Copyright © 1985, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

BOUNDED RECURSION IN DEDUCTIVE DATABASES

by

Y. E. Ioannidis

Memorandum No. UCB/ERL M85/6

2 February 1985

Cover

BOUNDED RECURSION IN DEDUCTIVE DATABASES

by

Y. E. Ioannidis

Memorandum No. UCB/ERL M85/6

2 February 1985

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

BOUNDED RECURSION IN DEDUCTIVE DATABASES

Yannis E. Ioannidis
Department of Electrical Engineering and Computer Science
Computer Science Division
University of California
Berkeley, CA 94720

Abstract

A virtual relation (or view) can be defined with a recursive statement that is a function of one or more base relations. In general, the number of times such a statement must be applied in order to retrieve all the tuples in the virtual relation depends on the contents of the base relations involved in the definition. However, there exist statements for which there is an upper bound on the number of applications necessary to form the virtual relation, independent of the contents of the base relations. Considering a restricted class of recursive statements, we give necessary and sufficient conditions for statements in the class to have this bound.

1. INTRODUCTION

In the past few years major attempts have been made to improve the power of database systems, in particular those based on the relational model ([Codd70]). A significant part of this effort has been in the direction of the formalization, design and development of deductive databases. As defined in [Gall84] , "a deductive database is a database in which new facts may be derived from facts that were explicitly introduced". A very important difference between a deductive and a conventional relational database is that in the former new facts may be derived recursively. This very characteristic of deductive databases is what makes query processing a difficult task in such an environment. The main problem that arises is how to detect the point at which further processing will give no more answers to a given query. Many researchers have studied and proposed solutions to this termination problem for various cases (see, for example, [Naqv84] , [Reit78] and [Chan81]). However no single solution is known for the general problem.

This research was supported by the National Science Foundation under Grant ECS-8300463

A common characteristic among all the proposed solutions that we are aware of is that the termination condition relies on the data explicitly stored in the database. In general this is necessary. However, there are some cases where a termination condition exists, which is independent of the particular instance of the database. The purpose of this paper is to identify and characterize these cases. Restricting ourselves to a particular class of recursive statements, we give necessary and sufficient conditions for the existence of a data-independent termination condition.

We assume that the reader has some familiarity with mathematical logic and graph theory, although nothing extremely involved from these fields will be needed. Nevertheless, we are going to use some of their notions without definition. The first few chapters of any standard text in mathematical logic (e.g. [Ende72]) and graph theory (e.g. [Bond76]) provide the necessary background. Furthermore we assume that the reader is familiar with relational databases at the level of [Date82] . Finally, we would refer the reader to [Gall78] and [Gall81] as extremely valuable sources of information on the relationship between mathematical logic and deductive databases.

The paper is organized as follows. In Section 2 we give the formal framework of a deductive database that we will be considering. Our investigation is restricted to a subset of all possible deductive databases. We outline all the restrictions we are imposing on the database and explain the reasons for doing so. In Section 3 we introduce some examples of cases where even though data is derived recursively, the termination point is known *a−priori* (i.e. it does not depend on the explicitly stored data). Section 4 contains the description of the graph model we used as a tool to derive our results. In Section 5, we state and prove the main result of this paper: the necessary and sufficient conditions for a termination condition to exist that is independent of the data explicitly stored in the database. Furthermore we illustrate our result with a number of characteristic examples. In Section 6 we present algorithms to check the conditions of the theorem on the graph model we have introduced. Section 7 discusses the importance of our

results and investigate ways in which they can be used to speed up query processing in deductive databases. Finally, in Section 8 we summarize our results and discuss more problems for future work in the area.

2. ASSUMPTIONS

The following definitions about first-order formulas ([Ende72]) will be useful in our analysis.

Definition 2.1: A first-order formula is equivalent to a *Horn clause* if and only if it is of the form

$$A_1 \wedge A_2 \wedge \cdots \wedge A_n \rightarrow C$$

with all the variables appearing in the formula being (implicitly) universally quantified.

The formula to the left of $\rightarrow$ will be called the *antecedent* and that to the right of $\rightarrow$ the *consequent*. Each one of C, A_1, A_2, ..., A_n is an atomic formula (see [Ende72]), i.e. it is of the form

$$P(t_1,t_2,...,t_n)$$

where P is a predicate symbol and t_i, $1 \leq i \leq n$, is a term (a variable symbol or a constant symbol or a function symbol "applied" on one or more terms). Finally, a Horn clause is recursive when the predicate that appears in the consequent appears at least once in the antecedent as well. Throughout the paper we will be using the terms "formula" and "statement" indistinguishably. We will also alternate between the terms "predicate" and "relation", in light of the discussions in [Gall78] .

Definition 2.2: Two variables x, y *appear under the same predicate* in a statement if and only if there is an atomic formula $P(...,x,...,y,...)$ appearing in the statement, where P is a predicate symbol.

Definition 2.3: Consider a recursive statement which is equivalent to a Horn clause. The sole predicate appearing in the consequent of the statement will be called the *recursive predicate*

of the statement. Any other predicate in the statement will be called *non−recursive*.

Definition 2.4: A variable will be called *consequent* if and only if it appears under the recursive predicate in the consequent of the statement. Otherwise it will be called *antecedent*.

We consider a deductive database to be a relational database (in the sense of [Codd70]) enhanced with a set of Horn clauses. If there is some recursive statement or a set of mutually recursive statements appearing in the database, then the termination problem mentioned in Section 1 arises. We will examine this problem with respect to the processing of a single recursive statement only.

We restrict our attention to recursive statements that satisfy the following conditions:

1) The recursive predicate of the statement appears only *once* in the antecedent.

2) There are *no function symbols* in the statement.

3) There are *no constant symbols* in the statement.

4) No variable appears more than once under the recursive predicate in the consequent. Furthermore, no subsequence of the variables appearing under the recursive predicate in the consequent is a permutation of the corresponding subsequence of the variables in the recursive predicate in the antecedent.

Our motivation behind restriction (1) is simplicity. Having more than one appearance of the recursive predicate in the antecedent severely complicates our analysis. Since many of the recursive statements expected in a real world system have the recursive predicate appearing only once in the antecedent, we believe that assumption (1) is reasonable. Function symbols appearing in a recursive statement may lead to infinite relations. For example, consider the following recursive statement containing the '+' function:

$$P(x) \rightarrow P(x+1)$$

Suppose that initially P contained the single tuple $<1>$. It is clear that the above statement makes P an infinite relation containing all the positive integers. Situations like that are not easily

handled in a database environment, if at all; to avoid them we have imposed restriction (2). The last two restrictions were imposed for the sole purpose of getting a uniform result. We speculate that it will not be very difficult to remove them, thereby generalizing our results. In fact, considering a recursive statement that does contain constant symbols, we may remove them by performing selections and projections on the relations involved ([Codd70]). The new statement is free of constant symbols and if applied to the new set of relations produced by the operations mentioned above, will give the same result as if the original statement was applied on the original relations. Regarding restriction (4), it may appear somewhat artificial, but its meaning will become clear shortly, when we will describe the way we model a recursive statement.

A final point worth mentioning here is that without loss of generality we may assume that there are no equalities in the statement. If there is any equality between two variables, we may easily remove it by replacing one of these variables with the other wherever it appears in the statement. It is clear that the new statement is equivalent to the initial one.

Definition 2.4: A recursive statement will be called *simple* if and only if it satisfies conditions (1) through (4) above and does not contain any equality symbol.

3. SOME EXAMPLES

Consider the following simple recursive statement α:

$$\alpha: \qquad P(x) \wedge Q(x,y) \rightarrow P(y)$$

Relation Q is a base relation in the system (that is, it is stored explicitly), whereas P is a derived relation. It is clear that in addition to α above, there has to be some non-recursive way to get some initial tuples into P. As an example assume that this is done with β:

$$\beta: \qquad R(x) \rightarrow P(x)$$

Assume that R is a base relation as well. A natural way of thinking about the processing of α is iteration. In particular, the statement is applied once on the initial contents of the relations involved and produces some new tuples for P. This process is repeated for these new tuples and then again, until no new tuples are produced. It is obvious that in general there is no upper

bound on the number of times this process has to be repeated in order to get all the derivable tuples for P. If we consider Q to represent the edge set of a directed graph and R to contain some nodes of the graph, then P comes to contain all nodes reachable from those in R. At step i of the iterative process described above, we insert into P all those nodes of the graph that are reachable from some node in R through a path of length i. Since the graph may contain arbitrarily long paths, it is not possible to know in advance how many iterations will be needed.

As another example of a simple recursive statement, consider γ:

$$\gamma: \qquad P(z) \wedge Q(z) \wedge R(y) \rightarrow P(y)$$

where Q and R are base relations. Clearly, one application of the statement is enough, regardless of the initial contents of the relations P, Q and R. Statement γ derives for P all the tuples in R, as long as there is initially one tuple in P that joins with (that is, is equal to) some tuple in Q. Any further step in the iteration will fail to produce any new tuples for P. So, for γ, unlike α, there exists an upper bound on the number of times the statement has to be applied to derive all the tuples possible in the recursive relation, that number being equal to 1.

As a third example consider δ:

$$\delta: \qquad P(z,x) \wedge Q(y) \rightarrow P(x,y)$$

with Q being a base relation. In this case we are taking the cartesian product of the projection on the second attribute of the initial copy of P with Q. However one step is not enough for δ. One more step will be needed, where actually the cartesian product of Q with itself will be derived for P. Nevertheless, there will be no need for a third step. Further processing will only continue producing the cartesian product of Q with itself. Therefore, δ, like γ, has an upper bound on the number of times it needs to be applied, only that now the tight upper bound is equal to two. This is not to say that the second step of the iteration will always produce new tuples for P. In fact, if Q is initially empty, not even the first step will be needed. However the point is that there exists an instance of Q and P that will need two steps, whereas there exists no instance of these relations that will need three.

The examples given above indicate that the way in which the variables appearing in the statement are connected with each other through the predicates, plays an important role on whether an upper bound on the number of iterative steps needed to produce all derivable tuples exists or not. In order to study the properties of these statements we have developed a graph model for them, which reflects this connection among the variables. The description of this model is the subject of the next section.

4. THE MODEL

Suppose that we are given a simple recursive statement. We will model this statement by a labeled, weighted, directed graph constructed as follows:

(i) To every variable appearing in the statement we associate a node in the graph.

(ii) For every pair of variables x,y that appear under the same non-recursive predicate Q in the statement there is a labeled undirected edge $(x-y)$ in the graph between the corresponding two nodes x,y, for each such predicate Q. The label of the edge is Q and its weight is 0.

(iii) For every pair of variables x,y such that x appears under the recursive predicate P in the antecedent and y appears in the corresponding position of the recursive predicate in the consequent, there is a directed edge $(x\rightarrow y)$ in the graph from node x to node y with weight 1 and its inverse edge $(y\rightarrow x)$ with weight -1. Each directed edge has label P.

The graph constructed this way from a simple recursive statement α will be called the *α-graph*. The subgraph induced on the α-graph by the undirected edges defined in (ii) will be called the *static α-graph*. The spanning subgraph of the α-graph with edge set its directed edges defined in (iii) will be called the *dynamic α-graph*. Finally, the weight of a path (cycle) in the graph is defined to be the sum of the weights of the edges along the path (cycle). Regarding undirected edges, they can be traversed in both directions, as if there were two opposite directed edges.

As an example consider the following simple recursive statement:

$$\alpha: \qquad P(z,w) \wedge Q(z,x) \wedge R(w,u) \wedge S(u,x,y) \rightarrow P(x,y)$$

The α-graph is shown in figure 4.1.

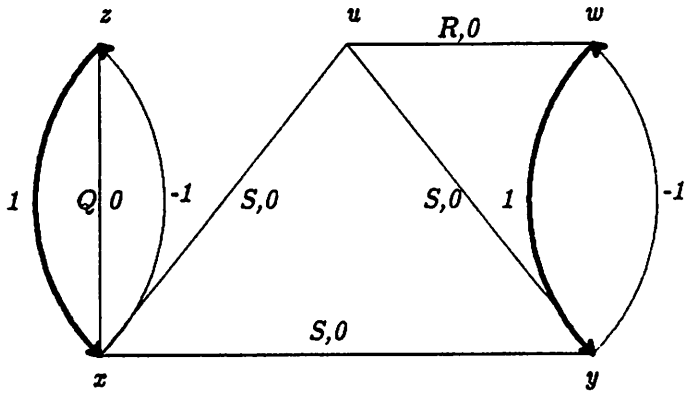

Fig. 4.1 : The α-graph

We can now see the meaning of restriction (4) in Section 2. All it says is that the dynamic subgraph of a simple recursive statement (restricted on the positive edges) is a forest. This has the implication that there is at most one path from any node to any other node in the subgraph, which proved to be crucial for the accuracy of our results.

