5-1988

On Purely Exponential Logic Queries

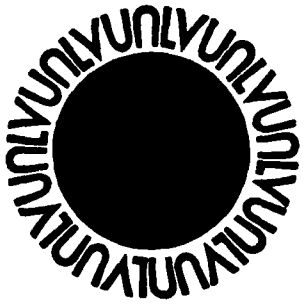
Kazem Taghva
University of Nevada, Las Vegas, kazem.taghva@unlv.edu

Tian-Zheng Wu
New Mexico Tech

ARO 24960.17-MA

(2)

Department of Computer Science and Electrical Engineering



University of Nevada, Las Vegas
Las Vegas, Nevada 89154

(2)

On Purely Exponential Logic Queries

by

*Kazem Taghva**
Department of Computer Science
University of Nevada
Las Vegas, NV 89154

and

Tian-Zheng Wu
Department of Computer Science
New Mexico Tech
Socorro, NM 87801

May 1988

* The research of this author was supported in part by U.S. Army Research Office under grant #DAAL03-87-G-0004.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

MASTER COPY - FOR REPRODUCTION PURPOSES

REPORT DOCUMENTATION PAGE

AD-A201259

1a. REPORT SECURITY CLASSIFICATION: Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S): ARO 24960.19-MA
6a. NAME OF PERFORMING ORGANIZATION: Univ. of Nevada, Las Vegas; 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION: U. S. Army Research Office
6c. ADDRESS (City, State, and ZIP Code): Dept. of Computer Science & Elec. Engr. 4505 Maryland Parkway Las Vegas, NV 89154	7b. ADDRESS (City, State, and ZIP Code): P. O. Box 12211 Research Triangle Park, NC 27709-2211
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: U. S. Army Research Office; 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: DAAL03-87-G-0004
8c. ADDRESS (City, State, and ZIP Code): P. O. Box 12211 Research Triangle Park, NC 27709-2211	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification): On Purely Exponential Logic Queries

12. PERSONAL AUTHOR(S): Kazem Taghva and Tian-Zheng Wu

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM ____ TO ____	May 1988	16

16. SUPPLEMENTARY NOTATION: The view, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number): Logic Progamming, Relational Data Model, Dependency, Recursive Queries.

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Purely exponential queries are logic programs of the form:

$$S(X) \leftarrow S(\bar{X}_1), S(\bar{X}_2), \ldots, S(\bar{X}_n).$$

$$S(\bar{X}) \leftarrow A(\bar{X}).$$

where S and A are predicates of arity m. In this paper, we provide a syntatic condition under which these queries can be rewritten as linear queries. As an application of this result, we give a new proof for Guessarian's theorem [4] on converting binary chained exponential queries to linear queries. Moreover, an infinite chain of progressively weaker template dependencies is constructed via expansion of the logic program for transitive closure of a relation R. This natural chain yields another proof for the result of R. Fagin, et al [3].

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: ☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION: Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Abstract

Purely exponential queries are logic programs of the form :

$$S(\bar{\mathbf{X}}) \leftarrow S(\bar{\mathbf{X}}_1),\ S(\bar{\mathbf{X}}_2),\ \ldots,\ S(\bar{\mathbf{X}}_n).$$

$$S(\bar{\mathbf{X}}) \leftarrow A(\bar{\mathbf{X}}).$$

where S and A are predicates of arity m. In this paper, we provide a syntactic condition under which these queries can be rewritten as linear queries. As an application of this result, we give a new proof for Guessarian's theorem [4] on converting binary chained exponential queries to linear queries. Moreover, an infinite chain of progressively weaker template dependencies is constructed via expansion of the logic program for transitive closure of a relation R. This natural chain yields another proof for the result of R. Fagin, et al[3].

1. Introduction

The recursive nature of logic programs has long been the subject of optimisation techniques [2][7]. Recently, the database community has taken interest in extending the expressive power of relational algebra by augmenting it with function-free Horn style logic queries. This extension has led to various optimisation techniques[2][7]. It seems, almost invariably, these techniques are most efficient in the processing of linear recursive queries. For this reason, there is a genuine interest in identifying those classes of non-linear recursive queries which can be rewritten as linear queries. Among these classes are binary chained purely exponential queries[4] and doubly recursive queries[9].

In this paper, we provide a sufficient condition for a subclass of purely exponential queries to be equivalent to linear queries. This subclass properly contains the class of binary chained purely exponential queries. In addition, as a by product of this work we construct a very natural progressively weaker infinite chain of *template dependencies*[3].

