation, E, (this strategy is the conventional technique) and derive all the tuples of E. Then we can apply the join operation with the relation B to find values of z. The answer will be the Cartesian product (symbol "X" will be used) of values of y and z. This evaluation step can be expressed as (σA) X $(E \bowtie B)$.

The second expansion (Figure 4(b)): The value of x is given and we can derive values of y as in the first expansion. There is no more selection and join operation possible. We will derive all the tuples of exit relation E. Then we can apply the join operation with B to find common tuples (for variables u_1 and v_1), and apply the join with B and A successively to find values of z. Evaluation steps will be (σA) X [$(E \bowtie B)BA$].

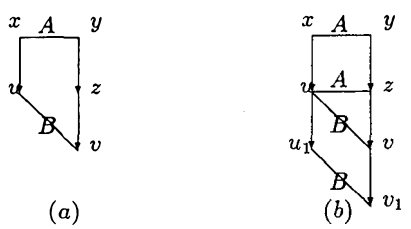


Figure 4

Therefore, the query evaluation plan for P(d,v,v) can be expressed as:

$$\sigma E, \quad (\sigma A) \text{ X } (\cup_{k=0}^{\infty} [\ (E \bowtie B)(BA)^k\ ])$$

For a query of the form P(v,v,d), we have the following evaluation plan from the resolution graphs:

$$\sigma E, \quad (\exists\ \cup_{k=0}^{\infty} [\ (AB)^k\ (E \bowtie B)\ ])\ A$$

The symbol "$\exists$" is used for the existance checking for the immediately following expression. This means if there is any tuple (not empty) that satisfies the expression in the ($\exists$...), then all the tuples in the relation A will be answers.

A general method for the unbounded formulas is not known at this time. But if we use the resolution graph, we can easily derive compiled formulas (or query evaluation plans) for individual cases.

7 NO NON–TRIVIAL CYCLES

In this section, we will consider components with no non-trivial cycles.

Theorem 7: A non-trivial component with no non-trivial cycle can't be transformed to a strongly stable formula.

pf : (1) Suppose that there is only one directed edge. The head and tail of the directed edge are not connected to each other by any undirected edge(s). By induction on the number of expansions, the head and tail will never be connected.

(2) If there is more than one directed edge, there are two possibilities, (a) directed edges are one directional, (b) directed edges are multi-directional. In case (a), there is a leftmost (or rightmost) node used as a tail of a directed edge. By induction on the number of expansions, the node will never be connected to any other nodes, and the formula can't be transformed to a stable one. In case (b), we can prove as in the previous theorem. □

Corollary 2: A component with no nontrivial cycle is bounded.

pf : There is no cycle of non-zero weight in this component. From Ioannidis's theorem, the component is bounded. □

Example 10.

$$(s10)\ P(x,y)\ :-\ B(y)\ \wedge\ C(x,y_1)\ \wedge\ P(x_1,y_1)$$

There are no nontrivial cycles in the I-graph for the formula (s10). If a query P(X,Y) is given, we can derive all the tuples from $E(x,y)$, $B(y) \wedge C(x,y_1) \wedge E(x_1,y_1)$ by first expansion, and finally $B(y) \wedge C(x,y_1) \wedge B(y_1) \wedge C(x_1,y_2) \wedge E(x_2,y_2)$. Further expansions will not produce any new tuples and the upper bound is 2 [Ioan 85].

8 DEPENDENT CYCLES

Although there are no complete general techniques developed so far for this class, we will show by examples that the I-graph and the resolution graph can be very useful in planning query processing for this kind of formula.

Theorem 8: A formula with a dependent cycle can't be transformed to a unit-cycle formula.

pf : (CASE 1) We already proved that an independent, multi-directional cycle can't be transformed to a stable formula. Furthermore, any multi-directional cycle can't be transfomed to a stable one. Indeed, any multi-directional cycle has at least one undirected edge (with two nodes, e.g. x_1 and x_2), and x_1 and x_2 are used as the tails of directed edges. To be stable, the nodes x_1 and x_2 should be disconnected, but will never be disconnected because later resolution graphs are obtained by appending the I-graph to the head of directed edges, and nodes x_1 and x_2 will never be split. Therefore a dependent cycle with a multi-directional cycle as a subcycle or a dependent cycle with an undirected edge, whose two nodes are used as the tail of the directed edges is not transformed to a stable one.

(CASE 2) The same claim can be applied to a component with one undirected edge whose two nodes are used as the heads of the directed edges. On the next expansion, the two nodes of the undirected edge become the tails of the two directed edges, and from (CASE 1) we can easily find that the component can't be transformed to stable.