Regarding the unification algorithm ([Robi65]), we would like to indicate an important relationship between that and the graph model analyzed above. Unification is a first-order theorem proving algorithm. The iterative process used for recursive statements in Section 2, is equivalent, with respect to the final outcome, to a unification process. In particular, consider two copies of the statement, with distinct variable symbols for all the antecedent variables. That is, consider α as given above and α' as given below:

$$\alpha': \qquad P(z',w') \wedge Q(z',x) \wedge R(w',u') \wedge S(u',x,y) \rightarrow P(x,y)$$

Clearly α is equivalent to α', since all we did was to change some variable names. We can now unify the recursive predicate in the antecedent of the first copy with that in the consequent of the second copy (the unification algorithm would work with the statements put in clause form, but its actions are equivalent to the ones we describe here). The resolvent is a new simple recursive statement, which if applied on the initial instance of the recursive predicate, will give exactly the

same result with the application of the original statement on the outcome of the first step of the iterative process. For our example the resolvent comes out to be:

$$P(z',w') \wedge Q(z',z) \wedge R(w',u') \wedge S(u',z,w) \wedge Q(z,x) \wedge R(w,u) \wedge S(u,x,y) \rightarrow P(x,y)$$

Regarding unification, the dynamic subgraph of a simple recursive statement captures some important information. Namely, it shows the substitution of the variables that one has to make to unify the two literals in the two copies of the statement. For every positive directed edge of the graph, the tail should be substituted in the second copy for the head, to obtain the resolvent. In the example above, z was substituted for x and w was substituted for y, which is exactly what the directed edges $(z \rightarrow x)$ and $(w \rightarrow y)$ in figure 4.1 indicate. We will not be directly referring to the unification algorithm, but the ideas behind it have a significant impact on our analysis.

Finally, there is a notational comment we would like to make about the graph model described above. According to the definition, there is a one-to-one correspondence between the positive and the negative directed edges. The positive ones alone are enough to carry all the information captured by the directed edges in the graph. Hereafter, we will be referring to the dynamic subgraph as containing the positive edges of the graph only, the negative ones implicitly assumed only whenever the weight of a path is discussed. Likewise, in all the figures we will draw the positive edges only. Finally, since the weight of some edge is easily determined from whether it is undirected (weight zero) or directed (weight one), we will put no weights on the edges.

5. THE PROBLEM

Let the following be a simple recursive statement.

$$P(x_1,x_2,\ldots,x_m) \wedge \beta \rightarrow P(y_1,y_2,\ldots,y_m) \qquad (1)$$

The subformula β is a conjunction of atomic formulas, none of which involves the predicate P, and all variables are assumed to be universally quantified. There are two equivalent ways of expressing (1) in a nonrecursive way.

- The above statement may be viewed as equivalent to the following infinite sequence of statements:

$P_0(x_1,x_2,...,x_m) \wedge \beta \rightarrow P_1(y_1,y_2,...,y_m)$
$P_1(x_1,x_2,...,x_m) \wedge \beta \rightarrow P_2(y_1,y_2,...,y_m)$
$P_2(x_1,x_2,...,x_m) \wedge \beta \rightarrow P_3(y_1,y_2,...,y_m)$
.....

In the above statements, P_0 denotes the initial contents of P, P_1 denotes the tuples "inserted" into P after applying the recursive statement once, P_2 denotes the tuples "inserted" in P after applying the recursive statement on the new tuples produced by the previous application, and so on. The final result for relation P is $\bigcup_{i=0}^{\infty} P_i$. The i-th statement above will be called the *i-th application* of statement (1). Note that this infinite non-recursive expression of (1) actually reflects the iterative process to materialize P, along the lines of our discussions in the previous sections.

- Statement (1) may be also viewed as equivalent to the statements

$P_0(x_1,x_2,...,x_m) \wedge \beta_0 \rightarrow P_1(y_1,y_2,...,y_m)$
$P_0(x_1^{(1)},x_2^{(1)},...,x_m^{(1)}) \wedge \beta_1 \wedge \beta_0 \rightarrow P_2(y_1,y_2,...,y_m)$
$P_0(x_1^{(2)},x_2^{(2)},...,x_m^{(2)}) \wedge \beta_2 \wedge \beta_1 \wedge \beta_0 \rightarrow P_3(y_1,y_2,...,y_m)$
.....

where, for all $i \geq 0$, there exists some substitution θ_i of the variables in β (the details of which will not concern us for the moment), such that $\beta_i = \beta[\theta_i]$ and $P_0(x_1^{(i-1)},x_2^{(i-1)},...,x_m^{(i-1)}) = P_0(x_1,x_2,...,x_m)[\theta_i]$, Note that θ_0 maps each variable to itself. In a similar but somewhat different way than for the applications of (1), the i-th statement above will be called the *(i-1)-th expansion* of statement (1), so that the first of these statements, which is actually the statement itself, is the 0-th expansion. Each one of these expansions is applied on the initial contents of P. Clearly, the P_i's above are the same as the ones in the "application" view of the recursive statement, i.e. the tuples produced by the i-th application of (1) for P_i, are the same as those produced by the $(i-1)$-th expansion of (1) for P_i. In the end P is again equal to $\bigcup_{i=0}^{\infty} P_i$. Note that the antecedent of each expansion is equal to that of the previous expansion with the recursive predicate being replaced by yet another instance of the antecedent of the original statement with different variables. In terms of the unification algorithm mentioned in Section 4, the k-th expansion of a

simple recursive statement α is the resolvent of its $(k-1)$-th expansion and α itself. We will be referring to the $k-$th expansion of a recursive statement α as α_k. Since the statement itself is its own 0-th expansion, we have that $\alpha = \alpha_0$.

In both cases above, the initial statement becomes equivalent to an infinite number of nonrecursive statements. However, since in a database environment all the relations are finite, and because of the fact that we are considering simple recursive statements only, which contain no functions, after some point the nonrecursive statements will stop producing any new tuples for P, and therefore the whole process eventually terminates. Moreover, the process terminates exactly when some i-th application (or the corresponding $(i-1)$-th expansion of (1)) fails to produce any new tuples for the first time.

This is an appropriate place for the following definitions.

Definition 5.1: Let α be a simple recursive statement. The *rank* of α is defined to be the smallest i such that α_i does not produce any tuple not already contained in some P_j for $0 \leq j \leq i$.

Note that, in general, the rank of α depends on the contents of the relations involved in α.

Definition 5.2: A simple recursive statement will be called *bounded* if and only if there exists a finite upper bound on its rank independent of the contents of the relations involved in the statement.

In view of the definitions above we can pose our problem as the following question:

When is a simple recursive statement bounded?

We are also interested in finding this upper bound in the cases it exists. The answer to this question is given by the following theorem.

Theorem: Let α be a simple recursive statement. Statement α is bounded if and only if the α-graph contains no cycle of non-zero weight. In that case a tight upper bound on the rank of α is equal to the maximum weight of any path in the α-graph.

Before proceeding in proving the above theorem we need to establish a regular naming scheme for the variables of simple recursive statements. Restriction (4) denotes that the dynamic α-graph is a forest; therefore, every connected component in the graph is a tree. For every leaf in such a tree there is unique path from the root of that tree to that leaf. We number each one of these paths with a unique positive integer. Having this in mind we will use the following naming convention for the variables. Every consequent variable will be called $y_{i,j}$. The index j is the distance of the variable from the root of the component of the dynamic α-graph where this variable belongs. The index i is the number assigned to the root-to-leaf path on which the variable lies. Of course, some variables belong to more than one root-to-leaf paths (e.g. all the variables with out-degree greater than 1). Fortunately, the fact that some variables can be given more than one name will not affect our results; on the contrary it will make our notation much simpler. As it concerns the antecedent variables, there is exactly one of them in each component of the dynamic α-graph, namely its root. If this was not the case some component of the graph would not be a tree, so we would violate restriction (4). This unique antecedent variable will be called $x_i^{(0)}$, where i is again any of the numbers assigned to the paths from this variable to the leaves of the component of the dynamic α-graph where the variable appears. Notice that any antecedent variable that does not appear under the recursive predicate is by itself a component in the dynamic α-graph. The significance of the superscript in the notation of the antecedent variables will be clarified shortly.

As an example of the notation, consider figure 5.1 of the α-graph of some simple recursive statement α, showing the names of the variables involved. The statement corresponding to the figure is

$$P(x_1^{(0)},y_{1,1},y_{1,2},y_{1,3},y_{1,4},x_2^{(0)},y_{2,1},x_3^{(0)},y_{3,1},x_{4_{(0)}},y_{4,1},x_5^{(0)}) \wedge$$

$$Q_1(y_{1,2},x_2^{(0)}) \wedge Q_2(y_{2,1},y_{3,2}) \wedge Q_3(x_3^{(0)},x_4^{(0)}) \wedge Q_4(y_{4,1},x_5^{(0)}) \wedge Q_5(y_{5,1},y_{1,5}) \rightarrow$$

$$P(y_{1,1},y_{1,2},y_{1,3},y_{1,4},y_{1,5},y_{2,1},y_{2,2},y_{3,1},y_{3,2},y_{4,1},y_{4,2},y_{5,1})$$

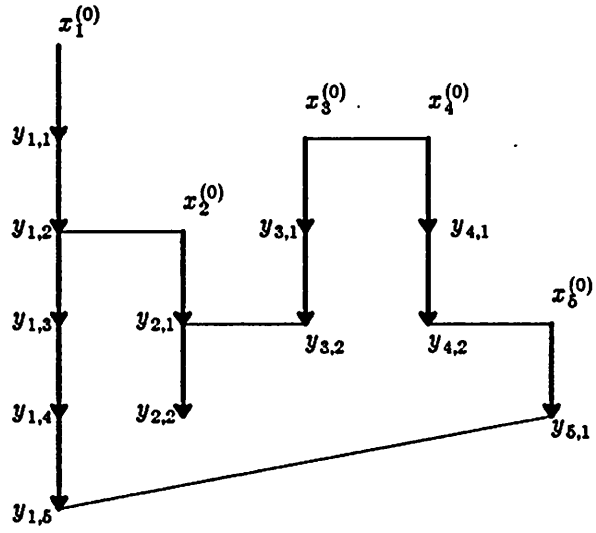

Fig. 5.1 : Example of variable naming

In the first expansion of some statement α, each consequent variable will be replaced by the corresponding variable appearing under the recursive predicate in the antecedent, as described in Section 4. Furthermore some new variables will be introduced to replace the antecedent variables. Our convention will be that each new variable will have the name of the one it replaces with the superscript increased by 1. That is, $x_i^{(0)}$ will be replaced by $x_i^{(1)}$. Similarly in the n-th expansion $x_i^{(0)}$ will be replaced by $x_i^{(n)}$. The meaning of the superscript should be clear now.

In order to be able to refer to the two kinds of variables in a uniform way we introduce the following generic variable name $z_{i,j}^{(n)}$, defined as

$$z_{i,j}^{(n)} = \begin{cases} x_i^{(n-j)} & 0 \leq j \leq n \\ y_{i,(j-n)} & j > n \end{cases} \tag{2}$$

Under the above naming scheme we can see that every antecedent variable $x_i^{(0)}$ in α can be written as $z_{i,0}^{(0)}$ and every consequent variable $y_{i,j}$ in α can be written as $z_{i,j}^{(0)}$. Hence, in this naming scheme, every variable in α can be written with superscript (0). We should also mention here that the mapping from z's to x's and y's is not one-to-one. Two z's with the same $n-j$ difference will actually represent the same variable.

Lemma 5.1: Let α be a simple recursive statement and let $z_{i,j}^{(0)}$ be a variable appearing in some position under some predicate in the antecedent. Then for α_n, this variable will be

substituted by $z_{i,j}^{(n)}$.

Proof: We will prove it by induction on n.

Basis: Let $n=0$. The lemma is trivially true.

Induction hypothesis: Suppose that the lemma is true for all expansions less than n. We are going to prove it for the n-th expansion.

a) If $j=0$ then from (2) we can see that we have an antecedent variable. Since in every expansion we have to use a new name to substitute the variable and our convention is to increase the superscript of the corresponding variable in the previous expansion, we get from the induction hypothesis that $z_{i,0}^{(0)}$ will be replaced by $z_{i,0}^{(n)}$.

b) If $j>0$ then from (2) again it is clear that $z_{i,j}^{(n)}$ is a consequent variable. Therefore, it will be replaced by the variable in the same position under the recursive predicate in the antecedent of $\alpha_{(n-1)}$. By the induction hypothesis this will be of the form $z_{i',j'}^{(n-1)}$, where $z_{i',j'}^{(0)}$ is the variable appearing in the same position under the recursive predicate in the antecedent of α. However $z_{i',j'}^{(0)}$ and $z_{i,j}^{(0)}$ are in the same root-to-leaf path in their component of the dynamic α-graph. So, it is $i' = i$. Furthermore from the construction of the graph there is a directed edge from $z_{i',j'}^{(0)}$ to $z_{i,j}^{(0)}$, and therefore the former is one closer to the root. So, $j' = j-1$. In conclusion we see that $z_{i,j}^{(0)}$ will be replaced by $z_{i,j-1}^{(n-1)}$ which however is equal to $z_{i,j}^{(n)}$ because of (2). □

In view of Lemma 5.1 above we have that in the expansions of α, as given in Section 4, it is

$$\theta_n = z_{i,j}^{(n)} \,/\, z_{i,j}^{(0)} \tag{3}$$

meaning that $z_{i,j}^{(n)}$ will be substituted for $z_{i,j}^{(0)}$.

