Overbound and Right-Linear Queries

Presented at PODS 1991

Inderpal Singh Mumick*
Computer Science Department
Stanford University
Stanford, CA 94305, USA.
mumick@cs.stanford.edu

Hamid Pirahesh
IBM Almaden Research Center
650 Harry Road, K55/801
San Jose, CA 95120, USA.
pirahesh@ibm.com

Abstract

We study the optimization of a useful subclass of linear recursive queries. In addition to the general purpose magic-sets transformation, special optimization techniques, such as the right-linear and context right-linear transformation have been proposed for such programs. These, and in fact most optimization techniques for database queries are based on the idea of full sips, where all available predicates are passed into a subgoal. This is widely believed to be a sound heuristic. However, we use partial sips, that do not pass all predicates into a subgoal, to develop an optimization algorithm for a class of linear queries and show that our algorithm is better than the previously known techniques. We syntactically identify a class of programs for which partial sips are always preferable to full sips. The result provokes thought on the utility of partial sips, and will hopefully lead to more research in the area.

The right-linear and context right-linear transformations, where applicable, are alternatives to the magic-sets transformation. In the presence of alternative optimizations, we need a way to choose between them. Given a program, can we choose one technique over another, confident that we are not discarding a good evaluation strategy? For example, it has been shown that, where applicable, the right-linear transformation always performs better than magic-sets. However the above question has been left unanswered for many of the proposed optimizations for special forms of recursion. We study the problem for one class of optimization techniques, comparing the context right-linear and magic-sets transformations.

Finally, we observe that right-linear and left-linear programs can be viewed as duals of each other. Every right-linear program can be written as an equivalent left-linear program, and vice versa.

*Part of this work was done at the IBM Almaden Research Center. Work at Stanford was supported by an NSF grant IRI-87-22886, an Air Force grant AFOSR-88-0266, and a grant of IBM Corporation.

1 Introduction

Magic-sets ([Ull89], Chapter 13) is a general purpose query optimization technique that rewrites a given program and query so that a bottom-up evaluation on the rewritten program is more efficient than on the original program, due to a reduction in the number of tuples computed and looked up in the database. Magic-sets has been recognized to be of extreme importance ([BR88, MFPR90]), and many new database systems (Starburst [HCL$^+$90], for example) use the magic-sets transformation as a basic optimization.

However, special classes of recursive queries have been identified for which magic-sets may not be the best rewrite strategy. The right-linear queries, defined by Naughton *et. al.* [NRSU89a], is one such class. The right-linear class is very important, since many common recursions (bill-of-materials, transitive closures) are right-linear. Naughton *et. al.* gave a right-linear (RL) transformation for the case of a single initial value in the query to a right-linear program. Kemp, Rao, and Somogyi [KRS90], and Mumick and Pirahesh [MP91] independently developed a *context right-linear* (CRL) transformation that is applicable when a query to a right-linear program has multiple initial values, and we want to link the answer tuples to the matching initial value. CRL is very important, since most queries on right-linear programs are likely to have multiple initial values.

Overbound Queries

To motivate partial sips, consider the transitive closure query ?- $t(u, v)$ with both arguments bound. The transitive closure program can be written in a right-linear fashion. Yannakakis [Yan90] shows that when computing a transitive closure, it is not profitable to use both destination and source bindings. Only one of the bindings can be used, and the interesting question for such queries is: do we search from the given source values, or do we search from the given destination values? However RL, CRL and the commonly used magic-sets transformations are not able to recognize this fact. They are based on the idea of full sips (sideways information passing strategies [Ull89], Chapter 12), and they try to use all available restrictions. As a result, they may compute a lot of redundant tuples. Can we improve RL, CRL and magic-sets by using partial sips and passing only some of the predicates? Given a query, can we find out which bindings are not useful in restricting search through the database? Is there a class of programs where partial sips are guaranteed to give better results than full sips?

In the first part of this paper, we define overbound right-linear queries as those that have too many arguments bound. A mechanism to recognize that only some of the bindings are useful in restricting search, so that a full sips is undesirable, is given. We present *new algorithms* that use partial sips for better optimization of overbound queries, and show that our *context overbound right-linear* transformation is better than CRL on all overbound programs, and is better than magic-sets under certain conditions.

Right-Linear Queries
The magic-sets transformation is applicable for all queries on a right-linear program. When the query has a single initial value, the RL transformation is also applicable. Naughton *et. al.* showed that RL *dominates* magic-sets on all right-linear programs, due to a reduction in the arity of the recursive predicate. The systems impact was very clear: if your query is right-linear with a single initial value, then you should do the right-linear transformation instead of magic-sets.

When a right-linear query has multiple initial values, CRL is an alternative to magic-sets. CRL may not reduce the arity of the recursive predicate, so that it is not clear whether the context right-linear transformation dominates magic-sets. Kemp *et. al.* believe that the cost of CRL will never be much worse than magic-sets, but that is not correct (Examples 3.2 and 3.3). So, when optimizing a right-linear query with multiple initial values, how should we choose between these two techniques? When can we be certain that the CRL will do better than magic-sets? Can we syntactically identify certain queries on which CRL is to be preferred?