2. Preliminaries

A *literal* is an expression of the form $P(X_1, X_2, \ldots, X_m)$, where P is a predicate symbol of arity m and X_i's are either variables or constants. A *rule* is a formula of the form

$$P(X_1, X_2, \ldots, X_m) \; :- \; Q_1(Y_{1,1}, Y_{1,2}, \ldots, Y_{1,m}),$$
$$Q_2(Y_{2,1}, Y_{2,2}, \ldots, Y_{2,m}), \; \ldots, \; Q_n(Y_{n,1}, Y_{n,2}, \ldots, Y_{n,m}),$$

where $P(X_1, X_2, \ldots, X_m)$ and $Q_i(Y_{i,1}, Y_{i,2}, \ldots, Y_{i,m})$ for $i = 1, 2, \ldots, n$ are literals. A rule is *recursive* if $P = Q_i$ for some i. A recursive rule is *linear* if there is exactly one occurrence of P on the right-hand side. We will call $P(X_1, X_2, \ldots, X_m)$ the *head* of the rule and $Q_1(Y_{1,1}, Y_{1,2}, \ldots, Y_{1,m}), Q_2(Y_{2,1}, Y_{2,2}, \ldots, Y_{2,m}), \ldots, Q_n(Y_{n,1}, Y_{n,2}, \ldots, Y_{n,m})$ the body of the rule. The variables appearing in the head of the rule are called *distinguished* variables, and all other variables are called *nondistinguished*. We will also make the assumption that all predicates except P are denoting base relations (i.e., relations explicitly stored in the database).

Intuitively, a rule states that the tuple $(X_1, X_2, \ldots, X_m)$ is in P, if tuples $(Y_{1,1}, Y_{1,2}, \ldots, Y_{1,m}), (Y_{2,1}, Y_{2,2}, \ldots, Y_{2,m}), \ldots$, and $(Y_{n,1}, Y_{n,2}, \ldots, Y_{n,m})$ are in $Q_1, Q_2, \ldots$ and Q_n

respectively. A *logic program* is a finite sequence of rules to be interpreted as a finite disjunct of rules.

Example 2.1 Let R denote a binary base relation, then the following program represents the transitive closure of R:

$$(I) \quad \begin{cases} r_1 : T(X,Y) :- T(X,Z),\ T(Z,Y). \\ r_2 : T(X,Y) :- R(X,Y). \end{cases}$$

Rule r_2 states that every tuple in R is also in T, while rule r_2 states that all other tuples in T should be obtained by the composition of tuples in T.

Although logic programs in general are evaluated via resolution methods, logic queries in database settings are evaluated by fixed point techniques due to the restricted form of these queries. We will demonstrate this technique using the program given in example 2.1. Assume that the base relation representing R is $\{(a,b),(b,d),(c,f)\}$.

Step 1: $T = \emptyset$, $R = \{(a,b),(b,c),(c,f)\}$.

Step 2: Place the current values of T and R from step 1 into the bodies of rules r_1 and r_2. As the current value of T is $\emptyset$, rule r_1 will not produce any tuple, while rule r_2 will add the current value of R to T, i.e.,

$$T = \{(a,b),(b,c),(c,f)\},$$

$$R = \{(a,b),(b,c),(c,f)\}.$$

Step 3: Place the current values of T and R from step 2 into the bodies of rules r_1 and r_2. Rule r_2 will not add any new tuples to T. Rule r_1 will add tuples (a,c) and (b,f) to T, as these two tuples are the result of taking join over the attribute Z and then projecting over the attributes X and Y. Hence,

$$T = \{(a,b),(b,c),(c,f),(a,c),(b,f)\},$$

$$R = \{(a,b),(b,c),(c,f)\}.$$

Step 4: Place the current values of T and R from step 3 into the bodies of rules r_1 and r_2. Again rule r_1 will not add any new tuples to T, while rule r_2 will add the tuple (a,f) to T. Hence,

$$T = \{(a,b),(b,c),(c,f),(a,c),(b,f),(a,f)\},$$

$$R = \{(a,b),(b,c),(c,f)\}.$$

Step 5: Place the current values of T and R from step 4 into the bodies of rules r_1 and r_2. At this time, neither rule produces any new tuples. The procedure terminates and the transitive closure of R is taken to be the last value of T from step 4.