As an example consider the following statement

$$P(x_1^{(0)},x_2^{(0)}) \wedge Q(x_1^{(0)},y_{2,1}) \wedge R(y_{1,1}) \rightarrow P(y_{1,1},y_{2,1})$$

Its 1-st and 2-nd expansions are given below.

$$P(x_1^{(1)},x_2^{(1)}) \wedge Q(x_1^{(1)},x_2^{(0)}) \wedge R(x_1^{(0)}) \wedge Q(x_1^{(0)},y_{2,1}) \wedge R(y_{1,1}) \rightarrow P(y_{1,1},y_{2,1})$$

$$P(x_1^{(2)},x_2^{(2)}) \wedge Q(x_1^{(2)},x_2^{(1)}) \wedge R(x_1^{(1)}) \wedge Q(x_1^{(1)},x_2^{(0)}) \wedge R(x_1^{(0)}) \wedge Q(x_1^{(0)},y_{2,1}) \wedge R(y_{1,1}) \rightarrow P(y_{1,1},y_{2,1})$$

Using the naming scheme we have introduced for the variables, the above three expansion (0-th, 1-st and 2-nd) can be written as follows:

$$P(z^{(0)}_{1,0},z^{(0)}_{2,0}) \wedge Q(z^{(0)}_{1,0},z^{(0)}_{2,1}) \wedge R(z^{(0)}_{1,1}) \rightarrow P(z^{(0)}_{1,1},z^{(0)}_{2,1})$$

$$P(z^{(1)}_{1,0},z^{(1)}_{2,0}) \wedge Q(z^{(1)}_{1,0},z^{(1)}_{2,1}) \wedge R(z^{(1)}_{1,1}) \wedge Q(z^{(0)}_{1,0},z^{(0)}_{2,1}) \wedge R(z^{(0)}_{1,1}) \rightarrow P(z^{(0)}_{1,1},z^{(0)}_{2,1})$$

$$P(z^{(2)}_{1,0},z^{(2)}_{2,0}) \wedge Q(z^{(2)}_{1,0},z^{(2)}_{2,1}) \wedge R(z^{(2)}_{1,1}) \wedge Q(z^{(1)}_{1,0},z^{(1)}_{2,1}) \wedge R(z^{(1)}_{1,1}) \wedge Q(z^{(0)}_{1,0},z^{(0)}_{2,1}) \wedge R(z^{(0)}_{1,1}) \rightarrow P(z^{(0)}_{1,1},z^{(0)}_{2,1})$$

Observe that each variable changes in the expansions according to Lemma 5.1.

Consider a typical path in the α-graph, corresponding to the part of the statement shown below (without loss of generality we may assume that all the undirected edges in the path come from binary predicates).

$$P(z^{(0)}_{1,0},z^{(0)}_{1,1},\ldots,z^{(0)}_{1,n_1-1},z^{(0)}_{2,0},z^{(0)}_{2,1},\ldots,z^{(0)}_{2,n_2-1},\ldots,z^{(0)}_{m,0},z^{(0)}_{m,1},\ldots,z^{(0)}_{m,n_m-1},\ldots) \wedge$$

$$Q_1(z^{(0)}_{1,k_1},z^{(0)}_{2,l_2}) \wedge Q_2(z^{(0)}_{2,k_2},z^{(0)}_{3,l_3}) \wedge \ldots \wedge Q_{m-1}(z^{(0)}_{m-1,k_{m-1}},z^{(0)}_{m,l_m}) \wedge \ldots \rightarrow$$

$$P(z^{(0)}_{1,1},z^{(0)}_{1,2},\ldots,z^{(0)}_{1,n_1},z^{(0)}_{2,1},z^{(0)}_{2,2},\ldots,z^{(0)}_{2,n_2},\ldots,z^{(0)}_{m,1},z^{(0)}_{m,2},\ldots,z^{(0)}_{m,n_m},\ldots) \qquad (4)$$

The graph corresponding to the path above can be seen in figure 5.2.

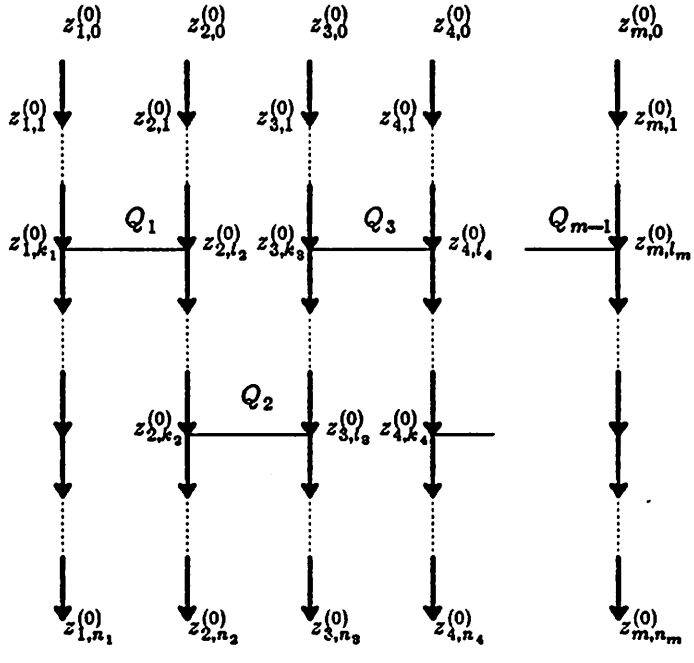

Fig. 5.2 : Typical path in the graph of some simple recursive statement

Because of Lemma 5.1 we have that α_n will be of the form

$$P(z^{(n)}_{1,0},z^{(n)}_{1,1},\ldots,z^{(n)}_{1,n_1-1},z^{(n)}_{2,0},z^{(n)}_{2,1},\ldots,z^{(n)}_{2,n_2-1},\ldots,z^{(n)}_{m,0},z^{(n)}_{m,1},\ldots,z^{(n)}_{m,n_m-1},\ldots)\wedge$$

$$\bigwedge_{j=1}^{m-1} Q_j(z^{(0)}_{j,k_j},z^{(0)}_{j+1,l_{j+1}})\wedge \bigwedge_{j=1}^{m-1} Q_j(z^{(1)}_{j,k_j},z^{(1)}_{j+1,l_{j+1}})\wedge\ldots\wedge \bigwedge_{j=1}^{m-1} Q_j(z^{(n)}_{j,k_j},z^{(n)}_{j+1,l_{j+1}})\wedge\ldots\rightarrow$$

$$P(z^{(0)}_{1,1},z^{(0)}_{1,2},\ldots,z^{(0)}_{1,n_1},z^{(0)}_{2,1},z^{(0)}_{2,2},\ldots,z^{(0)}_{2,n_2},\ldots,z^{(0)}_{m,1},z^{(0)}_{m,2},\ldots,z^{(0)}_{m,n_m},\ldots) \tag{5}$$

For the following lemmas it will be necessary to group the various Q_i's in a different way. In order to do that we will use the following family of functions f_n: $\mathbb{Z}\rightarrow\{0,1,\ldots,n-1\}$, ($\mathbb{Z}$ the set of integers) defined for all $n>0$ as follows:

$$f_n(x) = (x \bmod n)$$

In the above $(x \bmod n) = r$ if and only if $(x-r)$ divides n and $0 \leq r < n$.

Lemma 5.2: Function f_n: $S \rightarrow \{0,1,\ldots,n-1\}$ is one-to-one and onto for any $S\subseteq\mathbb{Z}$ of n consecutive integers and for any $n>0$.

Proof: Obvious from the definition of f_n. □

Lemma 5.3: For all integers x, y and all positive integers n

$$(f_n(x)+y)-f_n(x+y) = kn$$

for some integer k.

Proof: Let $x = p_1n + r_1$ and $y = p_2n + r_2$, where $0\leq r_1,r_2< n$. From this we get that

$$(f_n(x)+y)-f_n(x+y) = (x \bmod n)+y-((x+y) \bmod n) =$$

$$r_1 + p_2n + r_2 - ((r_1+r_2) \bmod n) = (p_2+d)n \text{ where } d\in\{0,1\}.$$

Therefore, $(f_n(x)+y)-f_n(x+y)$ is a multiple of n. □

Let

$$\sigma_k = \sum_{i=2}^{k}(k_i-l_i), \quad \text{for} \quad 1\leq k< m \tag{6}$$

where m, k_i, l_i are the subscripts appearing in (4) and (5). In figure 5.2, σ_k denotes the distance of the variables appearing under Q_k from those appearing under Q_1. Clearly it is $\sigma_1=0$. Using

f_n and σ_k, (5) can be rewritten as follows:

$$P(z_{1,0}^{(n)}, z_{1,1}^{(n)}, \ldots, z_{1,n_1-1}^{(n)}, z_{2,0}^{(n)}, z_{2,1}^{(n)}, \ldots, z_{2,n_2-1}^{(n)}, \ldots, z_{m,0}^{(n)}, z_{m,1}^{(n)}, \ldots, z_{m,n_m-1}^{(n)}, \ldots) \wedge$$

$$\bigwedge_{j=1}^{m-1} Q_j(z_{j,k_j}^{(f_{n+1}(0+\sigma_j))}, z_{j+1,l_{j+1}}^{(f_{n+1}(0+\sigma_j))}) \wedge \bigwedge_{j=1}^{m-1} Q_j(z_{j,k_j}^{(f_{n+1}(1+\sigma_j))}, z_{j+1,l_{j+1}}^{(f_{n+1}(1+\sigma_j))}) \wedge$$

$$\ldots \wedge \bigwedge_{j=1}^{m-1} Q_j(z_{j,k_j}^{(f_{n+1}(n+\sigma_j))}, z_{j+1,l_{j+1}}^{(f_{n+1}(n+\sigma_j))}) \wedge \ldots \rightarrow$$

$$P(z_{1,1}^{(0)}, z_{1,2}^{(0)}, \ldots, z_{1,n_1}^{(0)}, z_{2,1}^{(0)}, z_{2,2}^{(0)}, \ldots, z_{2,n_2}^{(0)}, \ldots, z_{m,1}^{(0)}, z_{m,2}^{(0)}, \ldots, z_{m,n_m}^{(0)}, \ldots) \qquad (7)$$

In light of Lemma 5.2 it is obvious that the above formula is equivalent to (5). We have only regrouped the predicates in the antecedent of the statement. The superscripts of the variables in each group in (7) are of the form $f_{n+1}(r+\sigma_j)$, with $0 \leq r \leq n$. For every such r, the group corresponding to it will be called the r-th *group* of α_n.

The following two lemmas give some interesting properties of the α-graph of some simple recursive formula α.

Lemma 5.4: Consider two variables $z_{i,j}^{(n)}$, $z_{i',j'}^{(n')}$, with $(n-j) \geq (n'-j')$, coming from the same root-to-leaf path of the dynamic α-graph of some simple recursive statement α. The two variables are connected in the dynamic α_t-graph, $t \geq 0$, if and only if the following hold:

(a) $(n-j)-(n'-j')=k(t+1)$ for some positive integer k

which says that the distance of the two variables in the α-graph is a multiple of $(t+1)$.

(b) $(n'-j'), (n-j) \leq t$

which means that both variables <u>do</u> appear in α_t.

Furthermore, the weight of the path connecting them in the dynamic α_t-graph is k, the coefficient of $(t+1)$ in (a).

Proof: It is easy to see that condition (b) holds. For (a) we have the following. From (7) or (5), we see that every consequent variable $z_{i,j}^{(0)}$ is the tail of a directed edge in the dynamic α_t-graph whose head is $z_{i,j-1}^{(t)}$ and that $z_{i,j}^{(0)}$ is the only variable with this property. Letting $j'' = (n'-j')$ we see that the tail of the directed edge whose head is $z_{i,j''}^{(n')}$ is $z_{i,j''-1}^{(n'+t)} = z_{i,j''}^{(n'+t+1)}$.

Using an easy induction we can conclude that if there is a path from $z_{i,j}^{(n)}$ to $z_{i,j}^{(n')}$ it has to be $z_{i,j}^{(n)} = z_{i,j}^{(n'+k(t+1))}$ for some positive integer k. It is clear that in that case, the weight of the path is k. This shows that (a) is a necessary condition for such a path to exist. The inverse part has a similar easy proof and will not be given here. Note that in the case that $(n-j) = (n'-j')$ the two variables are the same and therefore they are obviously connected in the dynamic α_t-graph by a path of weight $k=0$. □

5.1. SUFFICIENCY OF THE CONDITION

In order for some expansion of α to be redundant regardless of the contents of the relations involved, the antecedent of that expansion should be more restrictive than the antecedent of some previous expansion.

Lemma 5.5: Let α be a simple recursive statement such that the α-graph is connected and has no non-zero weight cycles. Let $\bar{\pi}$, $\bar{\pi} \geq 1$, be the maximum weight of any path in the α-graph. Then, α_n is redundant for all $n \geq \bar{\pi}$.