In the second part of this paper we analytically compare the performance of CRL with magic-sets, and show that depending on the database and the set of initial values, either could be better than the other by an order of magnitude. Magic-sets does better than CRL in the following scenarios: (1) The right-linear traversal of the underlying database graph is more efficient than a left-linear traversal, or (2) Fewer tuples are computed by the magic-sets transformation.

We propose a simpler cost metric that discounts the effect of right-linear traversal of the graph. The simplification involves counting the number of derived tuples rather than number of tuple derivations. The simpler cost metric can be justified for databases satisfying certain properties, as well as for special transitive closure algorithms proposed by Jakobsson [Jak91]. We then define a class of *Stable Right-Linear Programs* for which CRL dominates magic-sets under the simpler cost metric. The stable right-linear class is interesting and useful. For example, transitive closure programs and many bill-of-material queries are stable right-linear. The idea of stability can be extended to other forms of linear recursion.

Duality
In the third part of the paper, we look upon CRL as the translation of a right-linear program into a left-linear program, followed by a commutation of the selection condition with the least fix-point operator [AU79]. We use equivalence between right-linear and left-linear regular grammars to prove that every right-linear program can be written as a left-linear program. The conversion from left-linear to right-linear is also possible, but not very attractive from an optimization viewpoint as the selections can no longer be applied before the recursion.

The paper is organized as follows. After some preliminary definitions and examples in Section 1.1, we introduce overbound queries in Section 2 and give optimization algorithms for overbound queries. The efficiency of context right-linear transformation is compared to magic-sets in Section 3. We introduce a simpler cost metric and give the conditions under which it is applicable. Then, Section 4 defines a class of stable right-linear programs. We prove that for stable programs, the context right-linear transformation never loses to magic-sets under the simpler cost model, while it can outperform magic-sets by $O(n)$. Section 5 presents the duality between right-linear and left-linear programs. Conclusions are presented in Section 6.

1.1 Background

In this paper we consider Datalog programs with one idb predicate, u. We assume that every argument of each u literal is a distinct variable. We assume the reader is familiar with adornments, sips, magic-sets, conjunctive query containment and other concepts covered in [Ull89].

For our analytical performance analysis, the cost of evaluating a program is the number of successful tuple deductions during a semi-naive evaluation of the program. Ullman ([Ull89], Chapter 13.3) argues for such a cost model. Certain reasonable assumptions about presence of indices are made, and the cost of failed deductions is charged to the tuples derived earlier that trigger the failed deductions.

To compare the efficiency of optimization methods, we introduce the concepts of domination and strong domination.

Definition 1.1 Domination and Strong Domination: An optimization method $A1$ *dominates* another optimization method $A2$ on a given program P if the cost of evaluating P according to $A1$ is no more than the cost of evaluating P according to $A2$, for every instance of the edb.

An optimization method $A1$ *strongly dominates* another optimization method $A2$ on a given program P if $A1$ dominates $A2$ on P *and* if there is some query and some edb for which the cost of evaluating P according to $A1$ is better than the cost of evaluating P according to $A2$ by more than a constant factor (such as $O(n)$). □

Given an extensional (edb, or stored) relation $par(X, Y)$, defining parent of X is Y, the following recursive program P produces an intensional (idb, or derived) relation $anc(X, Y)$, defining ancestor of X is Y. Note that we use a letter, such as P, to represent a program comprised of rules $P1, P2, \ldots$

$$(P1): \quad anc(X, Y) :- par(X, Y).$$
$$(P2): \quad anc(X, Y) :- par(X, Z) \ \& \ anc(Z, Y).$$

Assume par has e tuples with the domain size of either argument being n. Queries over anc usually have some restrictions on the ancestors, children, or both. Restricting children in query Qa:

$$(Qa): \quad ?\text{-}\ init(X) \ \& \ anc^{bf}(X, Y).$$

causes the first argument of anc to be *bound* to the set of $init$ (initial) values. The anc goal is *adorned* anc^{bf}. The adornment bf is a labeling of the anc predicate indicating that the first argument is bound and the second argument is free.

A recursive rule is called linear if the rule body contains only one predicate that is mutually recursive with the head predicate. For example, rule $P2$ is linear. In rule $P2$, if we adorn the head anc^{bf}, and place the recursive anc subgoal *rightmost* in the body, we get the same bf adornment for the anc subgoal. Rule $P2$ is therefore said to be *right*-linear.