The above procedure will always terminate due to the existence of the least fixed point[1].

The *purely exponential* programs are defined to be programs of the form[4]:

$$\left.\begin{array}{l} S(X_1, X_2, \ldots, X_m) \ :- \ S(Y_{1,1}, Y_{1,2}, \ldots, Y_{1,m}), \cdots, S(Y_{n,1}, Y_{n,2}, \ldots, Y_{n,m}). \\ S(X_1, X_2, \ldots, X_m) \ :- \ A(X_1, X_2, \ldots, X_m). \end{array}\right\} \qquad (*)$$

The class of purely exponential programs contains a large number of natural examples such as transitive closure and Cartesian products. Moreover, the recursive rules in purely exponential programs are essentially template dependencies as defined in [3][8]. We say that a purely exponential program is *binary chained* if it has the following form:

$$S(X,Y) \ :- \ S(X,Z_1), \ S(Z_1,Z_2), \ \cdots, \ S(Z_{n-1},Y).$$

$$S(X,Y) \ :- \ A(X,Y).$$

Two programs P_1 and P_2 defining predicates S_1 and S_2 using the same set of base relations are *equivalent* if both P_1 and P_2 produce the same relation for S_1 and S_2 for all values of the base relations. For example, the program given in example 2.1 is equivalent to the following program:

$$(II) \qquad \begin{cases} r_1 : \ T(X,Y) \ :- \ R(X,Z), \ T(Z,Y). \\ r_2 : \ T(X,Y) \ :- \ R(X,Y). \end{cases}$$

One approach in establishing equivalence is to expand the recursive predicates S_1 and S_2 into disjunct of conjunctions of base predicates. Since programs such as transitive closure are not first order properties [1], in general the disjunct is infinite. The following infinite sequence defines the transitive closure of a relation R (commas are to be interpreted as *and*):

$$R(X,Y)$$

$$R(X,Z_0), r(Z_0,Y)$$

$$R(X,Z_0), R(Z_0,Z_1), R(Z_1,Y)$$

$$R(X,Z_0), R(Z_0,Z_1), R(Z_1,Z_2), R(Z_2,Y)$$

.

.

.

The first expression is obtained from rule r_2 of program (II), the second expression is obtained from rule r_1 of program (II) with the nondistinguished variable Z being renamed Z_0 and $T(Z,Y)$ rewritten as $R(Z,Y)$, the third expression is obtained from rule r_1 by rewriting $T(Z_0,Y)$ as $R(Z_0,Z_1),T(Z_1,Y)$ using rule r_1 recursively and then rewriting $T(Z_1,Y)$ as $R(Z_1,Y)$ using rule r_2, and so on. It is important that we rename the nondistinguished variables as we expand. Intuitively, the first expression represents the base relation R, the second represents all tuples obtained *from R via one application of transitivity*, the third represents all tuples obtained from R via two applications of transitivity, and so on. Then the transitive closure is defined to be the union of all relations defined by these expressions. If there is more than one occurrence of the recursive predicate, we must systematically expand all occurrences of the predicate by means of a selector function[6]. In the terminology of first order logic the above infinite sequence can be written as:

$$\{XY \mid R(X,Y)\}$$

$$\{XY \mid (\exists Z_0)\,(R(X,Z_0) \wedge R(Z_0,Y))\}$$

$$\{XY \mid (\exists Z_0)(\exists Z_1)\,(R(X,Z_0) \wedge R(Z_0,Z_1) \wedge R(Z_1,Y))\}$$

$$\{XY \mid (\exists Z_0)(\exists Z_1)(\exists Z_2)\,(R(X,Z_0) \wedge R(Z_0,Z_1) \wedge R(Z_1,Z_2) \wedge R(Z_2,Y))\}$$

.

.

.

Finally, we call a mapping ρ between variables of expressions e_1 and e_2 a *containment map*, *if ρ maps each distinguished variable to itself and for every literal $P(X_1,X_2,\ldots,X_m)$ of e_1,*

then $P(\rho(X_1), \rho(X_2), \ldots, \rho(X_m))$ is a literal of e_2. The next lemma states the relationship between a containment map and relations defined by expressions e_1 and e_2.

Lemma 2.1 If ρ is a containment map from e_1 to e_2, then the relation defined by e_2 is a subset of the relation defined by e_1.