Proof: Consider $\alpha_{\bar{\pi}+k}$, where $k \geq 0$. Let $Q(z_{1,k_1}^{(j)}, z_{2,k_2}^{(j)}, \cdots, z_{m,k_m}^{(j)})$ be a literal in the antecedent of $\alpha_{\bar{\pi}+k}$, with Q a non-recursive predicate, which in light of Lemma 5.1 first appeared in the j-th expansion. In the above, without loss of generality, we have chosen the root-to-leaf paths where the variables belong to be the ones numbered from 1 to m. Clearly it is $0 \leq j \leq \bar{\pi}+k$. Consider a node $z_{p,0}^{(0)}$ in the α-graph whose distance from some other node in the graph is $\bar{\pi}$ (i.e. there is a path in the graph starting at that node whose weight is the maximum possible). Let r be the distance of this node from any of the $z_{i,k_i}^{(0)}$'s of the predicate above. The distance from all of them is the same, because the distance between any two $z_{i,k_i}^{(0)}$'s is 0 (they are connected by an undirected edge labeled Q) and the graph contains no non-zero weight cycles. So, any two paths between two nodes have the same weight.

We will now attempt to make a substitution of the variables in the $(\bar{\pi}+k)$-th expansion of α that will transform its antecedent so that it contains all the conjunctive formulas of the

antecedent of the $(\bar{\pi}+k-1)$-th expansion plus a few more. This will prove that $\alpha_{\bar{\pi}+k}$ is not going to produce any new tuples, since it will have a more restrictive antecedent that $\alpha_{\bar{\pi}+k-1}$. Let the substitution θ be:

$$\begin{cases} z_{i,k_i}^{(n)} \rightarrow z_{i,k_i}^{(n-1)} & \forall\, n, \quad r+1 \leq n \leq \bar{\pi}+k \\ z_{i,k_i}^{(r)} \rightarrow z_{i,k_i}^{(\bar{\pi}+k)} & \end{cases}$$

All the other variables remain the same. Clearly, each variable maps to a different variable. It is easy to prove that

$$k_i \leq r$$

because otherwise we would get a path in the graph that has weight greater than $\bar{\pi}$. Hence, looking at (2) also, we can see that the above transformation is meaningful, i.e. it affects only antecedent variables. Note that for $i=p$, i.e. for the variables that are created in subsequent expansions of the dynamic root-to-leaf path where a maximum weight path starts, we have that $r=k_i$. Since $x_i^{(r-k_i)} = z_{i,k_i}^{(r)}$ we can see that in this case all the antecedent variables will change, i.e. the transformation will go all the way up to $x_i^{(0)}$ before returning to $x_i^{(\bar{\pi}+k)}$. We will have to distinguish three cases now.

a) $j < r$

In this case no variable in $Q(z_{1,k_1}^{(j)}, z_{2,k_2}^{(j)}, \cdots, z_{m,k_m}^{(j)})$ changes, so the predicate maps to itself.

b) $j > r$

Now all the variables in $Q(z_{1,k_1}^{(j)}, z_{2,k_2}^{(j)}, \cdots, z_{m,k_m}^{(j)})$ change to their predecessors, i.e. $z_{i,k_i}^{(j)}$ to $z_{i,k_i}^{(j-1)}$.

c) $j = r$

In this case all the variables change as well, but the new superscript is $\bar{\pi}+k$, i.e. $z_{i,k_i}^{(r)}$ maps to $z_{i,k_i}^{(\bar{\pi}+k)}$.

From Lemma 5.1 we know that every non-recursive predicate appearing in the antecedent of α, appears also in the antecedent of $\alpha_{\bar{\pi}+k-1}$ once for every number in $\{0,1,...,\bar{\pi}+k-1\}$ in the superscript of its variables. Therefore, with the above transformation we have shown that all the non-recursive predicates in $\alpha_{\bar{\pi}+k-1}$ are indeed "covered" by corresponding predicates in $\alpha_{\bar{\pi}+k}$.

All we have to prove now is that the recursive predicate in the antecedent of $\alpha_{\bar{\pi}+k}$ changes to that of the $\alpha_{\bar{\pi}+k-1}$ as well. But this is true because of Lemma 5.1. Consider a variable $z_{i,l_i}^{(\bar{\pi}+k)}$. The distance r of $z_{i,l_i}^{(0)}$ from $z_{p,0}^{(0)}$ is definitely no greater than $\bar{\pi}$, since $\bar{\pi}$ is the maximum weight of any path in the graph. Furthermore r cannot be equal to $\bar{\pi}$ either, because in that case, since $z_{i,l_i}^{(0)}$ is in the antecedent of α, there has to be an edge going out of it, and therefore there should have been a path of weight at least $\bar{\pi}+1$. Therefore, $r+1 \leq \bar{\pi}+k$. This implies that $z_{i,l_i}^{(\bar{\pi}+k)}$ maps to $z_{i,l_i}^{(\bar{\pi}+k-1)}$. From Lemma 5.1 we may conclude that the substitution of the variables maps the recursive predicate of $\alpha_{\bar{\pi}+k}$ to that of $\alpha_{\bar{\pi}+k-1}$ as well.

Since the antecedent of the $(\bar{\pi}+k-1)$-th expansion is equal to some part of that of the $(\bar{\pi}+k)$-th expansion, the $(\bar{\pi}+k)$-th expansion is redundant. □

Lemma 5.6: Let α be a simple recursive statement. Consider α_s and α_t, $0 \leq s < t$, such that α_t is more restrictive than α_s. Then the maximum weight of any path in both the dynamic α_s- and α_t-graph is 1.

Proof: Since α_t is more restrictive than α_s, the graph of the redundant expansion α_t contains a subgraph which is isomorphic to the graph of some previous expansion. The isomorphism is label preserving (i.e. any edge of the static subgraph corresponding to a predicate Q maps to an edge corresponding to the same predicate) and all the consequent variables map to themselves (because the consequent is the same in all expansions). Consider a single component G of the α_s-graph. Let H be the subgraph of G, consisting of those nodes in G that the corresponding variables do not map to themselves in the isomorphism from α_t to α_s. It is clear that there are only antecedent variables in H (otherwise we would not be able to change them)

and therefore we do not have any directed edges with their head in H. Hence, H is connected with the rest of G, say H', as shown in figure 5.3.

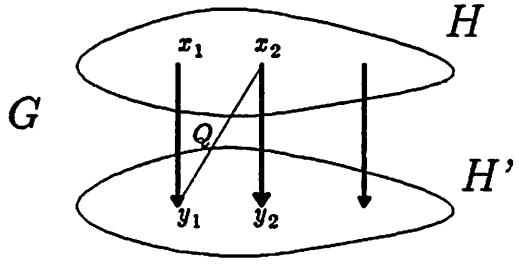

Fig. 5.3 : General form of the graph of some redundant expansion

It is also true that there are no tails of directed edges in H'. We can show this as follows. From (7) we see that no directed edge appears in the graph of more than one expansions. In the graph of every expansion, each consequent variable is the head of some directed edge, but the tail of the edge is always different. This is because of restrictions (3) and (4) in the definition of a simple recursive statement, and can be seen from the form of θ_n in (3) above. The isomorphism between the α_s-graph and some subgraph of the α_t-graph has to preserve the consequent variables as well as the directed edges. Therefore, the tail of every directed edge has to change in the isomorphism. Since H' is the subset of nodes/variables that do not change in the isomorphism, we may conclude that there can be no directed edge with its tail in H'. Hence, every single component of the α_s-graph (and the α_t-graph) consists of two subgraphs (either one of which may be empty) with static edges only, and we can have directed edges only from the one subgraph to the other. Therefore, the maximum weight of any path in both of the dynamic α_s- and α_t-graph is 1. □

Lemma 5.7: Let α be a simple recursive statement. If the α-graph contains a path of weight L, then the α_n-graph contains a path of weight $\lfloor \frac{n+L}{n+1} \rfloor$ †.

Proof: Let the statement in (4) denote a path in the α-graph with weight L as it is shown in figure 5.2. It is clear from the figure that the weight of the path is equal to

† By $\lfloor x \rfloor$ we denote the smallest integer greater than or equal to x.

$$L = k_1 + \sum_{i=2}^{m-1}(k_i - l_i) + n_m - l_m \tag{8}$$

Consider the n-th expansion of α as given in (7). We will first show that the r-th group in (7) forms a path in the α_n-graph. To accomplish this we will show that for every Q_{i-1}, Q_i the two variables $z_{i,l_i}^{(f_{n+1}(r+\sigma_{i-1}))}$, $z_{i,k_i}^{(f_{n+1}(r+\sigma_i))}$ are connected with a path in the dynamic α_n-graph; we do not exclude the possibility that the path has zero weight, i.e. that the two variables are the same.

Because of Lemma 5.3, we know that

$$(f_{n+1}(r+\sigma_{i-1}) - l_i) - (f_{n+1}(r+\sigma_i) - k_i) = k(n+1) \tag{9}$$

for some integer k. Furthermore from the definition of f_{n+1} we have that

$$f_{n+1}(r+\sigma_{i-1}) - l_i < n+1 \quad \text{and} \quad f_{n+1}(r+\sigma_i) - k_i < n+1.$$

Having shown the two things above, we have actually satisfied the conditions of Lemma 5.4 and therefore the given variables are connected in the dynamic α_n-graph by a path of weight k (as given in (9)). Moreover, Lemma 5.4 also implies that there is a path of directed edges of weight $\left\lfloor \frac{k_1 - f_{n+1}(r) + n}{n+1} \right\rfloor$ leading into $z_{1,k_1}^{(r)}$. To see this, consider $z_{1,k_1}^{(f_{n+1}(r))}$ (which is actually $z_{1,k_1}^{(r)}$, since $0 \leq r \leq n$) and $z_{1,0}^{(n)}$. There is no guarantee that there is a path from the latter to the former; in fact, such a path will exist if and only if $(n-0)-(f_{n+1}(r)-k_1)$ is a multiple of $n+1$, according to Lemma 5.4. However, instead of $z_{1,0}^{(n)}$ we may take $z_{1,0}^{(n')}$, with $0 \leq n' < n$ such that $(n'-0)-(f_{n+1}(r)-k_1)$ is a multiple of $n+1$, in which case there will be a path from this variable to $z_{1,k_1}^{(r)}$. Clearly, the weight of this path will be given by the formula mentioned above. Likewise we may show that there is a path of directed edges of weight $\left\lfloor \frac{n_m - l_m + f_{n+1}(r+\sigma_{m-1})}{n+1} \right\rfloor$ going out of $z_{m,l_m}^{(f_{n+1}(r+\sigma_{m-1}))}$. So, there is a path in the α_n-graph of weight

$$L_n = \frac{1}{n+1}\Big[f_{n+1}(r)+k_2-l_2-f_{n+1}(r+\sigma_2) + f_{n+1}(r+\sigma_2)+k_3-l_3-f_{n+1}(r+\sigma_3) + \cdots\cdots\cdots\cdots$$

$$f_{n+1}(r+\sigma_{m-2})+k_{m-1}-l_{m-1}-f_{n+1}(r+\sigma_{m-1})\Big] +$$

$$\lfloor\frac{k_1 - f_{n+1}(r) + n}{n+1}\rfloor + \lfloor\frac{n_m - l_m + f_{n+1}(r+\sigma_{m-1})}{n+1}\rfloor \Longleftrightarrow$$

$$L_n = \frac{1}{n+1}\Big[f_{n+1}(r)+ \sum_{i=2}^{m-1}(k_i-l_i)-f_{n+1}(r+\sigma_{m-1})\Big] +$$

$$\lfloor\frac{k_1 - f_{n+1}(r) + n}{n+1}\rfloor + \lfloor\frac{n_m - l_m + f_{n+1}(r+\sigma_{m-1})}{n+1}\rfloor \qquad (10)$$

From Lemma 5.2 we know that there is some $0 \leq r \leq n$ such that $\frac{k_1 - f_{n+1}(r) + n}{n+1}$ is integer. So, (10) can be written as

$$L_n = \lfloor\frac{k_1 + \sum_{i=2}^{m-1}(k_i-l_i)+n_m-l_m+n}{n+1}\rfloor$$

which in turn, considering the formula for L, gives

$$L_n = \lfloor\frac{n + L}{n + 1}\rfloor \qquad \square$$

Theorem 5.1: Let α be a simple recursive statement. If the α-graph contains no cycle of non-zero weight then α is bounded. In that case a tight upper bound is equal to the maximum weight of any path in the α-graph.

Proof: Suppose that the maximum weight of any path in the α-graph is $\bar{n}$. It is clear that each component in the α-graph expands independently from all the others, since there can never be any interaction among them. From Lemma 5.5 we have that every component's $\bar{n}$-th expansion is redundant. Since the components do not interact, we conclude that the whole graph's $\bar{n}-$th expansion is redundant. Hence, $\bar{n}$ is an upper bound on α's rank.

The upper bound that we have just given is tight. This means that $\alpha_{\bar{n}-1}$ is not properly more restrictive than any of the previous expansions. Lemma 5.7 shows that in the α_n-graph there exists a path of weight

$$L_n = \lfloor\frac{n + \bar{n}}{n + 1}\rfloor$$

This guarantees that the α_n-graph has at least one path of weight greater than 1, for all

$n \leq \bar{n}-2$. Assume that some α_s of these expansions, with $0 \leq s \leq \bar{n}-2$ is strictly less restrictive than α_t, $s \leq t$. Because of Lemma 5.6, the α_s-graph will be of the form of figure 5.3. Consider the path $(z_1 \rightarrow x \rightarrow z_2 \rightarrow y)$ in the α_s-graph. Our previous discussion guarantees that such a path, of weight greater than 1, does exist. Since the α_s-graph is isomorphic to a subgraph of the α_t-graph, we may conclude that such an isomorphic path, say of the form $(z_1' \rightarrow x \rightarrow z_2' \rightarrow y)$, exists in the α_t-graph as well. In order for this to be possible the original α-graph must be of the form shown in figure 5.3a.