Definition 1.2 Right-Linear Programs [NRSU89a]: Consider a program for a single idb predicate u and a goal u^α. We assume, without loss of generality, that the adornment α declares the first k arguments of u to be bound, and the remaining l, $l \geq 0$, arguments to be free. The program is said to be right-linear if each recursive rule can be written as:

$$(gr): \quad u^\alpha(X_1, \ldots, X_k, Y_1, \ldots, Y_l) :- first(X_1, \ldots, X_k, Z_1, \ldots, Z_k) \ \& \ u^\alpha(Z_1, \ldots, Z_k, Y_1, \ldots, Y_l).$$

and each nonrecursive exit rule can be written as:

$$(ge): \quad u^\alpha(X_1, \ldots, X_k, Y_1, \ldots, Y_l) :- exit(X_1, \ldots, X_k, Y_1, \ldots, Y_l).$$

where (1) each Y variable appears exactly twice in the recursive rule, once in the head and once in the recursive subgoal, in exactly the same argument position, and (2) $first$ and $exit$ are conjunctive queries of edb and built-in predicates. A Z_i variable can be equal to an X_j variable, in which case Z_i will not occur in $first$, and X_j may or may not occur in $first$. □

[NRSU89a] also defines left-linear and mixed-linear programs. Informally, a rule is left-linear if the head adornment can be preserved by placing the recursive subgoal leftmost. A mixed linear program consists of a combination of right-linear and left-linear rules.

Definition 1.3 Left-Linear Programs [NRSU89a]: Consider a program for a single idb predicate u and a goal u^α. We assume, without loss of generality, that the adornment α declares the first k arguments of u to be bound, and the remaining l, $l \geq 0$, arguments to be free. The program is said to be left-linear if each recursive rule can be written as:

$$(gl): \quad u^\alpha(X_1, \ldots, X_k, Y_1, \ldots, Y_l) :- u^\alpha(X_1, \ldots, X_k, Z_1, \ldots, Z_l) \ \& \ last(Z_1, \ldots, Z_l, Y_1, \ldots, Y_l).$$

and each non recursive rule is of the form ge in Definition 1.2, where (1) each X variable appears exactly twice in the recursive rule, once in the head and once in the recursive subgoal, in exactly the same argument position, and (2) $last$ and $exit$ are conjunctive queries of edb and built-in predicates. A Z_i variable can be equal to a Y_j variable, in which case Z_i will not occur in $last$, and Y_j may or may not occur in $last$.

If an X variable additionally occurs in $last$, the program is said to be *pseudo left-linear.* □

EXAMPLE 1.1 (Magic-sets for right-linear programs)
For query Qa on the ancestor program P, magic-sets computes a magic set m_anc^{bf} of all the ancestors of the initial values, and then computes ancestors for the magic values (Program Ma).

$$(Ma1): \quad anc^{bf}(X, Y) :- m\_anc^{bf}(X) \ \& \ par(X, Y).$$
$$(Ma2): \quad anc^{bf}(X, Y) :- m\_anc^{bf}(X) \ \& \ par(X, Z) \ \& \ anc^{bf}(Z, Y).$$

$$(Ma3): \quad m\_anc^{bf}(X) :- init(X).$$
$$(Ma4): \quad m\_anc^{bf}(Z) :- m\_anc^{bf}(X) \ \& \ par(X, Z).$$

Given t initial values, program Ma could compute $O(n)$

magic tuples in $O(e)$ time, and $O(n^2)$ anc tuples in $O(ne)$ time. □

Definition 1.4 Magic-Sets Transformation for Right-Linear Programs: On a right-linear program, the magic-sets transformation works as follows:

- The query
  ?- $init(X_1, \ldots, X_k)$ & $u^\alpha(X_1, \ldots, X_k, Y_1, \ldots, Y_l)$
  generates the nonrecursive magic rule:

  $(M1)$: $m\_u^\alpha(X_1, \ldots, X_k)$:- $init(X_1, \ldots, X_k)$.

- Every right-linear rule of the form gr in Definition 1.2 is transformed into the following two rules:

  $(M2)$: $m\_u^\alpha(Z_1, \ldots, Z_k)$:-
  $m\_u^\alpha(X_1, \ldots, X_k)$ &
  $first(X_1, \ldots, X_k, Z_1, \ldots, Z_k)$.

  $(M4)$: $u^\alpha(X_1, \ldots, X_k, Y_1, \ldots, Y_l)$:-
  $m\_u^\alpha(X_1, \ldots, X_k)$ &
  $first(X_1, \ldots, X_k, Z_1, \ldots, Z_k)$ &
  $u^\alpha(Z_1, \ldots, Z_k, Y_1, \ldots, Y_l)$.

- The general nonrecursive rule of the form ge in Definition 1.2 is transformed into:

  $(M3)$: $u^\alpha(X_1, \ldots, X_k, Y_1, \ldots, Y_l)$:-
  $m\_u^\alpha(X_1, \ldots, X_k)$ &
  $exit(X_1, \ldots, X_k, Y_1, \ldots, Y_l)$.

□

EXAMPLE 1.2 (Right-linear transformation)
Naughton, Ramakrishnan, Sagiv, and Ullman ([NRSU89a]) observed that if there is only one initial value (say 5), then rule $Ma1$ can be used to get all the answers to the query, so that rule $Ma2$ can be eliminated. As a result we never compute the ancestor tuples for ancestors of 5, potentially reducing the number of computed tuples by $O(n)$. Their right-linear (RL) transformation rewrites program P for query Qb as follows:

(Qb): ?- $anc^{bf}(5, Y)$.

$(Rb1)$: $anc^{bf}(5, Y)$:- $m\_anc^{bf}(X)$ & $par(X, Y)$.