3. Main Result

In this section, we will establish a sufficient condition to rewrite purely exponential queries of the form (*) into the following linear queries:

$$\left.\begin{array}{lll} S(X_1, X_2, \ldots, X_m) & :- & A(Y_{1,1}, Y_{1,2}, \ldots, Y_{1,m}), \cdots, A(Y_{(n-1),1}, Y_{(n-1),2}, \ldots, Y_{(n-1),m}), \\ & & S(Y_{n,1}, Y_{n,2}, \ldots, Y_{n,m}). \\ S(X_1, X_2, \ldots, X_m) & :- & A(X_1, X_2, \ldots, X_m). \end{array}\right\} \quad (**)$$

In order to motivate the readers, we first provide an example of a purely exponential query of the form (*) which is not equivalent to a linear query of the form (**).

Example 3.1 Consider the following two programs:

$$P_1: \quad \begin{cases} S(X_1, X_2, X_3) & :- & S(W, X_2, U),\ S(X_1, U, V),\ S(T, V, X_3). \\ S(X_1, X_2, X_3) & :- & A(X_1, X_2, X_3). \end{cases}$$

$$P_2: \quad \begin{cases} S(X_1, X_2, X_3) & :- & A(W, X_2, U),\ A(X_1, U, V),\ S(T, V, X_3). \\ S(X_1, X_2, X_3) & :- & A(X_1, X_2, X_3). \end{cases}$$

Let $A = \{(6,0,1), (7,1,2), (6,2,3), (8,3,4), (7,4,5)\}$. In order to see that P_1 and P_2 are not equivalent, we can expand both programs. We observe that the following expression

$$A(W_1, X_2, U_1)A(W, U_1, V_1)A(T_1, V_1, U)A(X_1, U, V)A(T, V, X_3)$$

can be obtained by first applying the recursive rule of P_1 at the leftmost occurrence of S and then replacing all occurrences of S by A. Now, by assigning $(6,0,1)$, $(7,1,2)$, $(6,2,3)$, $(8,3,4)$ and

$(7,4,5)$ to $A(W_1, X_2, U_1)$, $A(W, U_1, V_1)$, $A(T_1, V_1, U)$, $A(X_1, U, V)$ and $A(T, V, X_3)$ respectively, we generate the new tuple $(8,0,5)$ via program P_1. It is easy to see that the following is the infinite expansion of P_2:

$$
\begin{aligned}
e_1 &= A(W, X_2, U), A(X_1, U, V), A(T, V, X_3), \\
e_2 &= A(W, X_2, U), A(X_1, U, V), A(W_1, V, U_1), A(T, U_1, V_1), A(T_1, V_1, X_3), \\
e_3 &= A(W, X_2, U), A(X_1, U, V), A(W_1, V, U_1), A(T, U_1, V_1), \\
&\quad A(W_2, V_1, U_2), A(T_1, U_2, V_2), A(T_2, V_2, X_3), \\
&\vdots
\end{aligned}
$$

We note that the first two literals of e_1 is a prefix of e_2 and the first four literals of e_2 is a prefix of e_3, and so on. We observe that e_1 and e_2 do not produce the tuple $(8,0,5)$. Because of the way the variables are chained in e_k for $k \geq 3$, the first five literals should be assigned to $(6,0,1)$, $(7,1,2)$, $(6,2,3)$, $(8,3,4)$ and $(7,4,5)$ respectively. It can be seen that the tuple $(8,0,5)$ will never be generated.

In the database setting, in addition to the fact that function symbols are not allowed, there is another restriction which is known as the *safety rule*[2]. The safety requires that any distinguished variable should also occur somewhere in the body of the rule. Both function symbols and unsafe formulas cause nonterminating computations[2].

Definition 3.1 Let $P(X_1, X_2, \ldots, X_m)$ and $Q(Y_1, Y_2, \ldots, Y_m)$ be two literals, a *connection graph* from P to Q is a directed graph on m nodes for which there is an edge from node i to node j iff $x_i = y_j$. A purely exponential query is *uniformly connected* iff every two adjacent literals in the body of the rule have the same connection graph.

Example 3.2 Consider the following program:

$$
\begin{aligned}
S(X_1, X_2, X_3) &:- S(U, X_2, X_3), S(V, U, U), S(X_1, V, V). \\
S(X_1, X_2, X_3) &:- a(X_1, X_2, X_3).
\end{aligned}
$$

The connection graph from $S(U, X_2, X_3)$ to $S(V, U, U)$ is shown in Fig. 1.