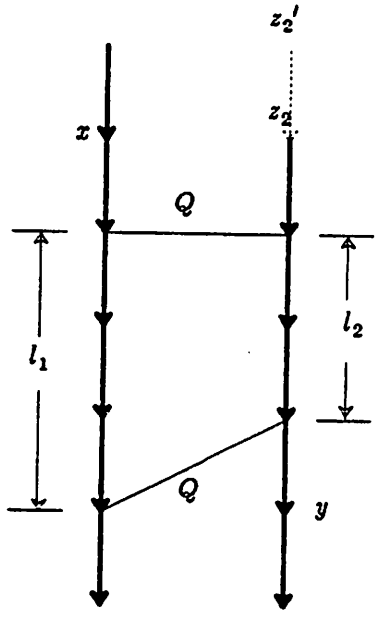

Fig. 5.3a : Graph of statement leading into redundant expansion with maximum path-weight greater than 1.

Since z_2 and z_2' are tails of directed edges leading to y (each one at a different expansion, of course) they have to belong to the same root-to-leaf path. Therefore, since x is adjacent to these two different variables the distances l_1 and l_2 in figure 5.3a have to be different as well. But this implies that there exists a cycle of non-zero weight in the α-graph which contradicts our initial hypothesis. Having obtained a contradiction we may conclude that the first expansion that can have a graph isomorphic to a subgraph of the graph of some subsequent expansion is the $(\bar{n}-1)$-th. Therefore, the first expansion that can be redundant is the $\bar{n}$-th. □

We will now try to illustrate the proof given above with a few examples. Consider formula γ from Section 3:

$$\gamma: \qquad P(z) \wedge Q(z) \wedge R(y) \rightarrow P(y)$$

Figure 5.4 shows that the γ-graph contains no cycles at all (excluding that formed through the implicitly assumed negative edge, which again is of weight zero).

Fig. 5.4 : The γ-graph

Furthermore the maximum weight of any path in the graph is 1. In our discussion in Section 3, we have concluded that one step of the iteration process is enough to derive all possible tuples in P, which is in perfect agreement with Theorem 5.1.

The fact that the rank of γ is bound by 1, can be seen from the first expansion of γ, which is

$$\gamma_1: \qquad P(z') \wedge Q(z') \wedge R(z) \wedge Q(z) \wedge R(y) \rightarrow P(y)$$

If we substitute z for z' and z' for z in γ_1, the antecedent of γ_1 becomes equal to the antecedent of γ with some additional literals conjuncted to it. The antecedent of γ_1 being strictly more restrictive than that of γ, implies that any tuple derived from the former is also derived from the latter. Thus γ_1 need never be considered.

The same conclusion could be drawn, by looking at the γ- and the γ_1-graphs, as they appear in figures 5.4 and 5.5 respectively.

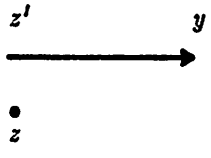

Fig. 5.5 : The γ_1-graph

We can see that the γ-graph is isomorphic to a subgraph of the γ_1-graph. The isomorphism preserves the consequent variables as well as the edges of the dynamic γ-graph. As for the antecedent variables the isomorphism maps them according to the substitution mentioned before, that makes the antecedent of γ part of that of γ_1.

We will now consider a much more complicated example. Consider the simple recursive formula α:

$$\alpha: \quad P(u_1,u_2,u_4,u_4,y) \wedge Q(u_1,u_2) \wedge R(u_2,u_3,x) \wedge S(w,z) \wedge T(v) \rightarrow P(v,w,x,y,z)$$

The α-graph appears in figure 5.6.

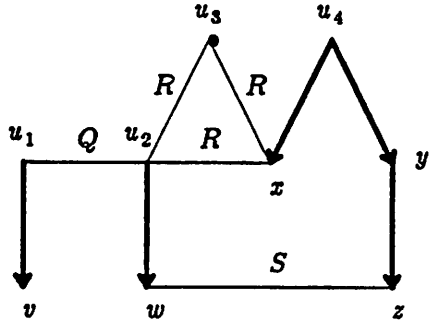

Fig. 5.6 : The α-graph

All the cycles in the α-graph have zero weight, including those formed through the negative directed edges, which according to our convention are not drawn (going along a negative edge can be thought of as going along a positive edge in the opposite direction and negating the weight). Hence, according to our theorem, α is bounded. The maximum weight of any path in the graph being 2, we may conclude that α_2 is redundant, i.e. two steps are enough in the iterative process to get the final result for P.

This becomes apparent if we look at the α_1- and the α_2-graphs. They are shown in figures 5.7 and 5.8 respectively.

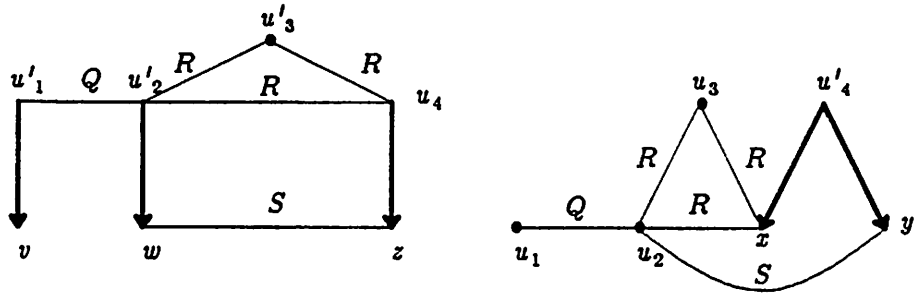

Fig. 5.7 : The α_1-graph

The α_1-graph in figure 5.7 shows two (connected) components, instead of one that initially appeared in the α-graph. Furthermore, the latter is not isomorphic to any subgraph of the former, which implies that there are some instances of the relations involved in α that will make α_1 produce some tuples that are not produced by α. Hence, α_1 is necessary.

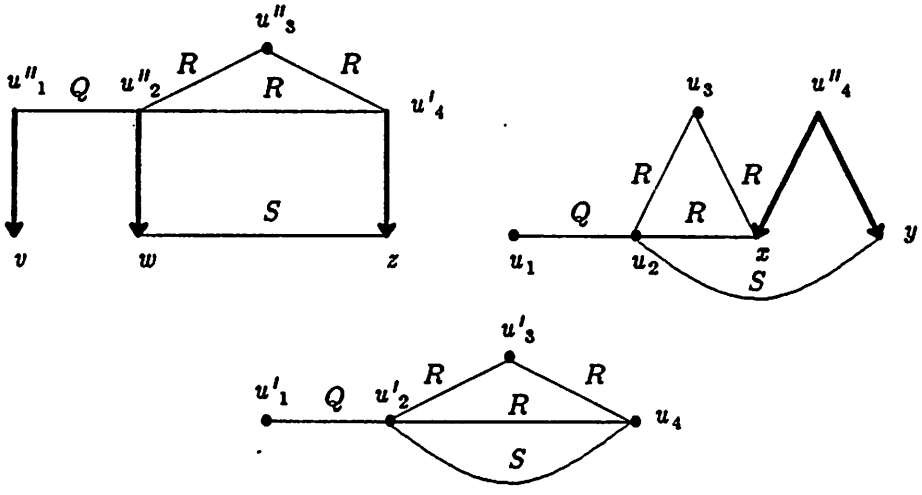

Fig. 5.8 : The α_2-graph

On the other hand the α_2-graph in figure 5.8 has three components, one more than the α_1-graph. Two of these components are isomorphic to those in the α_1-graph. Moreover, the isomorphism has all the desired properties, i.e. it maps consequent variables to themselves and preserves the labeling of the static edges. For α_1 and α_2 this means that changing the antecedent variable names in the latter appropriately, its antecedent becomes strictly more restrictive than that of the former. The rank of α is bounded by 2, therefore α_2 is not necessary.

Notice that the number of components increased in the first example with γ as well. In fact, this is true for all graphs free of non-zero weight cycles. If, for example, the graph of the initial statement is connected, then each expansion comes with one more component than the previous one. Considering (7), every group of literals, as presented there, forms a component. At some point, we get a component that contains no directed edges, i.e. no consequent variables. This expansion is exactly the first one that is redundant and it determines the bound on the rank of the initial statement.

Also notice that the last expansion that is significant (regarding the production of new tuples), is the first one with the maximum weight of any path in its graph being 1. This is proved in Lemmas 5.6 and 5.7 and Theorem 5.1 and may be seen in both the γ- and the α_1-graphs above.

5.2. NECESSITY OF THE CONDITION

We will now prove the inverse of Theorem 5.1. However, before doing so, we must prove the following very important lemma.

Lemma 5.8: Let α be a recursive statement.

a) If α is bounded then its k-th expansion is bounded as well, for all $k \geq 0$.

b) If for some $k \geq 0$, the k-th expansion of α is bounded then α is bounded as well.

Proof: We are going to prove the two problems separately. In what follows we will be using the fact that the l-th expansion of the k-th expansion of α is equal to the k-th expansion of the l-th expansion of α. Furthermore, each one of those is equal to the m-th expansion of α, where $m = (k+1)(l+1)-1$. In other words $(\alpha_k)_l = (\alpha_l)_k = \alpha_{(k+1)(l+1)-1}$.

a) Let α be a bounded recursive statement, with bound $\bar{\pi}$. This means that α_π is more restrictive than α_l, for some $0 \leq l < \bar{\pi}$. Consider α_k for some $k \geq 0$. We will prove that $\bar{\pi}$ is an upper bound on the rank of α_k also, and in particular we are going to show that $(\alpha_k)_\pi$ is more restrictive than $(\alpha_k)_l$. Using the fact mentioned in the beginning of the proof, it suffices to prove that $(\alpha_\pi)_k$ is more restrictive than $(\alpha_l)_k$.

Let S_k^l and S_k^π be the set of the tuples produced in the recursive predicate of α by $(\alpha_l)_k$ and $(\alpha_\pi)_k$ respectively. We will prove by induction that $S_k^\pi \subseteq S_k^l$ for all $k \geq 0$.

Basis: For $k=0$ this is obvious since it is given that α_π is more restrictive than α_l.

Induction Step: Assume that $S_j^{\pi+1} \subseteq S_j^l$ is true for all j-th expansions of α_l and α_π less than k, i.e. $0 \leq j < k$. We are going to show that it is also true that $S_k^{\pi+1} \subseteq S_k^l$. Having in mind the equivalence between the k-th expansion and the $(k+1)$-th application of some formula, we can see that S_i^l is the set of tuples produced by α_l if applied to an instance of the recursive predicate equal to S_{k-1}^l. Likewise S_k^π is the set of the tuples produced by α_π if applied to an instance of the recursive predicate equal to S_{k-1}^π. By the induction hypothesis we have that $S_{k-1}^\pi \subseteq S_{k-1}^l$. Hence, since α_π is more restrictive than α_l and is to on a subset of the tuples to which α_l is applied, we

can conclude that $S_k^{\bar{\pi}} \subseteq S_k^l$.

This does not depend on the initial contents of the recursive predicate, so we may conclude that $(\alpha_{\bar{\pi}})_k$ is more restrictive than $(\alpha_l)_k$, or equivalently $(\alpha_k)_{\bar{\pi}}$ is more restrictive than $(\alpha_k)_l$. Therefore, $\bar{\pi}$ is an upper bound on the rank of α_k. Our initial choice of k was arbitrary, so our conclusion is that if a recursive statement α is bounded then α_k is bounded also, for all $k \geq 0$.

b) Let α_k be the k-th expansion of a recursive statement α and suppose that α_k is bounded, say with bound $\bar{\pi}$. This means that $(\alpha_k)_{\bar{\pi}}$ is more restrictive than $(\alpha_k)_l$ for some $0 \leq l < \bar{\pi}$. However, as we have mentioned earlier, the former is equal to $\alpha_{(k+1)(\bar{\pi}+1)-1}$ and the latter is equal to $\alpha_{(k+1)(l+1)-1}$. Therefore, since $l \leq \bar{\pi}$, we have that $(k+1)(\bar{\pi}+1)-1$ is an upper bound on the rank of α.

Note that in both parts (a) and (b) above the upper bounds we have given are not tight in general, but this was not our concern. □

Consider a typical cycle in the α-graph. The part of the statement corresponding to the cycle is similar to (4) with one more predicate added to form the cycle and is repeated below for convenience:

$$P(z_{1,0}^{(0)}, z_{1,1}^{(0)}, \ldots, z_{1,n_1-1}^{(0)}, z_{2,0}^{(0)}, z_{2,1}^{(0)}, \ldots, z_{2,n_2-1}^{(0)}, \ldots, z_{m,0}^{(0)}, z_{m,1}^{(0)}, \ldots, z_{m,n_m-1}^{(0)}, \ldots) \wedge$$

$$Q_1(z_{1,k_1}^{(0)}, z_{2,l_2}^{(0)}) \wedge Q_2(z_{2,k_2}^{(0)}, z_{3,l_3}^{(0)}) \wedge \ldots \wedge Q_m(z_{m,k_m}^{(0)}, z_{1,l_1}^{(0)}) \wedge \ldots \rightarrow$$

$$P(z_{1,1}^{(0)}, z_{1,2}^{(0)}, \ldots, z_{1,n_1}^{(0)}, z_{2,1}^{(0)}, z_{2,2}^{(0)}, \ldots, z_{2,n_2}^{(0)}, \ldots, z_{m,1}^{(0)}, z_{m,2}^{(0)}, \ldots, z_{m,n_m}^{(0)}, \ldots) \qquad (11)$$

The graph corresponding to the cycle above can be seen in figure 5.9.

