PROCEEDINGS

# Southern African Computer Symposium 1991

# 6th de Rekenaarsimposium van Suider Afrika 1991

DE OVERBERGER HOTEL, CALEDON

**2 - 3 JULY 1991**

SPONSORED BY

**ISM**

**FRD**

**GENMIN**

EDITED BY

**M H Linck**

DEPARTMENT OF COMPUTER SCIENCE • UNIVERSITY OF CAPE TOWN

# PROCEEDINGS / KONGRESOPSOMMINGS

# 6th
# SOUTHERN AFRICAN COMPUTER SYMPOSIUM

# 6de
# SUIDELIKE-AFRIKAANSE REKENAARSIMPOSIUM

De Overberger Hotel, Caledon

2 - 3 JULY 1991

SPONSORED by

ISM
FRD
GENMIN

EDITED by

M H LINCK

Department of Computer Science

University of Cape Town

## TABLE OF CONTENTS

## FOREWORD

The 6th Computer Symposium, organised under the auspices of SAICS, carries on the tradition of providing an opportunity for the South African scientific computing community to present research material to their peers.

It was heartening that 31 papers were offered for consideration. As before all these papers were refereed. Thereafter a selection committee chose 21 for presentation at the Symposium.

Several new dimensions are present in the 1991 symposium:

* The Symposium has been arranged for the day immediately after the SACLA conference.

* It is being run over only 1 day in contrast to the 2-3 days of previous symposia.

* I believe that it is first time that a Symposium has been held outside of the Transvaal.

* Over 85 people will be attending. Nearly all will have attended both events.

* A Sponsorship package for both SACLA and the Research Symposium was obtained. (This led to reduced hotel costs compared to previous symposia)

A major expense is the production of the Proceedings of the Symposium. To ensure financial soundness authors have had to pay the page charge of R20 per page.

A thought for the future would be consideration of a poster session at the Symposium. This could provide an alternative approach to presenting ideas or work.

I would sincerely hope that the twinning of SACLA and the Research Symposium is considered successful enough for this combination survive. As to whether a Research Symposium should be run each year after SACLA, or only every second year, is a matter of need and taste.

A challenge for the future is to encourage an even greater number of MSc & PhD students to attend the Symposium. Unlike this year, I would recommend that they be accommodated at the same cost as everyone else. Only if it is financially necessary should the sponsored number of students be limited.

I would like to thank the other members of the organising committee and my colleagues at UCT for all the help that they have given me. A special word of thanks goes to Prof. Pieter Kritzinger who has provided me with invaluable help and ideas throughout the organisation of this 6th Research Symposium.

M H Linck
Symposium Chairman

## SYMPOSIUM CHAIRMAN

M H Linck, University of Cape Town

## ORGANISING COMMITTEE

D Kourie, Pretoria University.

P S Kritzinger, University of Cape Town.

M H Linck, University of Capè Town.

## SPONSORS

ISM

GENMIN

FRD

**LIST OF REFEREES FOR 6th RESEARCH SYMPOSIUM**

| NAME | INSTITUTION |
|---|---|
| Barnard, E | Pretoria |
| Becker, Ronnie | UCT |
| Berman S | UCT |
| Bishop, Judy | Wits |
| Berman, Sonia | UCT |
| Brink, Chris | UCT |
| Bodde, Ryn | Networks Systems |
| Bornman, Chris | UNISA |
| Bruwer, Piet | UOFS |
| Cherenack, Paul | UCT |
| Cook Donald | UCT |
| de Jaeger, Gerhard | UCT |
| de Villiers, Pieter | Stellenbosch |
| Ehlers, Elize | RAU |
| Eloff, Jan | RAU |
| Finnie, Gavin | Natal |
| Gaynor, N | AECI |
| Hutchinson, Andrew | UCT |
| Jourdan, D | Pretoria |
| Kourie Derrick | Pretoria |
| Kritzinger, Pieter | UCT |
| Krzesinski, Tony | Stellenbosch |
| Laing, Doug | ISM |
| Labuschagne, Willem | UNISA |
| Levy, Dave | Natal |

| | |
|---|---|
| MacGregor, Ken | UCT |
| Machanick, Philip | Wits |
| Mattison Keith | UCT |
| Messerschmidt, Hans | UOFS |
| Mutch, Laurie | Shell |
| Neishlos, N | Wits |
| Oosthuizen, Deon | Pretoria |
| Peters Joseph | Simon Fraser |
| Ram, V | Natal, Pmb. |
| Postma, Stef | Natal, Pmb |
| Rennhackkamp, Martin | Stellenbosch |
| Shochot, John | Wits |
| Silverberg, Roger | Council for Mineral Technology |
| Smit, Riël | UCT |
| Smith, Dereck | UCT |
| Terry, Pat | Rhodes |
| van den Heever, Roelf | UP |
| van Zijl, Lynette | Stellenbosch |
| Venter, Herman | Fort Hare |
| Victor, Herna | Stellenbosch |
| von Solms, Basie | RAU |
| Wagenaar, M | UPE |
| Wentworth, Peter | Rhodes |
| Wheeler, Graham | UCT |
| Wood, Peter | UCT |