$(Rb2)$: $m\_anc^{bf}(5)$.
$(Rb3)$: $m\_anc^{bf}(Z)$:- $m\_anc^{bf}(X)$ & $par(X, Z)$.

Program Rb could compute $O(n)$ magic tuples and $O(n)$ anc tuples in $O(e)$ time. □

Definition 1.5 Right-Linear Transformation — RL ([NRSU89a]): The RL transformation for an arbitrary right-linear program is as follows:

- Given constants $x_1, \ldots, x_k$, the query
  ?- $u^\alpha(x_1, \ldots, x_k, Y_1, \ldots, Y_l)$
  generates the single magic fact:

  $(R1)$: $m\_u^\alpha(x_1, \ldots, x_k)$.

- Every right-linear rule of the form gr in Definition 1.2 is transformed into:

  $(R2)$: $m\_u^\alpha(Z_1, \ldots, Z_k)$:- $m\_u^\alpha(X_1, \ldots, X_k)$ &
  $first(X_1, \ldots, X_k, Z_1, \ldots, Z_k)$.

- Every exit rule of the form ge in Definition 1.2 is transformed into:

  $(R3)$: $u^\alpha(x_1, \ldots, x_k, Y_1, \ldots, Y_l)$:-
  $m\_u^\alpha(X_1, \ldots, X_k)$ &
  $exit(X_1, \ldots, X_k, Y_1, \ldots, Y_l)$.

□

RL differs from magic-sets in that it eliminates the recursive rule $M4$, and modifies rule $M3$ to $R3$.

EXAMPLE 1.3 (Context right-linear transformation)
On program P and query Qa, context right-linear transformation produces the program Ca:

$(Ca1)$: $anc^{bf}(U, Y)$:- $cm\_anc^{bf}(U, X)$ & $par(X, Y)$.

$(Ca2)$: $cm\_anc^{bf}(X, X)$:- $init(X)$.
$(Ca3)$: $cm\_anc^{bf}(U, Z)$:- $cm\_anc^{bf}(U, X)$ & $par(X, Z)$.

We build a context magic set $cm\_anc^{bf}(U, X)$ that stores the initial U value (the context) from which an ancestor X is derived. This helps us link the answers to the correct initial value in rule $Ca1$.

Given t initial values, Program Ca could compute $O(tn)$ context magic tuples in $O(te)$ time. Only answer tuples are computed for anc. □

Definition 1.6 Context Right-Linear Transformation — CRL ([KRS90], [MP91]): The CRL transformation for an arbitrary right-linear program is as follows:

- The query
  ?- $init(X_1, \ldots, X_k)$ & $u^\alpha(X_1, \ldots, X_k, Y_1, \ldots, Y_l)$
  generates the nonrecursive magic rule:

  $(C1)$: $cm\_u^\alpha(X_1, \ldots, X_k, X_1, \ldots, X_k)$:-
  $init(X_1, \ldots, X_k)$.

- Every right-linear rule of the form gr in Definition 1.2 is transformed into:

  $(C2)$: $cm\_u^\alpha(U_1, \ldots, U_k, Z_1, \ldots, Z_k)$:-
  $cm\_u^\alpha(U_1, \ldots, U_k, X_1, \ldots, X_k)$ &
  $first(X_1, \ldots, X_k, Z_1, \ldots, Z_k)$.

- Every exit rule of the form ge in Definition 1.2 is transformed into:

  $(C3)$: $u^\alpha(U_1, \ldots, U_k, Y_1, \ldots, Y_l)$:-
  $cm\_u^\alpha(U_1, \ldots, U_k, X_1, \ldots, X_k)$ &
  $exit(X_1, \ldots, X_k, Y_1, \ldots, Y_l)$.

□

cm_u^α is the context magic predicate, with the following interpretation: $cm\_u^\alpha(\overline{U}, \overline{X})$ holds if $\overline{X}$ is a magic value by virtue of $\overline{U}$ appearing in the initial set. When computing the answers through rule $C3$, we can now link each answer $\overline{Y}$ value to the correct initial $\overline{U}$ value, since $\overline{U}$ is available in the context magic tuple. Thus, only the u^{α} tuples required in the answer are computed by program C.

EXAMPLE 1.4 (Full and partial sips) A sips[1] specifies how bindings are to be propagated in a rule. Consider query Qc:

$$(Qc):\quad ?\text{-}\ init(X,Y)\ \&\ anc^{bb}(X,Y).$$

with bindings on source and destination nodes, generating the adornment anc^{bb}. A *full sips* tries to pass all the bindings into the *anc* subgoal, so that rule $P2$ is rewritten as:

$$(Pc2):\quad anc^{bb}(X,Y) :\!-\ par(X,Z)\ \&\ anc^{bb}(Z,Y).$$

A *partial sips* holds back some of the bindings. A partial sips that does not pass bindings on the second column, rewrites $P2$ as:

$$(Pc2'):\quad anc^{bb}(X,Y) :\!-\ par(X,Z)\ \&\ anc^{bf}(Z,Y).$$

□

2 Overbound Queries

A commonly used heuristic in query optimization is that all available bindings should be used to restrict computation. Such a strategy is called *full sips*. Full sips are widely used in systems (in *NAIL!* [MNS+87] and Starburst [MFPR90], for example). RL and CRL are based on full sips, and magic-sets is usually done with respect to a full sips. However, we will see in Example 2.1 that sometimes all the bindings are not useful in restricting computation, and attempts to use the extra bindings can make the computation inefficient. We should use partial sips for such programs. In this section we show how to identify right-linear programs suitable for partial sips, how to select a partial sips, and how to optimize using the partial sips.