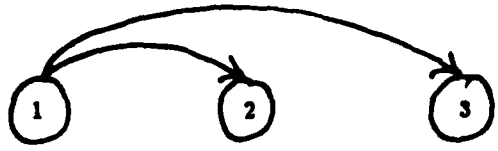


Fig. 1. The connection graph from $S(U, X_2, X_3)$ to $S(V, U, U)$

Furthermore, we observe that the connection graph from $S(V, U, U)$ to $S(X_1, V, V)$ is also the same graph in Fig. 1. Therefore, this program is uniformly connected.

Definition 3.2 Let **P** be the class of purely exponential programs P satisfying the following conditions:

(1) P is uniformly connected with no isolated node(i.e., for no node i, indegree(i)=outdegree(i)=0);
(2) Only adjacent literals in P have common nondistinguished variables;
(3) Every distinguished variable X_i occurring at position i of the head, can only occur at position i of all literals(i.e., typed distinguished variables).

We will prove that every program in **P** can be written in the form (**). It should be noted that **P** is a huge subclass of purely exponential queries, in particular it contains all binary chained purely exponential queries as defined in [4].

The next lemma is instrumental in proving our main theorem.

Lemma 3.1 *Every program P in* **P** *has the following properties:*

(1) For no node i in the connection graph of P, both indegree(i) and outdegree(i) are nonsero unless i is a *stationary node*(i.e., there is an edge from i to i[1]).

[1] We point out that due to the safety rule, stationary nodes are labeled by distinguished variables.

(2) If indegree(i)$\neq$ 0, then the variable at position i of the leftmost literal of P's body must be distinguished. Similarly, if outdegree(i)$\neq$ 0, then the variable at position i of the rightmost literal of P's body must be distinguished.

Proof: (1) Let i be a nonstationary node with nonsero indegree and outdegree, then by part(3) of definition 3.2, every literal of P's body must have a nondistinguished variable at position i. This implies that the distinguished variable at position i of P's head will not occur in P's body which is a violation of the safety rule.

(2) Suppose indegree(i)$\neq$ 0. If node i is stationary, then we are done as stationary nodes are labeled by distinguished variables. Therefore, suppose node i is not stationary, in which case again by part (3) of definition 3.2, every literal of P's body must have a nondistinguished variable at position i. This again violates the safety rule. A similar argument proves the case for which outdegree(i)$\neq$ 0. □

The next lemma states that if we expand a program P of $\mathbf{P}$, then every expression in the expansion of P enjoys the properties stated in lemma 3.1.

Lemma 3.2 Let e be an expression in the expansion of $P \in \mathbf{P}$, then e is uniformly connected and has the same connection graph as P. Moreover, both properties (1) and (2) of lemma 3.1 hold for e.

Proof: By induction on k, where k is the number of applications of recursive rule of P.

Basis $k = 0$. Obvious from lemma 3.1.

Inductive Step: Observe that for every application of the recursive rule, we increase the number of the literals by $(n-1)$. Suppose that e_k is obtained by k applications of the recursive rule:

$$e_k = S(Z_{1,1}, Z_{1,2}, \ldots, Z_{1,m}), \ldots, S(Z_{(p-1),1}, Z_{(p-1),2}, \ldots, Z_{(p-1),m}), S(Z_{p,1}, Z_{p,2}, \ldots, Z_{p,m}),$$
$$S(Z_{(p+1),1}, Z_{(p+1),2}, \ldots, Z_{(p+1),m}), \ldots, S(Z_{n+k(n-1),1}, Z_{n+k(n-1),2}, \ldots, Z_{n+k(n-1),m})$$

Now, if we expand on pth occurrence of S, we have

$$e_{k+1} = S(Z_{1,1}, Z_{1,2}, \ldots, Z_{1,m}), \ldots, S(Z_{(p-1),1}, Z_{(p-1),2}, \ldots, Z_{(p-1),m}),$$
$$S(W_{1,1}, W_{1,2}, \ldots, W_{1,m}), S(W_{2,1}, W_{2,2}, \ldots, W_{2,m}), \ \ldots, S(W_{n,1}, W_{n,2}, \ldots, W_{n,m}),$$
$$S(Z_{(p+1),1}, Z_{(p+1),2}, \ldots, Z_{(p+1),m}), \ldots, S(Z_{n+k(n-1),1}, Z_{n+k(n-1),2}, \ldots, Z_{n+k(n-1),m})$$