## 6TH RESEARCH SYMPOSIUM - 1991

## FINAL PROGRAM

**TUESDAY 2nd July 1991**

**10h00 - 13h00** **Registration**

**13h00 - 13h50** **PUB LUNCH**

**14h00 - 15h30 SESSION 1A**

Venue: Hassner

Chairman: Prof Basie von Solms

14h00 - 14h30
*"A value can belong to many types. "*
B H Venter, University of Fort Hare

14h30 - 15h00
*"A Transputer Based Embedded Controller Development System"*
M R Webster, R G Harley, D C Levy & D R Woodward, University of Natal

15h00 - 15h30
*"Improving a Control and Sequencing Language"*
G Smit and C Fair, University of Cape Town

**SESSION 1B**

Venue: Hassner C

Chairman:Prof Roelf v d Heever

14h00 - 14h30
*"Design of an Object Orientated Framework for Optimistic Parallel Simulation on Shared-Memory Computers"* P Machanick, University of Witwatersrand

14h30 - 15h00
*"Using Statecharts to Design and Specify the GMA Direct-Manipulation User Interface"* L van Zijl & D Mitton, University of Stellenbosch

15h00 - 15h30
*"Product Form Solutions for Multiserver Centres with Heirarchical Classes of Customers"* A Krzesinski, University of Stellenbosch and R Schassberger, Technische Universität Braunschweig

**15h30 - 16h00** **TEA**

**16h00 - 17h30 SESSION 2A**

Venue: Hassner

Chairman: Prof Derrick Kourie

16h00 - 16h30
*"A Reusable Kernel for the Development of Control Software"* W Fouché and P de Villiers, University of Stellenbosch

16h30 - 17h00
*"An Implementation of Linda Tuple Space under the Helios Operating System"* P G Clayton, E P Wentworth, G C Wells and F de-Heer-Menlah, Rhodes University

17h00 - 17h30
*"The Design and Analysis of Distributed Virtual Memory Consistency Protocols in an Object Orientated Operating System"* K MacGregor, University of Cape Town & R Campbell, University of Illinois at Urbana-Champaign

**19h30** **PRE-DINNER DRINKS**

**20h00** **GALA CAPE DINNER**
(Men: Jackets & ties)

**WEDNESDAY 3rd July 1991**

**7h00 - 8h15 BREAKFAST**

**8h15 - 9h45 SESSION 3A**

Venue: Hassner

Chairman: Assoc Prof P Wood

8h15 - 8h45
*"Concurrency Control Mechanisms for Multidatabase Systems"* A Deacon, University of Stellenbosch

8h45 - 9h15
*"Extending Local Recovery Techniques for Distributed Databases"* H L Victor & M H Rennhackkamp, University of Stellenbosch

9h15 - 9h45
*"Analysing Routing Strategies in Sporadic Networks"* S Melville, University of Natal

**SESSION 3B**

Venue: Hassner C

Chairman: Prof G Finnie

8h15 - 8h45
*The Design of a Speech Synthesis System for Afrikaans"* M J Wagener, University of Port Elizabeth

8h45 - 9h15
*"Expert Systems for Management Control: A Multiexpert Architecture"* V Ram, University of Natal

9h15 - 9h45
*"Integrating Simularity-Based and Explanation-Based Learning"* G D Oosthuizen and C Avenant, University of Pretoria

**9h45 - 10h15 TEA**

**10h15 - 11h00 SESSION 4**

Venue: Hassner

Chairman: Prof P S Kritzinger
Invited paper: E Coffman

**11h00 - 11h10 BREAK**

**11h10 - 12h40 SESSION 5A**

Venue: Hassner

Chairman: Prof C Bornman

11h10 - 11h40
*"Efficient Evaluation of Regular Path Programs"*
P Wood, University of Cape Town

11h40 - 12h10
*"Object Orientation in Relational Databases"*
M Rennhackkamp, University of Stellenbosch

12h10 - 12h40
*"Building a secure database using self-protecting objects"* M Olivier and S H von Solms, Rand Afrikaans University

**SESSION 5B**

Venue: Hassner C

Chairman: Prof A Krzesinski

11h10 - 11h40
*"Modelling the Algebra of Weakest Preconditions"*
C Brink & I Rewitsky, University of Cape Town