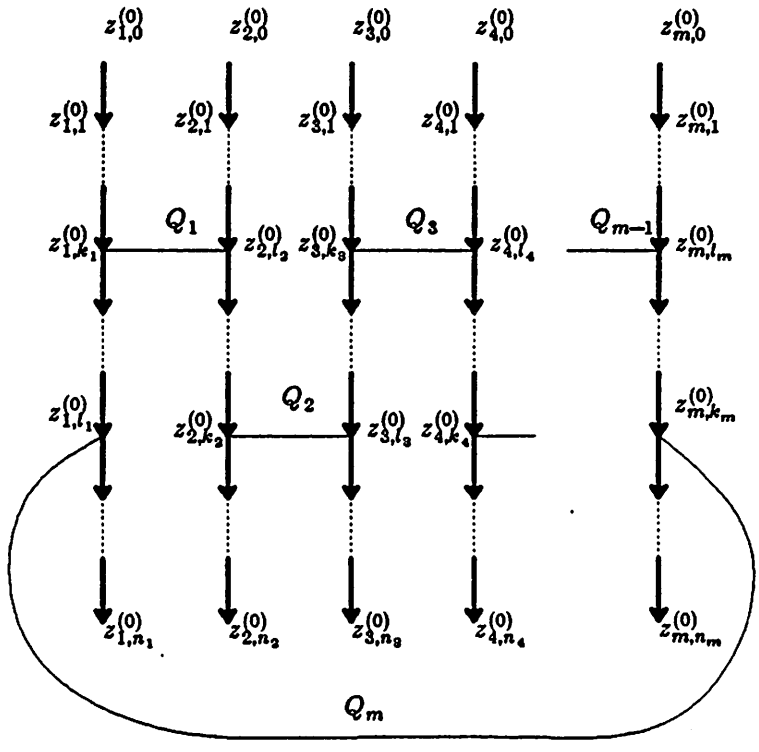

Fig. 5.9 : Typical cycle in the α-graph

Similarly to (7), α_n will be

$$P(z^{(n)}_{1,0},z^{(n)}_{1,1},\ldots,z^{(n)}_{1,n_1-1},z^{(n)}_{2,0},z^{(n)}_{2,1},\ldots,z^{(n)}_{2,n_2-1},\ldots,z^{(n)}_{m,0},z^{(n)}_{m,1},\ldots,z^{(n)}_{m,n_m-1},\ldots)\wedge$$

$$\bigwedge_{j=1}^{m} Q_j(z^{(f_{n+1}(0+\sigma_j))}_{j,k_j},z^{(f_{n+1}(0+\sigma_j))}_{j+1,l_{j+1}})\wedge\bigwedge_{j=1}^{m} Q_j(z^{(f_{n+1}(1+\sigma_j))}_{j,k_j},z^{(f_{n+1}(1+\sigma_j))}_{j+1,l_{j+1}})\wedge$$

$$\ldots\wedge\bigwedge_{j=1}^{m} Q_j(z^{(f_{n+1}(n+\sigma_j))}_{j,k_j},z^{(f_{n+1}(n+\sigma_j))}_{j+1,l_{j+1}})\wedge\ldots\rightarrow$$

$$P(z^{(0)}_{1,1},z^{(0)}_{1,2},\ldots,z^{(0)}_{1,n_1},z^{(0)}_{2,1},z^{(0)}_{2,2},\ldots,z^{(0)}_{2,n_2},\ldots,z^{(0)}_{m,1},z^{(0)}_{m,2},\ldots,z^{(0)}_{m,n_m},\ldots) \qquad (12)$$

where in fact it is $z^{(f_{n+1}(n+\sigma_m))}_{m+1,l_{m+1}} = z^{(f_{n+1}(n+\sigma_m))}_{1,l_1}$.

The following two lemmas could have been combined into one general lemma, but are presented here separately for clarity.

Lemma 5.9: Let α be a simple recursive statement. If the α-graph contains a cycle of weight $n+1$ for some $n \geq 0$ then the α_n-graph contains a cycle of weight 1.

Proof: Take a cycle of weight $n+1$ in the graph corresponding to (11). This means that

$$\sum_{i=1}^{m} k_i - \sum_{i=1}^{m} l_i = n+1 \tag{13}$$

Consider now the cycle's n-th expansion. This will be of the form given in (12). We will prove that the r-th group in (12), together with some dynamic edges of the α_n-graph, forms a cycle of weight 1, for all $0 \leq r \leq n$. Similarly to the proof of Lemma 5.7 we can show that for every Q_{i-1}, Q_i the two variables $z_{i,l_i}^{(f_{n+1}(r+\sigma_{i-1}))}$, $z_{i,k_i}^{(f_{n+1}(r+\sigma_i))}$ are connected with a path in the dynamic α_n-graph. All we have to show now, to prove that the r-th group forms a cycle, is that $z_{1,l_1}^{(f_{n+1}(r+\sigma_m))}$, $z_{1,k_1}^{(r)}$ are connected as well. However because of (13) and the definition of σ_m in (6) we can see that it is

$$z_{1,l_1}^{(f_{n+1}(r+\sigma_m))} = z_{1,l_1}^{(f_{n+1}(r+n+1-k_1+l_1))} = z_{1,l_1}^{(f_{n+1}(r-k_1+l_1))}$$

For these two variables now, that is $z_{1,l_1}^{(f_{n+1}(r-k_1+l_1))}$ and $z_{1,k_1}^{(r)}$, we can easily show that they satisfy the conditions of Lemma 5.4 and therefore they are connected in the dynamic α_n-graph.

As a last step in our proof we have to show that the weight of the cycle is 1. Similarly to Lemma 5.7, the weight L of the cycle comes up to be

$$\begin{aligned} L = \frac{1}{n+1}\Big[& f_{n+1}(r)+k_2-l_2-f_{n+1}(r+\sigma_2) + \\ & f_{n+1}(r+\sigma_2)+k_3-l_3-f_{n+1}(r+\sigma_3) + \\ & \ldots\ldots\ldots\ldots\ldots\ldots \\ & f_{n+1}(r+\sigma_m)+k_1-l_1-f_{n+1}(r)\Big] \iff \\ L = \frac{1}{n+1}\Big[& f_{n+1}(r)+\sum_{i=1}^{m}(k_i-l_i)-f_{n+1}(r)\Big] \iff \\ L = \frac{1}{n+1}(n+1) = 1 & \end{aligned}$$

This concludes the proof of Lemma 5.9. □

Lemma 5.10: Let α be a simple recursive statement. If the α-graph contains a cycle of weight 1 then the α_n-graph contains a cycle of weight 1, for all $n \geq 0$.

Proof: We will proceed as in the previous lemma. Consider a cycle of weight 1 in the graph corresponding to (11). We now have that

$$\sum_{i=1}^{m} k_i - \sum_{i=1}^{m} l_i = 1 \tag{14}$$

Regarding the n-th expansion of the cycle we will show that the whole collection of groups in (12), together with some directed edges from the dynamic α_n-graph form a cycle.

The fact that $z_{i,l_i}^{(f_{n+1}(r+\sigma_{i-1}))}$ and $z_{i,k_i}^{(f_{n+1}(r+\sigma_i))}$ are connected, for all $2 \leq i \leq m$ and for all $0 \leq r \leq n$ is proved in exactly the same way as the corresponding result in Lemma 5.9.

Consider now the last variable in the last predicate of the r-th group $z_{1,l_1}^{(f_{n+1}(r+\sigma_m))} = z_{1,l_1}^{(f_{n+1}(r+1-k_1+l_1))}$ (the equality coming from (6) and (14)) and the first variable in the first predicate of the $(r+1)$-th group $z_{1,k_1}^{(r+1)}$, for all $0 \leq r < n$. We will show that these are connected as well.

Lemma 5.3 is directly applicable for f_{n+1} with $x=r+1$ and $y=l_1-k_1$. Therefore, we have

$$\left((r+1)-k_1+l_1 \right) - \left(f_{n+1}(r+1-k_1+l_1) \right) = k(n+1)$$

for some integer k. This satisfies condition (a) of Lemma 5.4, if we take into account that $f_{n+1}(r+1) = r+1$ for the range of r's we are considering. Condition (b) is easily proved from the definition of f_{n+1}. Therefore, from Lemma 5.4 we can conclude that these variables are connected also.

The final step is to show that the last variable of the last predicate of the n-th group $z_{1,l_1}^{(f_{n+1}(n+\sigma_m))} = z_{1,l_1}^{(f_{n+1}(n+1-k_1+l_1))} = z_{1,l_1}^{(f_{n+1}(l_1-k_1))}$ is connected with the first variable of the first predicate of the 0-th group $z_{1,k_1}^{(0)}$. We have again used (6) and (14) together with the definition of f_{n+1}, to obtain the equalities of the variables. Lemma 5.3 can be applied again here for f_{n+1} with $x=0$ and $y=l_1-k_1$, to satisfy condition (a) of Lemma 5.4. Again, condition (b) is trivially provable and therefore, by Lemma 5.4, these two variables are connected as well.

Having proved the existence of a cycle in the α_n-graph we are now going to prove that its weight is equal to 1. The steps are similar to those of the corresponding part of Lemma 5.9.

Assuming that L is the weight of the cycle, we have

$$L = \frac{1}{n+1}\sum_{r=0}^{n}[\, f_{n+1}(r)+k_2-l_2-f_{n+1}(r+\sigma_2) +$$

$$f_{n+1}(r+\sigma_2)+k_3-l_3-f_{n+1}(r+\sigma_3) +$$

$$\ldots\ldots\ldots\ldots\ldots\ldots$$

$$f_{n+1}(r+\sigma_m)+k_1-l_1-f_{n+1}(r+1)\,] \iff$$

$$L = \frac{1}{n+1}\sum_{r=0}^{n}[\, f_{n+1}(r)+ \sum_{i=1}^{m}(k_i-l_i)-f_{n+1}(r+1)\,] \iff$$

$$L = \frac{1}{n+1}[\, (n+1)\sum_{i=1}^{m}(k_i-l_i)+f_{n+1}(0)-f_{n+1}(n+1)\,] \iff$$

$$L = \sum_{i=1}^{m}(k_i-l_i) \iff L = 1$$

This concludes the proof of Lemma 5.10. □

Lemma 5.11: Let α be a simple recursive statement. Suppose that the maximum weight of any path in the dynamic α-graph is 1. If there is a cycle of weight 1 in the α-graph then α is not bounded.

Proof: Suppose to the contrary that there is some bounded simple recursive statement α such that the dynamic α-graph has a maximum path-weight equal to 1 and the α-graph contains some cycle of weight 1. The cycle must have the form of figure 5.10.

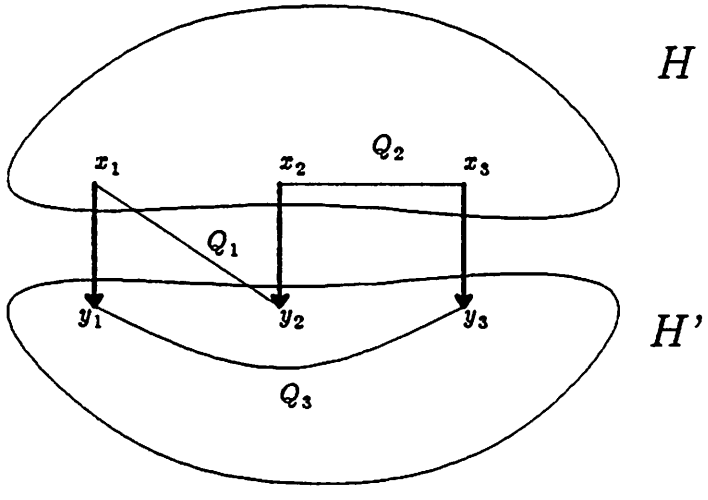

Fig. 5.10 : Typical weight 1 cycle with single dynamic edges

Without loss of generality we may assume that between any two directed edges there is at most one undirected edge, as it appears in figure 5.10. The more general case can be reduced to this special case above in a straightforward way. Because of the properties of the α-graph, its nodes may be divided into two sets H and H', as in figure 5.3. The subgraph H contains all the tails and H' contains all the heads of the directed edges. Furthermore, the fact that the graph contains a cycle of non-zero weight implies that there is at least one undirected edge between a node in H and a node in H'. If not, then every cycle would have an equal number of positive and negative directed edges, and would thus have weight zero. In figure 5.10 Q_1 is such an edge.