Obviously, it suffices to show that the connection graphs from $S(Z_{(p-1),1}, Z_{(p-1),2}, \ldots, Z_{(p-1),m})$ to $S(W_{1,1}, W_{1,2}, \ldots, W_{1,m})$ and from $S(W_{n,1}, W_{n,2}, \ldots, W_{n,m})$ to $S(Z_{(p+1),1}, Z_{(p+1),2}, \ldots, Z_{(p+1),m})$ are the same as P's connection graph. Let α be an edge from node i to node j in the connection graph of P, then by definition of the connected graph and our inductive hypothesis $Z_{(p-1),i} = Z_{p,j}$. Now, when we replaced the literal $S(Z_{p,1}, Z_{p,2}, \ldots, Z_{p,m})$ by $S(W_{1,1}, W_{1,2}, \ldots, W_{1,m}), S(W_{2,1}, W_{2,2}, \ldots, W_{2,m}), \ \ldots, S(W_{n,1}, W_{n,2}, \ldots, W_{n,m})$ in the $(k+1)$th application of the recursive rule, the variables $Z_{p,1}, Z_{p,2}, \ldots, Z_{p,m}$ are distinguished. Since indegree(j)$\neq$ 0, by part (2) of lemma 3.1, $W_{1,j}$ must be distinguished, i.e., $Z_{p,j} = W_{1,j}$. This implies that $Z_{(p-1,j)} = W_{1,j}$, and therefore there is an edge from node i to node j in the connection graph from $S(Z_{(p-1),1}, Z_{(p-1),2}, \ldots, Z_{(p-1),m})$ to $S(W_{1,1}, W_{1,2}, \ldots, W_{1,m})$. Furthermore, the stationary nodes will remain stationary. Finally, as we rename the nondistinguished variables in any application of the recursive rule, we will not create any edge which already does not exist in the connection graph of P.

In order to show that the connection graph from $S(W_{n,1}, W_{n,2}, \ldots, W_{n,m})$ to $S(Z_{(p+1),1}, Z_{(p+1),2}, \ldots, Z_{(p+1),m})$ is the same as P's connection graph, again let α be an edge from node i to node j. We will show that $W_{n,i} = Z_{(p+1),j}$. Since the assertion holds for e_k by inductive hypothesis, the definition of connection graph implies that $Z_{p,i} = Z_{(p+1),j}$. Also, as in the above case, since $Z_{p,1}, Z_{p,2}, \ldots, Z_{p,m}$ are distinguished variables for the $(k+1)$th application and outdegree(i)$\neq$ 0, by part (2) of lemma 3.1, $W_{i,n}$ is distinguished, i.e., $W_{n,i} = Z_{p,i}$. This implies that $Z_{p,i} = W_{n,i} = Z_{(p+1),i}$. Hence, there is an edge from node i to node j in the connection graph from $S(W_{n,1}, W_{n,2}, \ldots, W_{n,m})$ to $S(Z_{(p+1),1}, Z_{(p+1),2}, \ldots, Z_{(p+1),m})$. □

Theorem 3.1 Let $P \in \mathbf{P}$, then P is equivalent to a program of the form (**).

Proof: Let P' be the corresponding program of the form (**), we will show that P is equivalent to P'. Let E and E' be the expansions of P and P', respectively. Let $e \in E$ and suppose e is obtained by k applications of the recursive rule in P, then e has the form:

$$e = S(Z_{1,1}, Z_{1,2}, \ldots, Z_{1,m}), \ldots, S(Z_{n+k(n-1),1}, Z_{n+k(n-1),2}, \ldots, Z_{n+k(n-1),m})$$

Let e' be the expression obtained from k applications of the recursive rule in P' (observe that we can only expand on the rightmost literal), then e' has the form:

$$e' = S(W_{1,1}, W_{1,2}, \ldots, W_{1,m}), \ldots, S(W_{n+k(n-1),1}, W_{n+k(n-1),2}, \ldots, W_{n+k(n-1),m})$$

Let $\rho : e \rightarrow e'$ defined by $\rho(Z_{i,j}) = W_{i,j}$ for $j = 1, 2, \ldots, m$ and $i = 1, 2, \ldots, n + k(n-1)$. We will show that ρ is a containment map.