11h40 - 12h10
*"A Model Checker for Transition Systems"*
P de Villiers, University of Stellenbosch

12h10 - 12h40
*"A New Algorithm for Finding an Upper Bound of the Genus of a Graph"*
D I Carson and O R Oellermann, University of Natal

**12h45-12h55 GENERAL MEETING of RESEARCH SYMPOSIUM ATTENDEES**

Venue: Hassner

Chairman: Dr M H Linck

**13h00 - 14h00 LUNCH**

**FINIS 6th COMPUTER SYMPOSIUM**

**PAPERS**

**of the**

**6TH RESEARCH SYMPOSIUM**

# Efficient Evaluation of Regular Path Programs

Peter T. Wood
Department of Computer Science
University of Cape Town
Rondebosch 7700, South Africa

**Abstract**

The next generation of query languages for database systems should have the ability to express recursive queries, the efficient evaluation of which will be crucial to the success of these systems. One such query language which has been the subject of much research is Datalog. We define a class of Datalog programs, namely, the regular path programs, which can always be evaluated efficiently, in particular, when constants are present in a query. Efficient evaluation is ensured by reducing the number of arguments appearing in each predicate defined in the program. The class of regular path programs is incomparable to previous classes to which the technique of argument reduction has been applied.

## 1 Introduction

There is little doubt that the next generation of database systems should support query languages that are more powerful than those usually associated with relational databases. The language Datalog is one of those that has received much attention in this regard [Ullm88]. Datalog has a Prolog-like syntax, but, unlike Prolog, is evaluated bottom-up rather than top-down. The reasons for this are to ensure termination of queries and to exploit the efficiency of relational algebra for operations on sets of tuples [Ullm89].

**Example 1.1** Consider the following example taken from [Naug87]. Assume that the database comprises two relations: $likes(X,Y)$, which states that person $X$ likes product $Y$, and $knows(X,Y)$, which states that person $X$ knows person $Y$. Suppose that people buy the products that they like, or those bought by someone they know. Then the following program $P$ defines the relation $buys(X,Y)$ of people $X$ and the products $Y$ they buy.

$$\begin{aligned} buys(X,Y) &:- likes(X,Y). \\ buys(X,Y) &:- knows(X,Z), buys(Z,Y). \end{aligned}$$

Note that this program is recursive, in that *buys* is computed in terms of itself. This is a capability not found in traditional query languages.

If we are interested in the entire *buys* relation, then we phrase the query $?-buys(X,Y)$ against the program. In order to evaluate this query bottom-up, the system would undertake an iterative process known as semi-naive evaluation, in which successive approximations to the complete *buys* relation are computed. These approximations represent larger and larger subsets of the *buys* relation, with the final iteration producing the complete answer. The first approximation is the *likes* relation itself. Successive approximations are found by joining previous approximations of *buys* with *knows* (based on the common attribute $Z$) until no new tuples are found. □

One disadvantage of bottom-up methods is their inability in certain cases to "focus" the computation. For instance, if in the above example we wanted to know what John buys, rather than what everyone buys, then we would specify the query $? - buys(john, Y)$ instead of $? - buys(X, Y)$. However, a simple-minded bottom-up evaluation would still compute the entire *buys* relation before selecting out those tuples whose first component is John. What is needed is a method analogous to pushing select operations into relational algebra expressions [Ullm88], but one which works for recursive programs.

Top-down methods are usually better in this respect. In our example, a top-down method would first find which products John likes, then whom John knows along with the products they like, and so on. One way to achieve such behaviour for bottom-up evaluation is to transform a given program so that its bottom-up evaluation mimics a top-down evaluation. This is the goal of a method known as Magic Sets [BMSU86,BR87].

**Example 1.2** Given the above program $P$ along with query $? - buys(John, Y)$, Magic Sets produces the following:

$$
\begin{array}{lll}
magic(john). & & \\
magic(Z) & :- & magic(X), knows(X, Z). \\
buys(X, Y) & :- & magic(X), likes(X, Y). \\
buys(X, Y) & :- & magic(X), knows(X, Z), buys(Z, Y). \\
? - buys(john, Y). & &
\end{array}
$$

The purpose of the first two rules is to compute all possible bindings of the first argument of *buys* that would be used in a top-down evaluation. This "magic" predicate is then substituted into each of the original rules in order to restrict the bindings in a bottom-up computation. □

However, the Magic Sets transformation in the above example is still not the most efficient possible. It is not hard to see that the recursive rule for *buys* is now redundant, since the transitive closure of the *knows* relation starting with John is computed by the second magic rule, from which all the products bought by John are found by the nonrecursive rule for *buys*. We can therefore transform the program to the following:

$$
\begin{array}{lll}
magic(john). & & \\
magic(Z) & :- & magic(X), knows(X, Z). \\
buys(john, Y) & :- & magic(X), likes(X, Y). \\
? - buys(john, Y). & &
\end{array}
$$

Notice that the number of arguments in the recursive predicate in the program (initially *buys*, now *magic*) has been reduced from two to one. This technique, known as ***argument reduction*** or ***factoring***, has been applied previously to various classes of Datalog programs [NRSU89a,NRSU89b]. In this paper, we apply the technique to a different class of programs, namely, the ***regular path programs***.