We will primarily use the "application" view of α as described in the beginning of Section 5. Let $\bar{\pi}$ be the bound of α. We will find an instance of the relations involved in α that will need at least $\bar{\pi}+1$ iterations to produce all the tuples in the recursive relation, thus coming to a contradiction. Consider the following $\bar{\pi}+2$ distinct constants: $\{a_0,a_1,a_2,...,a_{\bar{\pi}},a_{\bar{\pi}+1}\}$. Suppose that every relation Q whose corresponding undirected edge has both nodes in H, like Q_2 in figure 5.10, contains the tuples $<a_0,a_0>$, $<a_1,a_1>$, ..., $<a_{\bar{\pi}},a_{\bar{\pi}}>$. Likewise assume that every relation Q whose corresponding undirected edge has both nodes in H', like Q_3 in figure 5.10, contains the tuples $<a_1,a_1>$, $<a_2,a_2>$, ..., $<a_{\bar{\pi}+1},a_{\bar{\pi}+1}>$. Finally suppose that every relation Q whose corresponding undirected edge has one node in H and the other in H' contains the tuples $<a_0,a_1>$, $<a_1,a_2>$, ..., $<a_{\bar{\pi}},a_{\bar{\pi}+1}>$. In figure 5.10, edge Q_1 is such an example. For the time being we assume that the first attribute of the relation is the one corresponding to the variable in H. Later we will remove this restriction. As it concerns the recursive predicate P, we assume that initially it contains only one tuple with all its attributes having the same value a_0. With this instance of the relations of the formula we will show that one of the attributes of the recursive predicate will not take the value $a_{\bar{\pi}+1}$ until the $\bar{\pi}+1$-th iteration.

Notice that in the α-graph each directed edge shows how values are produced in one of the attributes of P. Also notice, that there are two nodes in the α-graph for every attribute of P; one in H and another in H'. The attribute mentioned above is identified with the help of the

following new graph.

(i) To every pair of attributes of P, that are connected with a static edge in H', we associate a node in the graph. These nodes will be called the *roots* of the graph.

(ii) To each one of the remaining attributes we also associate a node.

(iii) There is an undirected edge between any two nodes such that the corresponding attributes are likewise connected with a static edge in H. These edges will be called *join* edges.

(iv) Finally, there is a directed edge from a node v_1 to a node v_2 if, in the α-graph, the node corresponding to v_1 in H is connected with a static edge to the node corresponding to v_2 in H'.

As an example, consider the graph shown if figure 5.10a.

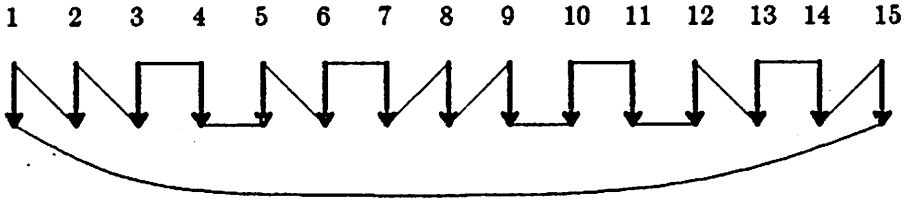

Fig. 5.10a : Another weight 1 cycle with single dynamic edges

The corresponding graph constructed in the way described above appears in figure 5.10b. Nodes (1,15), (4,5), (9,10) and (11,12) are the roots.

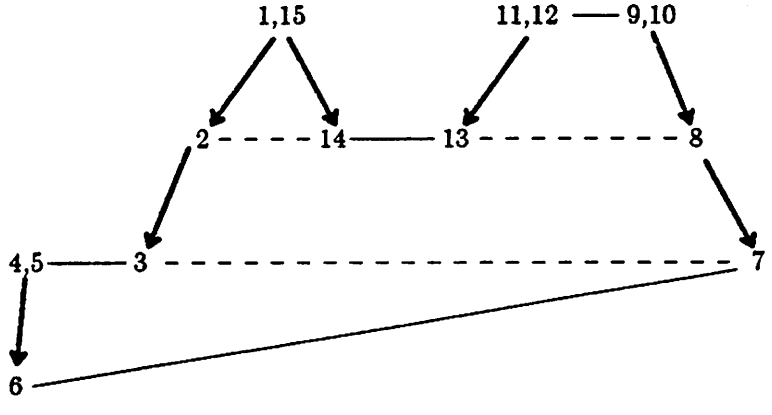

Fig. 5.10b : Graph showing the interaction of attributes

The — edges are not part of the graph. They only show a particular matching between the attributes which will be described shortly. It is not difficult to see that this new graph (or rather

its underlined undirected graph) will always be a cycle. The directed edges show how the values in one attribute (the tail) will affect the values in another attribute (the head) in the next iteration. Only one node is necessary for every pair of attributes that satisfy condition (i) because the contents of the relations are such that every tuple in P will have the same value in both attributes of the pair. Notice that the nodes of (i) (the roots) are exactly the ones with directed in-degree equal to zero. The only case that there are no root nodes is when the α-graph has exactly one directed (and one undirected) edge. In this trivial case the graph constructed as above is a single node with no edges. Then, α represents the transitive closure of the non-recursive relation in α and is clearly unbounded. From now on, we assume that the recursive predicate in α has at least three attributes and therefore the graph does have roots.

Our focus now will be to see, as the iterations proceed, which other pairs of attributes will be getting equal values as well. Each such pair of attributes will be called a *matched* pair. There is an odd number of such nodes to match (there are (arity of P)-2(# of roots) nodes to match and P is of odd arity). So there will be one node that will not be matched. This node will be the one that will be used to contradict the boundedness of α.

We match the nodes inductively as follows:

Stage 1: Each root may be considered a matched pair on its own.

Stage 2: Because the graph is a cycle, each root has at most two children. Match any two nodes as long as they are children of the same root.

Stage k: Imagine that whenever two nodes are matched, they are connected with an undirected *match* edge. Consider any maximal path of match and join edges created in the previous stage as a new root and then proceed as in stage 2 (maximal in the sense that it cannot be extended with more match or join edges). An equivalent way to look at it is that we match nodes that can be reached from one of the original roots using two disjoint paths of the same weight (i.e. again counting only the directed edges).

Applying the above procedure to the graph of figure 5.10b the nodes are matched according to the — edges shown there. As we can see, node 6 is the one that does not get matched. We need that, in general, when the whole process terminates all but one of the non-root nodes are matched. Suppose that we are at stage k and more than one node is unmatched. Take one of the roots formed at the previous stage (that is, a cluster of nodes connected by a match-join path). It is not difficult to see that the graph of the new roots together with the unmatched nodes is again a cycle. So, there must be two directed edges going out of every new root and leading into two unmatched nodes. So the process can only terminate when there is only one unmatched node in the graph.

It is useful to point out here that after we get a maximum match as described above, the unmatched node is connected to its father with a match-join path. Assume that the matching process needs k stages. We will now show what values are obtained in each attribute of the recursive predicate as we iteratively process α. First, consider the attributes that are matched. For each attribute matched at stage i in the above algorithm (including the roots that are viewed as the matched nodes in the first stage) the tuples produced at step j contain the following values:

$$\{a_j\} \qquad \text{for } j<i$$

$$\{a_i,\ldots,a_{\pi+1}\} \qquad \text{for } j\geq i$$

This is proved by induction on j:

Basis: Let j=0. In step zero of the processing there is only one tuple in P and all the attributes contain a_0.

Induction step: Assume that the above is true for all steps up to l. We will prove it for l+1. Every attribute that is a root (or rather that is one of the two attributes that form a root) always receives the full set of values from relations like Q_3 in figure 5.10. So, from the first step the whole set $\{a_1,\ldots,a_{\pi+1}\}$ is produced. Since a root is matched at stage 1, this proves our conjecture.

Every non-root attribute matched at stage i has a father matched at stage $i-1$. If $l < i-1$, then the values of the father of the attribute are in $\{a_l\}$, from the induction hypothesis. The form of α and the contents of the non-recursive relations imply that we can only get α_{l+1} in the attribute concerned in step i. Since $l+1 < i$ this agrees with our statement. If $l > i-1$, the values in the father of the attribute are in $\{a_{i-1},...,a_{\pi+1}\}$, from the induction hypothesis. This implies that its child will get values in $\{a_i,...,a_{\pi+1}\}$, which is the desired result since $l+1 \geq i$.

We will now show that in step j the unmatched attribute receives only the value a_j. Notice that in every step of the iterative process, all attributes that are connected with a match-join path have the same value in all qualifying tuples. This is because a match edge between two attributes means that all the tuples have the same value in these attributes, whereas a join edge forces equal values to the qualifying tuples because of the particular instance of the relations that we have chosen. Consider the father of the unmatched node. Since we are not dealing with the trivial case mentioned above (i.e. the recursive predicate has arity greater than 1), such a node does exist. It is either a non-root attribute or a root that is connected with a join edge to a non-root attribute. This is implied by the fact that every node in the graph has degree 2 (it is a cycle) and that, if the father was a root with two directed edges emanating from it, both its children would be matched, according to the algorithm above. In either case the effect is the same. Consider the values of the father of the unmatched attribute. In all qualifying tuples at any step these values are restricted to the values of some non-root matched attribute. In the case that there is only one value qualifying in the father attribute, say a_j (produced at step j), we can only get a_{j+1} in the unmatched attribute at step $j+1$. After some step, say k, all the matched attributes contain values from non-singleton sets. But in this case, as it has been mentioned above, there is a match-join path from the unmatched node to its father. So, if at step j the unmatched attribute contains the value a_j in all produced tuples, its father is forced to contain that value alone in all qualifying tuples. Therefore, at step $j+1$, the unmatched attribute will only get a_{j+1}.

Our assumption was that α was bounded, with bound $\bar{n}$. As we have proved above we will not get a tuple in P with $a_{\bar{n}+1}$ in the unmatched attribute until the $\bar{n}$+1-th iteration. Having obtained a contradiction we may conclude that α cannot be bounded.

We will now remove our restriction that for all static edges with one end in H and the other in H', the first attribute of the corresponding non-recursive relation is in H and the second is in H'. We will also remove our restriction that the three groups of relations (all in H, all in H' and going from to H to H') are disjoint. The proof is similar to the one above and we will only give a sketch of it. The instances of the relations are all equal. They all contain all tuples of the form $<a_j,a_j>$, $<a_j,a_{j+1}>$ and $<a_{j+1},a_j>$, for all $0 \leq j \leq \bar{N}+1$, where $\bar{N}$ will be given shortly. The difference with the previous simpler case is that now we may have different values in a tuple for two matched attributes. However the difference is bounded. If at one step the maximum difference between the values of any two matched attributes is d, then it will be $2d$ for their immediate children. Also, the difference between the two ends of a match-join path is the sum of the maximum differences for the match edges along the path. From these two observations we may get a crude estimate for an upper bound for $\bar{N}$. Assume that there are m matched edges in the graph. This means that we will need at most m stages to perform the matching. At any stage we can have at most m match edges in any match-join path. With an easy induction we can show that when we reach the step where the unmatched attribute is connected to its father with a match-join path, the difference between the values in these two attributes in any qualifying tuple can be at most $2^m m^{m-1}$. So, if we pick $\bar{N} = \bar{n}(2^m m^{m-1})+1$ we can show as before that we cannot get $a_{\bar{N}}$ before the $(\bar{n}+1)$-th iteration. Therefore even in the general case we may conclude that α is unbounded. □

Theorem 5.2: Let α be a simple recursive statement. If α is bounded then the α-graph contains no cycle of non-zero weight.

Proof: Suppose to the contrary that the α-graph contains a cycle of weight $n+1$, $n \geq 0$. Because of Lemma 5.9, we know that the α_n-graph contains a cycle of weight 1. Lemma 5.8

implies that α_n is bounded. Hence, there must be two expansions $(\alpha_n)_s$ and $(\alpha_n)_t, s < t$, such that the latter is more restrictive than the former. Lemma 5.6 implies that the maximum weight of any path in the dynamic $(\alpha_n)_s$-graph is 1. Furthermore, applying Lemma 5.10, we have that the $(\alpha_n)_s$-graph contains a cycle of weight 1. These last two points satisfy the conditions of Lemma 5.11; we may therefore conclude that $(\alpha_n)_s$ is not bounded. Applying Lemma 5.8 again we have that α is unbounded, which contradicts our initial hypothesis.

Having obtained a contradiction, we conclude that our assumption was wrong and therefore if α is bounded the α-graph cannot contain any cycle of non-zero weight. □

We will now attempt to illustrate Theorem 5.2 with an example. Consider the statement below:

$$\beta: \qquad P(u_1,w,u_2,u_3) \wedge Q(w,u_2) \wedge R(y,u_3) \wedge S(x,z) \rightarrow P(w,x,y,z)$$

The β-graph appears in figure 5.11.