By lemma 3.2, both e and e' have the same connection graph. We first prove that ρ is well-defined. Suppose $Z_{i,j} = Z_{i',j'}$, then we need to show that $W_{i,j} = W_{i',j'}$. In case $Z_{i,j}$ is distinguished, then by part (3) of definition 3.2, $j = j'$. Now, by part (1) of definition 3.2, node j must be stationary. Hence, $Z_{i,j} = Z_{i',j'} = W_{i,j} = W_{i',j'} = X$, for some distinguished variable X. If $Z_{i,j}$ is nondistinguished, then by part (2) of definition 3.2, $Z_{i,j}$ and $Z_{i',j'}$ must either occur in two adjacent literals or the same literal. In case they occur in adjacent literals, by definition of the connection graph, there must be an edge from node j to node j'. Since both e and e' have the same connection graph, then $W_{i,j} = W_{i',j'}$. Finally, if they both occur in the same literal, again by the fact that e and e' have the same connection graph it follows that $W_{i,j} = W_{i',j'}$.

In order to show that ρ is a containment map, we observe that $S(\rho(Z_{p,1}), \rho(Z_{p,2}), \ldots, \rho(Z_{p,m})) = S(W_{p,1}, W_{p,2}, \ldots, W_{p,m})$. All that remains to be shown is that ρ maps distinguished variables to distinguished variables. Let $Z_{i,j} = X$ be a distinguished variable. If node j is stationary, then X occurs at position j of all literals in both e and e'. Hence, $\rho(Z_{i,j}) = W_{i,j} = X$. If indegree($j$)$\neq 0$, then by lemma 3.2, $Z_{1,j} = X = W_{1,j} = \rho(Z_{1,j})$. If outdegree($j$)$\neq 0$, then by lemma 3.2, $Z_{n+k(n-1),j} = X = W_{n+k(n-1),j} = \rho(Z_{n+k(n-1),j})$. This shows that every expression in E can be mapped to an expression in E'. The converse is trivial. □

I. Guessarian [4] has shown that binary chained purely exponential queries can be written as linear queries by using a very elaborate fixed point technique. This result follows immediately from theorem 3.1.

Corollary 3.1 **Binary chained purely exponential queries can be written as linear queries.**

4. Progressively Weaker Chain of Template Dependencies

A full *template dependency*(TD) is a formal statement τ of the form:

$$\forall Y_{1,1}\forall Y_{1,2}\ldots\forall Y_{1,m}\cdots\forall Y_{n,1}\forall Y_{n,2}\ldots\forall Y_{n,m}\forall X_1\forall X_2\ldots\forall X_m($$
$$R(Y_{1,1},Y_{1,2},\ldots,Y_{1,m})\wedge R(Y_{2,1},Y_{2,2},\ldots,Y_{2,m})\wedge\cdots\wedge R(Y_{n,1},Y_{n,2},\ldots,Y_{n,m})$$
$$\longrightarrow R(X_1,X_2,\ldots,X_m)),$$

where for $i=1,\ldots,m$, $X_i=Y_{j,k}$ for some $1\leq j\leq n$ and $1\leq k\leq m$.

In [3], an infinitely weaker and stronger sequence τ_0, τ_1, τ_2, ... of template dependencies is constructed via the TD graph. These sequences have been used to establish various results regarding TDs. We will show here that the expansion of program (I) for transitive closure of a relation R will provide a natural example of an infinitely weaker chain. We will use the notation, $\tau\models\sigma$, to state that TD σ is a *logical consequence* of τ.

Theorem 4.1 There exists an infinite sequence of full TDs τ_0, τ_1, τ_2, τ_3, ... such that $\tau_i\models\tau_{i+1}$ for each i, $i=1,2,3,\ldots$ and no two τ_is are equivalent.

Proof: For simplicity, we will drop the quantifiers from the TDs' notation. Let τ_0, τ_1, τ_2, τ_3, ... be the following expressions in the expansion of the transitive closure of R.

$$\tau_0: \quad R(X,U) \wedge R(U,Y) \rightarrow R(X,Y).$$

$$\tau_1: \quad R(X,U_1) \wedge R(U_1,U_2) \wedge R(U_2,Y) \rightarrow R(X,Y).$$

$$\tau_2: \quad R(X,U_1) \wedge R(U_1,U_2) \wedge R(U_2,U_3) \wedge R(U_3,U_4) \wedge R(U_4,Y) \rightarrow R(X,Y).$$

$$\tau_3: \quad R(X,U_1) \wedge R(U_1,U_2) \wedge R(U_2,U_3) \wedge R(U_3,U_4) \wedge R(U_4,U_5) \wedge R(U_5,U_6)$$
$$\wedge R(U_6,U_7) \wedge R(U_7,U_8) \wedge R(U_8,Y) \rightarrow R(X,Y).$$

.