(CASE 3) Let's consider the dependent, one-directional cycle. Assume that there is an extra undirected edge that makes the cycle dependent. We can assume that the node (call the node x) of the extra undirected edge is used as a tail of one directed edge, and the other node (called y) is used as a head of one directed edge because we already mentioned the other possible cases. If the value of x is given on a query (only the value of one variable is known), then two variables are determined in the next expansion. On further expansions, there is no possibility that only one variable is determined. So the formula can not be transformed to stable. □

Corollary 3: A formula can be transformed to an equivalent unit-cycle formula if and only if it has only one-directional cycles.

pf: From the previous theorems. □

Remark : We have shown that the "**semantic**" and the "**syntactic**" definitions of strongly stable and transformable to strongly stable formulas are equivalent. Therefore, only one-directional cycles can be transformed to stable formulas.

Example 11.

$$(s11)\ P(x,y)\ :-\ A(x,x_1) \wedge B(y,y_1) \wedge C(x_1,y_1) \wedge P(x_1,y_1)$$

The corresponding 1st and 2nd resolution graphs for (s11) are in Figure 5. If a query form P(d,v) is given, from the second expansion, all the variables in the recursive

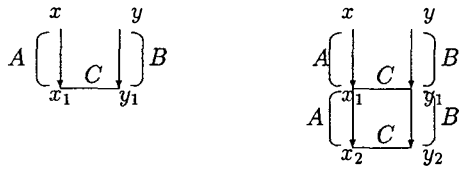
predicate are determined, and there is no non-determined part. The query evaluation plan for P(d,v) is the following. The symbol { } is used to express the parallel evaluation of relations.

$$\sigma E,\ \sigma A - C - B - E,\ \sigma A - C - B - \begin{Bmatrix} A \\ \{\ \} \\ B \end{Bmatrix} - C - E, \ldots$$

The simplified formula is

$$\sigma\ E,\quad \sigma\ A - C - B - E,$$

$$\cup_{k=1}^{\infty}\ \sigma\ A - C - B - [\ \begin{Bmatrix} A \\ \{\ \} \\ B \end{Bmatrix} - C\ ]^k - C - E$$



(a) (b)

Figure 5

9 MIXED CYCLES

In this class, we will consider disjoint combinations of different classes.

Theorem 9: A formula constructed by disjoint combination of one-directional cycles (class A) and components from other classes can not be transformed to a unit-cycle formula.

pf : All the disjoint components will be expanded independently (not connected),therefore we can see the property easily. □

The general method for this class is not known at this time. Further studies should be done on the formulas in this class. Using the resolution graph, we can derive compiled formulas or query evaluation plan for individual cases.

From the classification, we can find the following properties of recursive formulas. Recall the various cases.

Theorem 10: If a formula P is constructed by a disjoint combination of { A2, A4}, then the tight upper bound of P is the least common multiple (L) -1 of the weight of all the cycles.

pf : After L expansions, the formula comes back to the original form. Therefore, the formula is bounded and the upper bound is L-1. □

Theorem 11: A formula constructed by a disjoint combination of {A2, A4, B, D} is bounded.

pf : Each component will be expanded independently, and all the components are bounded, therefore the formula is bounded. □

Remark : A formula constructed by a disjoint combination of bounded components will be bounded.

Theorem 12: The above classification is complete.

pf : Our analysis is done on each component. There are four possibilities on each component, (1) no non-trivial cycle, (2) one-directional cycles, (3) multidirectional cycles, (4) dependent cycles. There is no overlap between these classes. A Disjoint combination of cycles in the same class will be in the same class. A disjoint combination of the different classes will be in the mixed type class. Therefore the classification is complete with no overlap. □

Example 14.

$$(s12)\ P(x,y,z)\ :-\ A(x,u) \wedge B(y,v) \wedge C(u,v) \wedge D(w,z) \wedge P(u,v,w)$$

The corresponding resolution graphs for the 1st and 2nd expansions are in Figure 6. The formula is a disjoint combination of classes (D) and (A1). For a query P(d,v,v), we have the following.

incoming query : P(d,v,v)
first expansion : P(d,d,v)
second expansion : P(d,d,v)

For the remaining expansions, we have P(d,d,v). We can easily predict that pattern from the graph. If x is known, on the next expansion, u, v will be determined because there is a connection between x and u, v. Following the first expansion, we have the same pattern (the length of the cycle period is 1). For all other possible queries for the formula (s12), we can easily plan the query processing.

The evaluation plan for the query P(d,v,v) will be the following :

$$\sigma\ E,\quad \cup_{k=0}^{\infty}\ \sigma\ A - C - B - [\ \begin{Bmatrix} A \\ \{\ \} \\ B \end{Bmatrix} - C\ ]^k - E - D^{k+1}$$

At the first expansion, the evaluation is performed $A - C - B$ to derive y and then, $E - D$ to derive z. From the second expansion, variables in the first and second position are determined and the query evaluation plan is similar to that of a two dimensional stable formula. Formula (s12) becomes stable after certain expansions depending on the query forms. For a query form P(d,v,v), the formula becomes stable from second expansion, and for a query P(v,v,d), the formula is stable from the beginning. In general, the formula becomes stable after some expansion, but different from one query form to another query form. Formula (s12) can not be transformed to a strongly stable formula. Our definition of strongly stable emphasized the query independent property. But after certain expansions, the query evaluation of (s12) can be considered similar to that of stable formulas.