EXAMPLE 2.1 Consider the transitive closure query Qc on the ancestor program P with bindings for both sources and destinations:

$$(Qc):\quad ?\text{-}\ init(X,Y)\ \&\ anc^{bb}(X,Y).$$

Let there be $t = O(n^2)$ tuples in the initial set *init*, with $t_1 = O(n)$ different sources, and $t_2 = O(n)$ different destinations. As discussed in Section 1, query Qc contains more bindings than can be effectively used. Yannakakis [Yan90] shows that a full transitive closure (anc^{ff}) can be computed in time $O(ne)$, where n is the number of nodes, and e is the number of edges (number of tuples in *par*). A single source transitive closure can be computed in time $O(e)$, so if we have t_1 sources, we should be able to compute the answer in $O(t_1 e)$.

Since Program P is right-linear with respect to adornment[2] bb, CRL can be applied. CRL tries to use bindings on both arguments to get the following program Cc:

$$(Cc1):\quad anc^{bb}(U,Y) :\!-\ cm\_anc^{bb}(U,Y,X,Y)\ \&\ par(X,Y).$$
$$(Cc2):\quad cm\_anc^{bb}(X,Y,X,Y) :\!-\ init(X,Y).$$
$$(Cc3):\quad cm\_anc^{bb}(U,Y,Z,Y) :\!-\ cm\_anc^{bb}(U,Y,X,Y)\ \&\ par(X,Z).$$

Program Cc forces an evaluation where we traverse the graph from the sources (assume the first argument is the source) all the way up, carrying the destination values during the traversal. For each (U,Y) binding in the initial set, program Cc links the (U,Y) value to each ancestor of U, computing $O(tn)$ tuples in $O(te)$ time. If $t = O(n^2)$, CRL performs poorly.

Performance can be improved if we do not carry the destination values during the upward traversal of the graph. Consider program Oc below where we disregard the binding on Y column for a moment, and solve as for anc^{bf}.

$$(Oc0):\quad anc^{bb}(U,Y) :\!-\ cm\_anc^{bf}(U,X)\ \&\ par(X,Y).$$

$$(Oc2):\quad cm\_anc^{bf}(X,X) :\!-\ init(X,Y).$$
$$(Oc3):\quad cm\_anc^{bf}(U,Z) :\!-\ cm\_anc^{bf}(U,X)\ \&\ par(X,Z).$$

Program Oc generates extra anc^{bb} tuples compared to program Cc, since the bindings on Y have not been used to restrict anc^{bb}. We can avoid computing the extra anc^{bb} tuples if we push the initial bindings into rule $Oc0$ to derive the rule:

$$(Oc1):\quad anc^{bb}(U,Y) :\!-\ init(U,Y)\ \&\ cm\_anc^{bf}(U,X)\ \&\ par(X,Y).$$

Program Oc now consists of rules $Oc1, Oc2$, and $Oc3$. Rule $Oc0$ is discarded. Program Oc computes $O(t_1 n)$ tuples in time $O(t_1 e)$. Each of the e tuples of *par* matches with at most t_1 tuples of cm_anc^{bf}, so cm_anc^{bf} and *par* in rule $Oc1$ can be joined in $O(t_1 e)$. Each tuple in the result of the join matches at most one tuple of *init*, so all three relations can be joined in time $O(t_1 e)$. Performance of program Oc is thus the best we could expect. We claim program Oc strongly dominates program Cc. The number of deductions for anc^{bb} tuples is the same in the two programs, but the number of deductions for context magic tuples is always less in program Oc, sometimes by a factor of $O(t)$ (as when the initial bindings are $\{11, 12, \ldots, 1t\}$).

If $t = 1$, we can improve the efficiency of the RL transformation on P by a constant factor by using the above partial sips. The arity of the context magic predicate will be reduced, but the number of deductions of context magic tuples will be the same. □

2.1 Overbound Right-Linear Programs

We generalize the observations in Example 2.1 to a class of programs where partial sips are desirable. We need

[1] sips is an acronym for *s*ideways *i*nformation *p*assing *s*trategy/*s*trategies.

[2] In general, if a program P is right-linear with respect to adornment α, it is also right-linear with respect to an adornment more bound than α.

two simple operations on adornments.

A bbf adornment gives bindings to the first two arguments of a tertiary predicate. We can *split* the bindings by taking a projection on each of the first two argument, corresponding to bff and fbf adornments. We define an operator $+$, such that $bbf = bff + fbf$, and say that bbf can be *split* into bff and fbf adornments. A split is *vacuous* if one of the resulting adornments has all arguments free. For example, $bbf = bbf + fff$ is a vacuous split. A split is *useful* if it is not vacuous. As another example, bb can be usefully split into bf and fb.