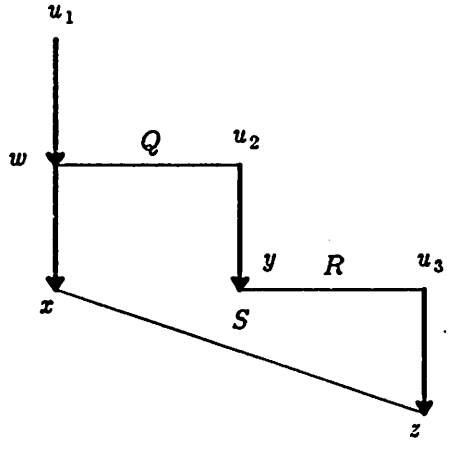

Fig. 5.11 : The β-graph

The β-graph contains a cycle of weight 1, namely $(w \rightarrow u_2 \rightarrow y \rightarrow u_3 \rightarrow z \rightarrow x \rightarrow w)$. Recall that the edge $(x \rightarrow w)$ with weight -1 does exist, even though it is not shown in the figure, and also that an undirected edge can be traversed in both directions. Hence, according to our theorem, β cannot be bounded. The expansions of β become quite complicated and difficult to read. Thus we will attempt to convince ourselves of the unboundedness of β by looking only at the graphs of these expansions. The β_1- and β_2-graphs appear in figures 5.12 and 5.13 respectively. Contrary to what

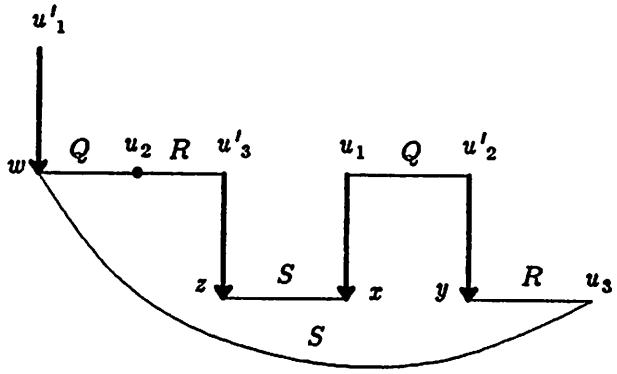

Fig. 5.12 : The β_1-graph

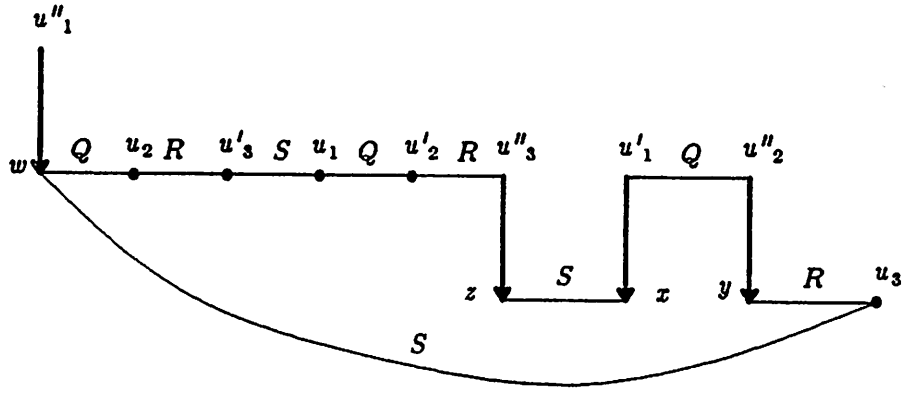

Fig. 5.13 : The β_2-graph

happened to the graphs of bounded statements, the graphs of β's expansions continue to have a single component, but the number of undirected (static) edges in the original cycle increases. This continues, no matter how many expansions we perform. Clearly, for two cycles to be isomorphic, they have to contain the same number of edges. Since no expansion can have a graph which is isomorphic to a subgraph of any previous expansion, there can be no upper bound on the number of them that are significant for the result. Notice that all expansions have a cycle of weight 1 in their graph, which should be expected because of Lemma 5.10.

The results in Sections 5.1 and 5.2 lead us into the following theorem.

Theorem 5.3: Let α be a simple recursive statement. Statement α is bounded if and only if the α-graph contains no cycle of non-zero weight. In that case a tight upper bound on the rank of α is equal to the maximum weight of any path in the α-graph.

Proof: The proof follows immediately from Theorems 5.1 and 5.2. □

The condition of Theorem 5.3 is sufficient for a statement to be bounded even when restrictions (3) and (4) of Section 2 are removed. However it is not necessary. For example consider the trivial example

$$P(y,x) \rightarrow P(x,y)$$

This statement is not simple. It violates restriction (4) by having a subsequence of the variables under the recursive predicate in the consequent being a permutation of the corresponding variables in the antecedent. The graph of the statement appears in figure 5.14.

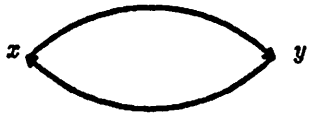

Fig. 5.14 : Graph violating restriction (4)

As expected the dynamic graph (restricted to the positive edges), which in this case is equal to the whole graph, is not a forest. Even though it is clear that the statement is bounded with bound 1, the graph contains a cycle of weight 2, thus violating Theorem 5.3. Future work should attempt to generalize the condition of the theorem to include statements like this as well.

6. ALGORITHMS

The condition of Theorem 5.3 can be easily checked in the α-graph of some simple recursive statement α. We can use a depth-first search algorithm on the graph in its original form with all positive and negative dynamic edges. Whenever we start searching a new component we assign the number zero to the first node we visit. Every time we traverse an edge in the graph, we

increase this count by the weight of the edge and we assign the current contents of the count to the node at the other end of the edge. If at any point we visit a node already visited and the two numbers assigned to it are not equal, then there is a non-zero weight cycle in the graph, and therefore α is not bounded. Otherwise α is bounded. The maximum weight of any path within a component is the sum of the maximum positive number plus the absolute value of the minimum negative number assigned to any of its nodes. The maximum such number over all the components is the bound of α. The algorithm just described appears below.

Input: Some α-graph $G=(V,E,W)$, where V is the set of nodes, E the set of edges and W the mapping of every edge to its weight; the graph is represented by adjacency lists $L[v]$, for $v \in V$.

Output: If α is bounded then give its bound, otherwise give "UNBOUNDED".

Algorithm: In the following algorithm $D[v]$ denotes the count assigned to v as it was described above, and $W[v,w]$ denotes the weight of the edge $(v \rightarrow w)$.

```
begin
    bound:=0;
    for all v in V do mark v "new";
    while there exists a vertex v in V marked "new" do
        begin
            maxpos:=0; maxneg:=0;
            SEARCH (v,0);
            bound:=MAX(bound,maxpos+maxneg)
        end
    output(bound);
end
procedure SEARCH (v,count):
    begin
        mark v "old";
        D[v] := count;
        maxpos := MAX(maxpos,count);
        maxneg := MAX(maxneg,-count);
        for every vertex w in L[v] do
            if w is marked "new" then SEARCH (w,count+W[v,w])
            elseif D[w] ≠ count+W[v,w] then output ("UNBOUNDED"); exit;
    end
```

Alg. 6.1

Lemma 6.1: Algorithm 6.1 returns "UNBOUNDED" if and only if its input graph has some cycle of non-zero weight, otherwise it returns the maximum weight of any path the graph. Furthermore it requires $O(\nu+\epsilon)$ steps, where ν is the number of nodes and ϵ is the number of

edges in the graph.

Proof: The correctness of the algorithm is obvious, from our discussion in the beginning of this section. As it concerns its running time, the algorithm is a depth-first search of a graph, with a constant number of operations in each step. Therefore, its running time is $O(\nu+\epsilon)$. □

7. APPLICATIONS

Besides its theoretical interest, our result may have considerable implications on how general recursive statements can be processed in a deductive database environment. We have no general results in that direction but we speculate that many recursive statements can be decomposed into "smaller" ones, some of which are bounded. Our unbounded statements will be smaller than the initial one and this will result in faster processing. Furthermore, the parts of the result that correspond to bounded statements will be obtained in a bounded number of steps independent of the rest of the statement. This should result in greater efficiency, since the processing of the original statement may involve many more steps than the bound of the bounded statements. Processing the original statement in its initial form would recompute the same things again and again. Of course there will be some overhead in the end to combine the results of the various substatements in a way that produces the same result as the original statement. In many cases though, the effort will be worth some net savings in computational cost.

As an example of such a decomposition consider the following statement

$$P(z,w) \wedge Q(z,w) \wedge R(x,y) \wedge S(z,x) \rightarrow P(x,y)$$

The graph of this statement is shown in figure 7.1.

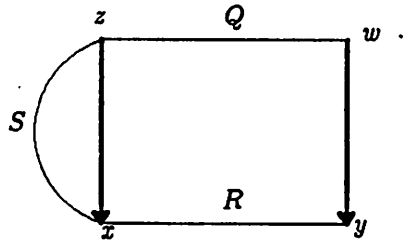

Fig. 7.1 : Graph of decomposable statement

The statement can be decomposed into the two statements

$P_1(z,w) \wedge Q(z,w) \wedge R(x,y) \rightarrow P_1(x,y)$

$P_2(z) \wedge S(z,x) \rightarrow P_2(x)$

The corresponding graphs of the two statements appear in figure 7.2.

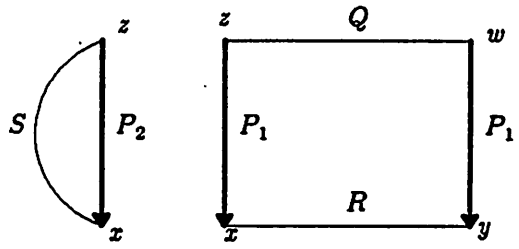

Fig. 7.2 : Graphs of statements after decomposition

As we can see the first statement is bounded, with bound 1, while the second is unbounded. Processing the two statements separately and then combining the two results may affect the processing time significantly.

As for a single statement, the information that it is bounded might prove to be useful as well. Its definition can be expressed non-recursively in a finite form. This makes applicable all the tools used in conventional relational databases to find fast access paths to process the statement. It also makes it much easier to compile such an access path compared to the effort needed for a general unbounded statement (e.g. see [Naqv84]). Finally, when we know that a statement is bounded, with say bound $\bar{n}$, we never need to process the statement for an $(\bar{n}+1)$-th time, only to discover that no new tuples are produced. For statements with small bounds, say 1 or 2, this may prove to be quite significant. We should also note that, even though we have examined the problem of bounded recursive statements in a deductive database context, our results apply to other similar environments, like those based on the PROLOG programming language.

All the above lead us to the conclusion that the existence of bounded recursive statements and our ability to characterize them is an important step towards efficient processing algorithms for recursion.

8. CONCLUSIONS

We have considered a restricted class of recursive statements in the context of a deductive database. We have demonstrated that some such statements are amenable to an equivalent finite nonrecursive expression, i.e. using the first n expansions of the statement, where n is its bound. By modeling such a statement with a weighted graph, we have shown that the property that the statement can be expressed in a finite way is equivalent to the property that the graph has no cycles of non-zero weight. Finally, we have indicated some possible implications of our result in the construction of efficient algorithms to process recursive statements.

Many issues are left for future work. We are currently attempting to obtain necessary and sufficient conditions for more general classes of recursive statements, by removing some of the restrictions (1) to (4) of Section 2. We believe that this should not be difficult for restrictions (3) and (4) and partly for restriction (2). As an even more important task for the future we consider the study of unbounded recursive statements. We are currently investigating the possibility of decomposing such a statement into smaller ones some of which are bounded, in the way it was demonstrated in Section 7. Finally, much work needs to be done for unbounded, non-decomposable recursive statements.

Acknowledgements: I am deeply indebted to Timos Sellis for the innumerable discussions I had with him on the subject. I would also like to give thanks to Prof. E. Wong for all his valuable help and guidance and to Eric Hanson for his useful comments on earlier drafts of this paper.

9. REFERENCES

[Bond76]
Bondy, J. A. and U. S. R. Murty, *"Graph Theory with Applications"*, North Holland, 1976.

[Chan81]
Chang, C. L., "On Evaluation of Queries Containing Derived Relations in a Relational Data Base", in *"Advances in Data Base Theory"*, **Vol. 1**, edited by H. Galaire, J. Minker and J. M. Nicolas, Plenum Press, New York, N.Y., 1981, pages 235-260.

[Codd70]
Codd, E. F., "A Relational Model of Data for Large Shared Data Banks", *CACM* **13**, 6 (1970), pages 377-387.

[Date82]
Date, C. J., *"An Introduction to Database Systems", 3rd edit.*, Addison-Wesley, Reading, MA, 1982.

[Ende72]
Enderton, H. B., *"A Mathematical Introduction to Logic"*, Academic Press, New York, N.Y., 1972.

[Gall78]
Gallaire, H. and J. Minker, *"Logic and Data Bases"*, Plenum Press, New York, N.Y., 1978.

[Gall81]
Gallaire, H., J. Minker, and J. M. Nicolas, *"Advances in Data Base Theory"*, Plenum Press, New York, N.Y., 1981.

[Gall84]
Gallaire, H., J. Minker, and J. M. Nicolas, "Logic and Databases: A Deductive Approach", *ACM Computing Surveys* **16**, 2 (1984).

[Naqv84]
Naqvi, S. and L. Henschen, "On Compiling Queries in Recursive First-Order Databases", *JACM* **31**, 1 (1984).

[Reit78]
Reiter, R., "Deductive Question-Answering on Relational Data Bases", in *"Logic and Data Bases"*, edited by H. Galaire and J. Minker, Plenum Press, New York, N.Y., 1978, pages 149-177.

[Robi65]
Robinson, J. A., "A Machine Oriented Logic Based on the Resolution Principle", *JACM* **12**, 1 (1965), pages 23-41.