In the next section, we begin by defining the class of regular path programs. Section 3 is devoted to the application of argument reduction to regular programs when constants are specified in queries. Our claim that this leads to efficient evaluation of such programs is explored in Section 4, where it is shown that the transformed programs are often considerably less time consuming to evaluate than the original programs. Conclusions and topics for future research are discussed in Section 5.

## 2 Regular Path Programs

In the previous section, we saw one example of a regular path program; we now define the class of such programs exactly. The term regular path program is derived from the fact that there is a correspondence between these programs and the problem of finding paths which satisfy a given regular expression in a labelled, directed graph. This connection is explored in more detail in [Wood90], although it should be noted that the regular path programs defined below differ from the programs defined in that paper.

Essentially, we want to generalise programs such as that given in the previous section in two ways. Firstly, we would like to have more complicated rule structures as demonstrated by the following example.

**Example 2.1** Assume we have a semantic network in which the nodes represent instances, classes or properties, while edges from instances to classes are labelled with *isa* (classification), edges between classes are labelled with *ako* (generalisation), and edges from either instances or classes to properties are labelled with *can*, *has* or *is*. The following program finds the relationship *i_inherit* between instances and the properties they inherit.

$$\begin{array}{lcl} i\_inherit(X,Y) & :- & isa(X,Z), c\_inherit(Z,Y). \\ c\_inherit(X,Y) & :- & ako(X,Z), c\_inherit(Z,Y). \\ c\_inherit(X,Y) & :- & can(X,Y). \\ c\_inherit(X,Y) & :- & has(X,Y). \\ c\_inherit(X,Y) & :- & is(X,Y). \end{array}$$

Note that the above program displays a natural correspondence to the notion of a regular grammar. □

A second way in which we would like to generalise programs is by adding arguments to predicates in order to pass more information among the rules. The following example demonstrates this ability.

**Example 2.2** Suppose we have an application involving a network with values labelling the edges connecting nodes. Assume that this information is stored as an edge relation $e(X,Y,Z)$, which states that there is an edge from $X$ to $Y$ labelled $Z$. If we want to find all nodes that are connected by routes on which at most two alternating values are used, the following program will suffice.

$$\begin{array}{lcl} s(X,Y,U,V) & :- & e(X,Z,U), t(Z,Y,U,V). \\ t(X,Y,U,V) & :- & e(X,Z,V), s(Z,Y,U,V). \\ s(X,Y,U,V) & :- & e(X,Z,U), e(Z,Y,V). \end{array}$$

Note that the program still has a regular structure and contains mutually recursive predicates $s$ and $t$. The variables $U$ and $V$ are used to hold the pairs of edge labels, while the alternation of values is captured by the mutual recursion. □

Before we proceed, we need to introduce some standard terminology. Given a rule such as

$$buys(X,Y) \ :- \ knows(X,Z), buys(Z,Y).$$

the predicate to the left of : − is called the ***head*** of the rule, while those to the right constitute the ***body*** of the rule. Variables in a program are denoted by strings with an initial upper case letter (e.g. $X$), while constants are numeric values or strings having an initial lower case letter (e.g. *john*). As is common with Datalog, we assume that all rules are ***safe***, that is, every variable appearing in the head of a rule also appears somewhere in the body. We also assume that there are no constants in the rules, except possibly for the query clause. Predicates that correspond to relations stored in the database (such as ***knows*** above) are called *EDB predicates* (extensional database predicates); those that are defined by rules (such as ***buys*** above) are called *IDB predicates* (intensional databases predicates). We make the standard assumption that no EDB predicate appears in the head of any rule.

Regular path programs are defined as follows:

1. Rules must correspond to productions of a regular grammar; in other words, the body of each rule must comprise either (a) a single EDB predicate and a single IDB predicate (an *internal* rule), or (b) one or more EDB predicates (an *exit* rule).

2. All IDB predicate occurrences must have the same number of arguments.

3. Given an internal rule of the form

$$s(X,Y,U,V) \ :- \ e(X,Z,U),t(Z,Y,U,V).$$

   where $s$ and $t$ are IDB predicates, $e$ and $t$ have exactly one variable in common that does not appear in the head ($Z$ above). The position of this variable in $t$ is called the ***linking*** position.