If u is right-linear with respect to adornment bbf, we refer to its bound arguments (first two) as right-linear arguments. If u is left-linear with respect to fbb, we refer to its bound arguments (last two) as left-linear arguments.

The key property of the anc^{bb} program that makes a partial sips attractive is that some of its right-linear arguments are also left-linear. anc is right-linear with respect to bb, but it is left-linear with respect to fb, so that its second argument is both right-linear and left-linear. We should not pass the bindings on left-linear arguments using the right-linear transformations. The class of *overbound right-linear programs* captures this condition.

Definition 2.1
Overbound Right-Linear Programs: Program S is overbound right-linear with respect to adornment α if

- S is right-linear with respect to adornment α, and
- Adornment α can be usefully split into adornments β and γ, such that S is right-linear with respect to adornment β, and left-linear with respect to adornment γ.

If S is not overbound right-linear with respect to adornment β, then $\beta + \gamma$ is the *best split* for α. □

EXAMPLE 2.2 The ancestor program P is overbound right-linear with respect to adornment bb, since P is right-linear with respect to adornment bb, and bb can be usefully split into bf, for which P is right-linear, and fb, for which P is left-linear. P is not overbound right-linear with respect to adornment bf, so $bf + fb$ is the best split.

If a program S is right-linear with respect to bff and left linear with respect to fbb, S is overbound right-linear with respect to bbb. bbb can be split into $\beta = bbf$ and $\gamma = ffb$, but this is not the best split, since S is overbound right linear for $\beta = bbf$ also (we can split β into $bff + fbf$ satisfying conditions of Definition 2.1). $bbb = bff + fbb$ is the best split. □

The preferred partial sips for overbound right-linear programs can be obtained from the best split $\alpha = \beta + \gamma$. Only the bindings on the right-linear arguments given by β should be used to restrict u.

Definition 2.2 Context Overbound Right-Linear Transformation ($CORL$): Let

?- $init(\overline{X_\beta}, \overline{X_\gamma})$ & $u^\alpha(\overline{X_\beta}, \overline{X_\gamma}, \overline{Y})$.

be the given overbound right-linear query, and let $\alpha = \beta + \gamma$ be the best split. $\overline{X_\beta}$ denotes arguments of u^α indicated bound by adornment β, $\overline{X_\gamma}$ denotes arguments of u^α indicated bound by adornment γ, and $\overline{Y}$ denotes the remaining arguments of u^α $CORL$ transformation works in two steps.

Step 1: Define

$$init_\beta(\overline{X_\beta}) \text{ :- } init(\overline{X_\beta}, \overline{X_\gamma}).$$

and apply the CRL transformation for u^β to program P, using $init_\beta(\overline{X_\beta})$ as the set of initial bindings.

Step 2: Rewrite each rule

$$u^\alpha(\overline{U_\beta}, \overline{X_\gamma}, \overline{Y}) \text{ :- } cm\_u^\beta(\overline{U_\beta}, \overline{X_\beta}) \;\&\; exit(\overline{X_\beta}, \overline{X_\gamma}, \overline{Y}).$$

for u^α obtained after Step 1 as:

$$u^\alpha(\overline{U_\beta}, \overline{X_\gamma}, \overline{Y}) \text{ :- } init(\overline{U_\beta}, \overline{X_\gamma}) \;\&\; cm\_u^\beta(\overline{U_\beta}, \overline{X_\beta}) \;\& \\ exit(\overline{X_\beta}, \overline{X_\gamma}, \overline{Y}).$$

□

EXAMPLE 2.3 Applying $CORL$ to the ancestor program P and the query Qc, Step 1 gives the program Oc consisting of rules $Oc0, Oc2$, and $Oc3$ on Page 5. In Step 2, we rewrite rule $Oc0$ into rule $Oc1$. □

Theorem 2.1 *For an overbound right-linear program P with query u^α, the $CORL$ transformation is correct and strongly dominates CRL.* □

Proof:
Correctness of $CORL$ follows from correctness of the two steps in Definition 2.2. Step 1 is correct due to the correctness of CRL. Step 2 corresponds to pushing selections into each rule for u^α. There are no recursive rules for u^α in the program produced by CRL transformation in Step 1, and therefore Step 2 preserves equivalence.
Efficiency: Consider a recursive rule of the form gr in the given program. Since u is left-linear with respect to adornment γ, the X variables $\overline{X_\gamma}$ cannot appear in the conjunctive query $first$.

Hence, if we apply the CRL transformation for adornment α, the bindings for $\overline{X_\gamma}$ are just carried along in each context magic tuple in the CRL transformation. If we apply the $CORL$ transformation, the $\overline{X_\gamma}$ arguments are simply dropped from the context magic tuples. Thus, the context magic set is cheaper to compute in the $CORL$ transformation. The number of deductions for u^α tuples is the same in both CRL and $CORL$ transformations.