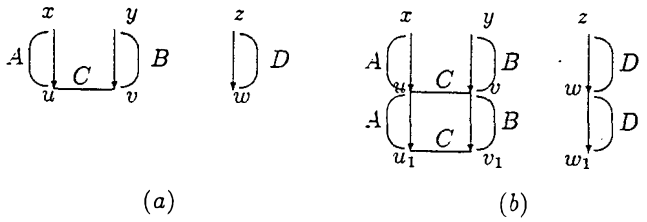


Figure 6

Discovering a general compilation method for non-transformable recursive formulas will be a difficult task. But we can apply the graph model and easily do planning based on the connectivity of relations.

10 FURTHER STUDIES

We presented a classification of linear recursive formulas based on a graph model. Our analysis of the compilation and query processing of some interesting classes discloses that the formulas in each class share common characteristics in their compiled formulas and query processing plans. Therefore, it shows that the I-graph model is a valuable tool in the classification of recursive formulas and deriving planning mechanisms for recursive queries.

Our study is confined to recursions consisting of single linear recursive formula. There are many other interesting studies on the processing of more general recursions, such as magic set method, the Alexander method, and Query-Sub-Query approach. However, we believe that the graph method provides a powerful tool in the study of behaviour of linear recursions in complex variable patterns. Once such behaviour is well understood, the compilation and further optimization can be explored in depth, which in turn will have strong impact on the further development of efficient recursive query processing methods.

Although the scope of this paper is limited to some classes of linear recursive rules, we believe that further exploration on unbounded cycles, dependent cycles, and mixed cycles will produce interesting results. Moreover, the exploration of the application of the I-graph model to the compilation of the multiple linear recursive rules, non-linear resursive rules and other kinds of recursion is another future research topic.

References

[Banc 86] F. Bancilhon and P. Ramakrishnan, "An Amateure's Introduction to Recursive Query Processing Strategies", *Proceedings of ACM-SIGMOD Conference on Management of Data*, 1986.

[Banc 86] F. Bancilhon, D. Maier, Y. Sagiv and J. Ullman, "Magic Sets and Other Strange Ways to Implement Logic Programs", *Proceedings of the 5th ACM SIGMOD-SIGART Symposium on Principles of Database Systems*, 1986.

[Gall 84] H. Gallaire, J.Minker and J. Nicolas, "Logic and Databases: A Deductive Approach", *Computing Surveys, Vol.1 16, No.2*, June 1984.

[Han 85a] J. Han and H. Lu, "Some Performance Results on Recursive Query Processing in Relational Database Systems", *Proceedings of the 2nd international Conference on Data Engineering"*, IEEE Computer Society, 1985.

[Han 85b] J. Han, *"Pattern-Based and Knowledge-Directed Query Compilation for Recursive Databases"*, Ph.D Dissertation, University of Wisconsin, Madison, 1985.

[Han 87] J. Han and L. Henschen, "Handling Redundency in Recursive Query Processing", *Proceedings of the ACM SIGMOD Conference on Management of Data*, 1987.

[Hens 84] L.J. Henschen and S. Naqvi, "On Compiling Queries in Recursive First-Order Databases", *JACM 31(1)*, 1984, pages 47-85.

[Ioan 85] Y. Ioannidis, "A Time Bound on the Materialization of Some Recursively Defined Views", *Proceedings of the 11th International Conference on Very Large Databases*, Stockholm, Sweden, Aug. 1985.

[Naug 86] J. Naughton, "Data Independent Recursion in Deductive Databases", *Proceedings of the ACM SIGACT-SIGMOD Symposium on Principle of Database Systems*, 1986.

[Reit 78] R. Reiter, "Deductive Question-Answering on Relational Databases", in *Logic and Databases*, edited by H. Galaire and J. Minker, Plenum Press, New York, N.Y., 1978, pages 147-177.

[Reit 84] R. Reiter, "Toward a Logical Reconstruction of Relational Database Theory", In *On Conceptual Modelling*, edited by M.L. Brondie, J. Mylopoulos and J.W. Schmit, Springer-Verlag, New York, 1984, pages 163-189.

[Shap 80] S. Shapiro and D. Mckay, "Inference with Recursive rules", *Proceedings of AAAI*, 1980, pages 151-153.

[Ullm 85] J. Ullman, "Implementation of Logical Query Languages for Databases", *ACM Transactions on Database Systems 10(3)*, 1985.

[Youn 87] C. Youn, "A New Graph Model on Recursive Formulas in Deductive Databases", *Proceedings of First Annual Meeting of the Midwest Artificial Intelligence and Cognitive Science Society*, Chicago, IL, April 1987.

[Youn 88] C. Youn *"A Classification of Recursive Formulas in Deductive Databases"*, in preparation for Ph.D. dissertation, Department of EE/CS, Northwestern University, May, 1988.