4. The position in the head $s$ corresponding to the linking position in $t$ is occupied by a variable ($X$ above) which appears only in $e$. This is the ***source*** variable.

5. The remaining positions in $s$ must be occupied by variables appearing in the same positions in $t$—the ***persistent*** variables ($Y$, $U$ and $V$ above).

6. The source (or linking) position for each IDB predicate in the program must be the same.

7. For the whole program there must be exactly one persistent position such that for every IDB predicate appearing in the body of a rule the variable occupying that position appears nowhere else in the body. This position is called the ***sink*** position, and in each rule the corresponding variable is called the ***sink*** variable.

**Example 2.3** Referring to the program in Example 2.1, it is easy to see that it conforms to the definition of a regular path program. In each of the internal rules, $Z$ is the linking variable, $X$ is the source variable, and $Y$ is the sink variable. There are no other persistent variables.

Now turning to the first internal rule of Example 2.2, once again $Z$ is the linking variable and $X$ the source variable; hence, $Y$, $U$ and $V$ are persistent variables. Because $U$ also appears in $e(X,Z,U)$, the third argument position in IDB predicates cannot be the sink position. From the second internal rule, we establish that the fourth IDB argument position (occupied by $V$) also cannot be the sink position. Hence, $Y$ is the sink variable in each of the internal rules. □

The reader should not be mistaken into thinking that the EDB predicate must always precede the IDB predicate in an internal rule. The rule

$$s(X,Y,V) \ :- \ t(X,Z,V), e(Z,Y,V).$$

where $e$ is the EDB predicate, does not violate the definition: $Z$ is the linking variable, $Y$ the source variable, and $X$ the sink variable. Some examples violating the definition are given below.

**Example 2.4** The pair of internal rules

$$\begin{aligned} s(X,Y,V) \ &:- \ e(X,Z,U), t(Z,Y,V). \\ t(X,Y,V) \ &:- \ e(X,Z,V), s(X,Y,Z). \end{aligned}$$

violates item (6) in the definition, since the source position is 1 in the first rule and 3 in the second. On the other hand, the pair of rules

$$\begin{aligned} s(X,Y,V) \ &:- \ e(X,Z,U), t(Z,Y,V). \\ t(X,Y,V) \ &:- \ e(X,Z,Y), s(Z,Y,V). \end{aligned}$$

violates item (7), since the first rule implies that the sink position must be 2, while the second rule implies it must be 3. □

The definition of regular path programs can be extended in a number of ways. Firstly, we can allow a ***chain*** of EDB predicates of the form

$$e_0(X,W_1), e_1(W_1,W_2), \ldots, e_n(W_n,Z)$$

rather than the single EDB predicate $e(X,Z)$ in an internal rule. It is also possible to allow tuples of variables rather than single variables for source and sink variables. We do not use these extensions in the remainder of the paper as they tend to make the notation more difficult to follow.

In the next section, we consider regular path programs with queries in which constants appear, for example, $? - buys(john, Y)$ or $? - s(X, y_0, U, v_0)$. If the constant appears in the source position, we call the program ***right-linear***; if constants appear in one or more of the persistent positions, the program is called ***left-linear***. These terms are consistent with those used in [NRSU89a].

# 3 Argument Reduction

We now turn our attention to ways in which a left- or right-linear regular path program can be rewritten so that its evaluation can be performed more efficiently. In this section, we show that the techniques presented in [NRSU89a] can be extended to apply to these programs.

## 3.1 Left-Linear Programs

If any persistent variables are bound to constants in a query to a regular path program $P$, we simply substitute the constants for the corresponding variables in all EDB predicates in $P$ and delete the variables wherever else they appear in $P$.

**Example 3.1** Consider the program of Example 2.2 in which argument positions 2, 3 and 4 are persistent. If we have the query $s(X, y_0, U, v_0)$, then the program is rewritten as follows.

$$\begin{array}{lll} s(X,U) & :- & e(X,Z,U), t(Z,U). \\ t(X,U) & :- & e(X,Z,v_0), s(Z,U). \\ s(X,U) & :- & e(X,Z,U), e(Z,y_0,v_0). \end{array}$$

All occurrences of $Y$ and $V$ in IDB predicates have been deleted, while all occurrences of $Y$ and $V$ in EDB predicates have been replaced by $y_0$ and $v_0$, respectively. As a result, the number of arguments in each IDB predicate occurrence has been reduced from four to two. □

Given a regular path program $P$, we can always rewrite it so that (1) each rule uses the same set of variables in its head, and (2) the source variable appears in the first argument position of each IDB predicate in the head, the sink variable appears in the second position, and the remaining persistent variables appear in the same order in each rule. We will assume this standard form from now on.