$CORL$ essentially decomposes $cm\_u^\alpha(U_\beta, U_\gamma, X_\beta, U_\gamma)$ into $cm\_u^\alpha(U_\beta, X_\beta)$ and $init(U_\beta, U_\gamma)$, using the fact that the multi-valued dependency ($U_\beta \longrightarrow\!\!\!\!\!\rightarrow X_\beta \mid U_\gamma$) holds. The full context magic set is effectively reconstructed in Step 2 before generating the answers from the base rules, so that no extra u^α tuples are generated. □

Corresponding to the RL transformation for a single initial value ($t = 1$), we can define an *overbound right-linear* (ORL) transformation based on the partial sips obtained above. ORL improves on RL by a constant factor.

2.2 Magic-sets on Overbound Right-Linear Programs

EXAMPLE 2.4 Magic-sets transformation on overbound program P for query Qc generates:

$(Mc1)$: $anc^{bb}(X,Y)$:- $m\_anc^{bb}(X,Y)$ & $par(X,Y)$.
$(Mc2)$: $anc^{bb}(X,Y)$:- $m\_anc^{bb}(X,Y)$ & $par(X,Z)$ & $anc^{bb}(Z,Y)$.
$(Mc3)$: $m\_anc^{bb}(X,Y)$:- $init(X,Y)$.
$(Mc4)$: $m\_anc^{bb}(Z,Y)$:- $m\_anc^{bb}(X,Y)$ & $par(X,Z)$.

Program Mc computes $O(nt_2)$ tuples of m_anc^{bb} in time $O(et_2)$, and $O(t)$ tuples of anc^{bb} in time $O(et_2)$. As in CRL, destination values are carried along in the magic predicate. A partial sips can be used to ignore the destination bindings, generating the following program MOc:

$(MOc1)$: $anc^{bb}(X,Y)$:- $m\_anc^{fb}(Y)$ & $m\_anc^{bf}(X)$ & $par(X,Y)$.
$(MOc2)$: $anc^{bb}(X,Y)$:- $m\_anc^{bf}(X)$ & $par(X,Z)$ & $anc^{bb}(Z,Y)$.
$(MOc3)$: $m\_anc^{fb}(Y)$:- $init(X,Y)$.

$(MOc4)$: $m\_anc^{bf}(X)$:- $init(X,Y)$.
$(MOc5)$: $m\_anc^{bf}(Z)$:- $m\_anc^{bf}(X)$ & $par(X,Z)$.

Here, we have pushed the initial Y values (m_anc^{fb}) into rule $MOc1$. Program MOc computes the magic set more efficiently ($O(n)$ tuples in time $O(e)$) than program Mc, but spends more time computing anc^{bb} ($O(nt_2)$ tuples in time $O(et_2)$) due to a cross product between initial X and Y values.

The relative performance of programs Mc and MOc thus depends on the database. □

Let *overbound magic-sets* (MO) be the generalization, to all overbound right-linear programs, of the transformation described in the above example. Neither of $CORL$, MO, or magic-sets dominates each other. However, we have the following result:

Theorem 2.2 *If* $\mid \overline{U_\beta} \mid = O(1)$*, $CORL$ strongly dominates magic-sets and overbound magic-sets on every overbound right-linear program.* ($\mid \overline{U_\beta} \mid = t_1$ in our examples.) □

Theorem 2.2 states that if the number of sources (t_1) in the initial set is a constant, and correspondingly the number of destinations is $O(t)$, then $CORL$ will perform better than the alternative magic-sets optimizations.

Proof: Given a right-linear program for u^β, the cost of computing the context magic set can potentially be worse than the cost of computing the magic set by at most a factor of t_1 = number of initial values, since CRL may repeat the derivation of a magic tuple for every initial value. Thus, if $t_1 = O(1)$, the cost of computing the context magic-set in $CORL$ is of the same order as the cost of computing the magic-set in MO, which is no worse than the cost of computing the magic-set in the magic-sets transformation.

Again, with $t_1 = O(1)$, the number of deductions for u^α answer tuples in $CORL$ is of the same order as in the magic-sets transformation, which is no worse than the number of deductions for u^α in MO. □

2.3 Left-linear Transformation for Overbound Right-Linear Programs

The techniques we have considered ($CORL$, magic-sets, and MO) compute all ancestors of each initial source. Depending on the database and the initial values, it may be better to traverse the graph downwards from the initial destination values. A left-linear (LL) transformation, using only the bindings on the left-linear (γ) arguments will enable such an evaluation in time $O(t_2e)$. The overbound right-linear condition tells us that the program is simultaneously left-linear if we use partial sips, and that this option should be considered.

$CORL$ and LL can be viewed as duals of each other. On the ancestor program P and query Qc, $CORL$ computes all ancestors of each initial source, and then does a selection on initial destinations. LL computes all descendents of each initial destination, and then does a selection on initial sources.

3 Efficiency of Context Right-Linear Transformation

In the presence of multiple values in the initial binding set, there are two alternatives to the context right-linear transformation: (1) Apply RL multiple times, once for each initial value, and (2) Apply magic-sets.