.

.

$$\tau_i: \quad R(X,U_1) \wedge R(U_1,U_2) \wedge \ldots \wedge R(U_{2^i-1},U_{2^i}) \wedge R(U_{2^i},Y) \rightarrow R(X,Y).$$

$$\tau_{i+1}: \quad R(X,U_1) \wedge R(U_1,U_2) \wedge \ldots \wedge R(U_{2^{i+1}-1},U_{2^{i+1}}) \wedge R(U_{2^{i+1}},Y) \rightarrow R(X,Y).$$

.

.

.

We first show that $\tau_1 \models \tau_2$ (the general case is the obvious generalisation of this). Let r satisfy τ_1, and suppose that $(a,a_1),(a_1,a_2),(a_2,a_3),(a_3,a_4),(a_4,b)$ are tuples of r mapped to the hypothesis rows of τ_2, respectively. We want to show that (a,b) is also a member of r. Since τ_1 holds in r, if we map $(a,a_1),(a_1,a_2),(a_2,a_3)$ to $(X,U_1)(U_1,U_2)(U_2,Y)$ respectively, then we must have tuple (a,a_3) in r. Now, map $(a,a_3),(a_3,a_4),(a_4,b)$ to the hypothesis of τ_1, again since τ_1 holds in r, we must have (a,b) in r.

To show that $\tau_i \not\models \tau_{i+1}$, let

$$r = \{(X,U_1),\ (U_1,U_2),\ \ldots,\ (U_{2^{i+1}-1},\ U_{2^{i+1}}),\ (U_{2^{i+1}},Y),\ (X,Y)\}$$

i.e., a relation consisting of all τ_{i+1}'s rows. If we map $(X,U_1),\ (U_1,U_2),\ \ldots,\ (U_{2^i},\ U_{2^i+1})$ to the hypothesis rows of τ_i, then for r to satisfy τ_i, tuple (X,U_{2^i+1}) must be in r, which is not according to our construction of r. □

5. Conclusion

We have identified a sufficient condition to rewrite purely exponential queries as linear queries. Consequently, as a corollary, we have obtained a new proof for the result of Guessarian [4]. In addition, a natural infinite chain of progressively weaker TDs is constructed via expansion of the logic program for transitive closure of a relation R. We hope the techniques developed in this paper motivate further research in this area.

Reference

[1] Aho,A.V., and Ullman,J.D., Ullman, Universality of Data Retrieval Languages, *ACM Symp. on Principles of Prog. Lang.*, 1979, 110–120.

[2] Banchilon,F., and Ramakrishnan, R., An Amateur's Introduction to Recursive Query Processing Strategies, *Proc. ACM SIGMOD Conf.*, 1986, 16–52.

[3] Fagin,R., Maier,D., Ullman,J.D., and Yannakakis,M., Tools for Template Dependencies, *SIAM J. on Computing 12:1*, 1983, 36–59.

[4] Guessarian,I., Fixpoint Techniques in Data Base Recursive Logic Programs, *Bull. EATCS, 29*, 1986, 32–35.

[5] Jagadish,H.V., Agrawal,R., and Ness,L., A Study of Transitive Closure As a Recursion Mechanism, *Proc. ACM SIGMOD Conf.*, 1987, 331–344.

[6] Naughton,J., Data Independent Recursion in Deductive Databases, *ACM SIGMOD-SIGACT Symp. on Principles of Database Systems*, 1986, 267–279.

[7] Sagiv,Y., Optimising Datalog Program, *ACM SIGMOD-SIGACT Symp. on Principles of Database Systems*, 1987, 349–362.

[8] Taghva,K., Some Characterizations of Finitely Specifiable Implicational Dependency Families, *Information Processing Letters 23*, 1986, 153-158.

[9] Zhang,W., and Yu,C.T., A Necessary Condition for a Doubly Recursive Rule to Be Equivalent to A Linear Recursive Rule, *Proc. ACM SIGMOD Conf.*, 1987, 345–356.