Let the query to the program be given by $? - q(X, \overline{V})$, the tuple of persistent variables being $\overline{V}$. We assume that $\overline{W} = W_1, \ldots, W_m$ is the subtuple of variables in $\overline{V}$ that are bound to constants $w_i$, $1 \leq i \leq m$, in the query, and that $\theta$ is the substitution that replaces each $W_i$ by $w_i$, $1 \leq i \leq m$. Let $\overline{V} - \overline{W}$ denote the removal of all variables in $\overline{W}$ from $\overline{V}$. The general method is as follows.

1. Given an exit rule of the form

$$t(X, \overline{V}) \quad :- \quad \mathcal{E}_1, \ldots, \mathcal{E}_k.$$

where $\mathcal{E}_1, \ldots, \mathcal{E}_k$ are $e$ (EDB) literals, transform it to

$$t(X, \overline{V} - \overline{W}) \quad :- \quad \theta(\mathcal{E}_1), \ldots, \theta(\mathcal{E}_k).$$

2. Given an internal rule of the form

$$t(X, \overline{V}) \quad :- \quad e(X, Z, U), s(Z, \overline{V}).$$

where $U$ appears in $\overline{V}$, transform it to

$$t(X, \overline{V} - \overline{W}) \quad :- \quad e(X, Z, \theta(U)), s(Z, \overline{V} - \overline{W}).$$

The query $q(X, \overline{V} - \overline{W})$ is now applied to the transformed program.

## 3.2 Right-Linear Programs

If the source variable $X$ in a query $q(X, Y, \overline{V})$ is bound to a constant, we apply a transformation based on that of Magic Sets [BMSU86,BR87], similar to the technique in [NRSU89a]. The first step in such a transformation is the top-down propagation of the binding patterns through a program $P$, leading to an *adorned* program $P^{ad}$ [Ullm88], in which each IDB predicate $p$ has an adornment $\alpha$ indicating which arguments of $p$ are bound and which are free. For example, $p^{bf}$ means that the first argument of $p$ is bound while the second is free.

**Example 3.2** Consider again the program $P$ of Example 2.2. If we assume that the query is $? - s(x_0, Y, U, V)$, then the adorned program $P^{ad}$ is as follows.

$$\begin{array}{lcl} s^{bfff}(X,Y,U,V) & :- & e(X,Z,U), t^{bfbf}(Z,Y,U,V). \\ t^{bfbf}(X,Y,U,V) & :- & e(X,Z,V), s^{bfbb}(Z,Y,U,V). \\ s^{bfbb}(X,Y,U,V) & :- & e(X,Z,U), t^{bfbb}(Z,Y,U,V). \\ t^{bfbb}(X,Y,U,V) & :- & e(X,Z,V), s^{bfbb}(Z,Y,U,V). \\ s^{bfff}(X,Y,U,V) & :- & e(X,Z,U), e(Z,Y,V). \\ s^{bfbb}(X,Y,U,V) & :- & e(X,Z,U), e(Z,Y,V). \end{array}$$

In the query, only $X$ is bound so we start with the adornment $bfff$ for $s$. Given that $X$ is bound in the first rule, by the time $t$ is evaluated in a top-down evaluation, $Z$ and $U$ will also be bound; hence the adornment $bfbf$ for $t$. This process continues until all adornments which are generated in a top-down manner for all IDB predicates have been considered. □

The second step in the transformation is to use the standard technique to derive the set of magic rules for $P^{ad}$ [BMSU86]. The following example demonstrates the method.

**Example 3.3** We use the adorned program $P^{ad}$ from the previous example. The magic rules are used to compute bottom-up those bindings for variables that would have been used in a top-down computation. These bindings are represented by so-called magic predicates, one for each adorned version of an IDB predicate in $P^{ad}$. The magic predicates are formed by prefixing $m\_$ to the IDB predicates. We first generate a rule for the constant in the query:

$$m\_s^{bfff}(x_0)$$

For each internal rule in $P^{ad}$ of the form

$$t^{\alpha_1}(X,Y,\overline{V}) \; :- \; e(X,Z,U), s^{\alpha_2}(Z,Y,\overline{V}).$$

we generate its magic rule by (i) prefixing both $s$ and $t$ with $m\_$, (ii) deleting all free variables in $s$ and $t$, and (iii) exchanging $m\_s$ and $m\_t$. The reason for step (ii) is that we are interested only in bound variables, while that for step (iii) is that we want to simulate top-down evaluation by bottom-up evaluation. This gives rise to the following set of magic rules:

$$\begin{array}{lcl} m\_t^{bfbf}(Z,U) & :- & m\_s^{bfff}(X), e(X,Z,U). \\ m\_s^{bfbb}(Z,U,V) & :- & m\_t^{bfbf}(X,U), e(X,Z,V). \\ m\_t^{bfbb}(Z,U,V) & :- & m\_s^{bfbb}(X,U,V), e(X,Z,U). \\ m\_s^{bfbb}(Z,U,V) & :- & m\_t^{bfbb}(X,U,V), e(X,Z,V). \end{array}$$

□

In the final step of the transformation, magic predicates are introduced into the exit rules of $P^{ad}$. Unlike in the usual Magic Sets algorithm, only the magic rules and these modified exit rules are used to answer the original query.

**Example 3.4** Continuing with our running example, we must substitute each of $m\_s^{bfff}$ and $m\_s^{bfbb}$ into the exit rule of the program, giving:

$$s(Y,U,V) \ :- \ m\_s^{bfff}(X), e(X,Z,U), e(Z,Y,V).$$
$$s(Y,U,V) \ :- \ m\_s^{bfbb}(X,U,V), e(X,Z,U), e(Z,Y,V).$$

Note once again that the number of arguments in all IDB predicates has been reduced by the transformation. □

We now summarise the general method. Given a program $P$ and query $q(x_0, Y, \overline{V})$, $P$ is rewritten as follows.

1. Generate the adorned program $P^{ad}$ from $P$ and the query.
2. Generate the magic rules from $P^{ad}$ according to the algorithm in [BMSU86].
3. For each exit rule in $P^{ad}$ of the form

$$p^{\alpha}(X,Y,\overline{V}) \ :- \ \mathcal{E}_1, \ldots, \mathcal{E}_k.$$

where $\mathcal{E}_1, \ldots, \mathcal{E}_k$ are $e$ (EDB) literals, generate the rule

$$q(Y,\overline{V}) \ :- \ m\_p^{\alpha}(X,\overline{U}), \mathcal{E}_1, \ldots, \mathcal{E}_k.$$

where $\overline{U}$ is the tuple of persistent variables bound according to $\alpha$.

The query $q(Y,\overline{V})$ is now applied to the generated program.

In common with [NRSU89a], the above method has the advantage over Magic Sets that magic predicates are substituted into exit rules alone, the remaining rules of the original program being discarded.

# 4 Efficient Evaluation

The technique of argument reduction has been shown to speed up the evaluation of a range of Datalog programs. The programs to which the technique is applied in [NRSU89b] generalise the one-sided programs of [Naug87], the separable programs of [Naug88], and the right-, left- and combined-linear programs of [NRSU89a]. These programs are the so-called *RLC-stable* programs: those containing only right-linear, left-linear and combined-linear rules in terms of a single IDB predicate and one exit rule[1]. For these programs it has been proved that the evaluation of a program $P$ transformed by argument reduction is never less efficient than the evaluation of $P$ transformed by the Magic Sets algorithm, which is the best general purpose technique for speeding up evaluation of Datalog programs. In fact, as shown in [NRSU89b], programs transformed by argument reduction can lead to an order of magnitude improvement over Magic Sets in terms of evaluation efficiency.

We have already seen that regular path programs can contain multiple exit rules (Example 2.1), as well as more than one IDB predicate and mutually recursive rules (Example 2.2). Thus, there are regular path programs that are not RLC-stable. On the other hand, regular path programs do not contain combined-linear rules, so there are RLC-stable programs that are not regular path programs.

We show below by means of an example that, given a regular path program that is not RLC-stable, the transformations described in this paper can also lead to an order of magnitude improvement over Magic Sets in terms of evaluation efficiency.

[1]There are other restrictions as well.

**Example 4.1** Once again, we consider the program $P$ of Example 2.2, excluding the persistent variables $U$ and $V$ which are not needed for the present purpose:

$$\begin{aligned} s(X,Y) &:- e(X,Z), t(Z,Y). \\ t(X,Y) &:- e(X,Z), s(Z,Y). \\ s(X,Y) &:- e(X,Z), e(Z,Y). \end{aligned}$$

Assume that the query is $? - s(x_0, Y)$ and that the relation $e$ contains the $2n+1$ tuples $\{(x_i, x_{i+1}) \mid 0 \leq i \leq 2n-1\}$. The program $P^{ad}$ is as follows:

$$\begin{aligned} s^{bf}(X,Y) &:- e(X,Z), t^{bf}(Z,Y). \\ t^{bf}(X,Y) &:- e(X,Z), s^{bf}(Z,Y). \\ s^{bf}(X,Y) &:- e(X,Z), e(Z,Y). \end{aligned}$$

From $P^{ad}$ we get the following magic rules:

$$\begin{aligned} & m\_s^{bf}(x_0). \\ m\_t^{bf}(Z) &:- m\_s^{bf}(X), e(X,Z). \\ m\_s^{bf}(Z) &:- m\_t^{bf}(X), e(X,Z). \end{aligned}$$

The relation $m\_t^{bf}$ contains the $n$ values $\{x_1, x_3, \ldots, x_{2n-1}\}$, while $m\_s^{bf}$ contains the $n+1$ values $\{x_0, x_2, \ldots, x_{2n}\}$. Substituting the magic predicate into the exit rule according to the method of Section 3.2 yields the following:

$$s^{bf}(Y) \;:- \; m\_s^{bf}(X), e(X,Z), e(Z,Y).$$

The relation $s^{bf}$ contains the values $\{x_2, x_4, \ldots, x_{2n}\}$, which is the answer to the query. On the other hand, the Magic Sets algorithm produces

$$s^{bf}(X,Y) \;:- \; m\_s^{bf}(X), e(X,Z), e(Z,Y).$$

where $s^{bf}$ containes the tuples $\{(x_{2i}, x_{2i+2}) \mid 0 \leq i \leq n-1\}$. So in both cases the relation for $s^{bf}$ contains $n$ tuples. But now Magic Sets goes on to substitute magic predicates into the internal rules of the program as well, yielding:

$$\begin{aligned} s^{bf}(X,Y) &:- m\_s^{bf}(X), e(X,Z), t^{bf}(Z,Y). \\ t^{bf}(X,Y) &:- m\_t^{bf}(X), e(X,Z), s^{bf}(Z,Y). \end{aligned}$$

Here $s^{bf}$ contains the tuples $\{(x_{2i}, x_{2j}) \mid 0 \leq i < j \leq n\}$, while $t^{bf}$ contains the tuples $\{(x_{2i-1}, x_{2j-1}) \mid 1 \leq i < j \leq n\}$. These two relations together contain exactly $n^2$ tuples, an order of magnitude larger than the relations in the program transformed by argument reduction. □

# 5 Conclusion

We have defined a class of Datalog programs, the regular path programs, whose efficiency of evaluation can often be improved significantly by applying the technique of argument reduction. This class of programs is incomparable to previous classes to which this technique has been applied.

It has been shown that bottom-up evaluation using the Magic Sets transformation followed by semi-naive evaluation is never less efficient than top-down evaluation [Ullm89]. On the other hand, Magic Sets does not perform as well as it might for certain classes

of programs and it is highly unlikely that any single method will prove to be the most efficient for all programs. This has led to the search for subclasses of programs which are amenable to specialised techniques for improving evaluation efficiency. One such class of programs are the RLC-stable programs [NRSU89b]. The regular path programs defined in this paper now provide another such class.

One obvious area of future research is to attempt to establish whether the class of regular path programs can be integrated with the RLC-stable programs, thereby producing a strictly broader class of programs to which argument reduction can be applied.

## References


[BMSU86] F. Bancilhon, D. Maier, Y. Sagiv and J.D. Ullman, "Magic Sets and Other Strange Ways To Implement Logic Programs," *Proc. 5th ACM Symp. on Principles of Database Systems*, 1986, pp. 1–15.

[BR87] C. Beeri and R. Ramakrishnan, "On the Power of Magic," *Proc. 6th ACM Symp. on Principles of Database Systems*, 1987, pp. 269–283.

[Naug87] J.F. Naughton, "One-Sided Recursions," *Proc. 6th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems*, 1987, pp. 340–348.

[Naug88] J.F. Naughton, "Compiling Separable Recursions," *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 1988, pp. 312–319.

[NRSU89a] J.F. Naughton, R. Ramakrishnan, Y. Sagiv and J.D. Ullman, "Efficient Evaluation of right-, left-, and combined-linear rules," *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 1989, pp. 235–242.

[NRSU89b] J.F. Naughton, R. Ramakrishnan, Y. Sagiv and J.D. Ullman, "Argument Reduction by Factoring," *Proc. 15th Int. Conf. on Very Large Data Bases*, 1989, pp. 173–182.

[Ullm88] J.D. Ullman, "*Principles of Database and Knowledge-Base Systems, Vol. I & II*," Computer Science Press, Potomac, Md., 1988.

[Ullm89] J.D. Ullman, "Bottom-up Beats Top-down for Datalog," *Proc. 8th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems*, 1989, pp. 140–149.

[Wood90] P.T. Wood, "Factoring Augmented Regular Chain Programs," *Proc. 16th Int. Conf. on Very Large Data Bases*, 1990, pp. 255–263.