3.1 Multiple Applications of the Right-Linear Transformation — MRL

CRL and MRL are very similar. For every initial value u, program Ca derives the context magic tuple $cm\_anc^{bf}(u,Z)$ iff the program Rb derives the magic tuple $m\_anc^{bf}(Z)$ during the invocation with initial value u. In fact CRL can be seen as the compiled version of MRL, where the looping variable (initial binding value) is compiled into the magic predicate.

The number of tuples deduced by CRL and MRL is thus the same. However, we would not be justified in using the number of deduced tuples as a metric to compare CRL and MRL, as this cost model ignores certain factors that are important in this comparison, such as the effects of set-oriented computation and buffering,

We claim that the evaluation of MRL is more expensive, since MRL forces a fragmented tuple-at-a-time computation, while CRL permits set-oriented computation. The substantial benefits of set-oriented computation over a fragmented computation are summarized by Mumick *et. al.* in [MFPR90]. Kemp *et. al.* [KRS90] provide experimental verification for our conclusions.

3.2 Magic-sets

Magic-sets and CRL both allow set-oriented computation. We are therefore justified in comparing magic-sets and CRL under the metric of number of tuple deductions.

Given a right-linear program S for some adornment α, let M be the magic-sets transformation and let C be the CRL transformation. Let u^α be the recursive predicate, and let m_u^α and cm_u^α be the magic and context magic predicates. u^α and m_u^α (or cm_u^α) are the only two idb predicates in program M (or C). The context magic set cm_u^α is more expensive to compute than the magic set m_u^α, since the arity of the former is double the arity of the latter. However, program M computes many u^α tuples that have non-initial magic values in the bound columns. Program C does not compute any such tuples. We will compare the total number of deductions in programs M and C.

Let there be t tuples in the initial set of bindings, $init$, and let program M make d' and h' derivations to compute d tuples for m_u^α and h tuples for u^α respectively, while program C makes D' and H' derivations to compute D tuples for cm_u^α and H tuples for u^α respectively. Further, Let f_i and f_o be the fan-in (in degree) and fan out (out degree) of the graph $\mathcal{G}_1$ constructed as follows: the nodes are k tuples, where k is the number of bound arguments in adornment α. For every conjunctive query $first$ in a recursive rule of the form gr, if $first(x_1, \ldots, x_k, z_1, \ldots, z_k)$ is true, there is an edge from node $(x_1, \ldots, x_k)$ to node $(z_1, \ldots, z_k)$. Let f_e be the fan-in of the graph $\mathcal{G}_2$ constructed as follows: the nodes are k tuples and l tuples, where k is the number of bound arguments and l the number of free arguments in adornment α. Edges are from k tuple nodes to l tuple nodes only. For every conjunctive query $exit$ in an exit rule of the form ge, if $exit(x_1, \ldots, x_k, y_1, \ldots, y_l)$ is true, there is an edge from node $(x_1, \ldots, x_k)$ to node $(y_1, \ldots, y_l)$. If we restrict the k tuple nodes in graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ to the d magic tuples, the largest possible value for f_i, f_o, and f_e is d. Then

- $d' \leq D' \leq td'$ and $d \leq D \leq td$ since each deduction for a magic tuple can be repeated t times, once for each initial value, and each magic tuple can be paired with each of the t initial values.
- $d' \leq f_i d$ and $D' \leq f_i D$ since in a left-linear evaluation each $m\_u^\alpha(x_1, \ldots, x_k)$ or $cm\_u^\alpha(u_1, \ldots, u_k, x_1, \ldots, x_k)$ tuple can be derived as many times as the indegree of $(x_1, \ldots, x_k)$.

Figure 1: *par* relation for Example 3.1

- $H \leq h \leq dH$. H is the number of answer tuples for query Q, and must be deduced in program M. Program M will usually produce extra u tuples by linking an answer $(y_1, \ldots, y_l)$ value to some of the d magic values that are not in the initial binding set.
- $H \leq H' \leq f_e H$ and $h \leq h' \leq f_o h$. CRL program may deduce an answer tuple f_e times from the exit rules. Magic program may deduce a u^α tuple f_o times since in a right linear evaluation, a $u^\alpha(x_1, \ldots, x_k, y_1, \ldots, y_l)$ tuple can be derived as many times as the outdegree of $x_1, \ldots, x_k$.
- $h_0 \leq H' \leq th_0$ where h_0 is the number of u^α tuples derived from the base rule of the magic program. Each $u^\alpha(x_1, \ldots, x_k, y_1, \ldots, y_l)$ tuple derived from the base rule of the magic program corresponds to a deduction for a $u^\alpha(u_1, \ldots, u_k, y_1, \ldots, y_l)$ in the base rule of CRL program, but the deduction may not be for a new $u^\alpha(u_1, \ldots, u_k, y_1, \ldots, y_l)$ tuple.

Thus, cost(magic-sets) $= d' + h' = O(f_i d + f_o h) = O(f_i d + f_o dH)$ is incomparable to cost(CRL) $= D' + H' = O(tf_i d + th_0)$ or $O(tf_i d + f_e H)$. CRL can thus be worse than magic-sets by at most a factor of t, the cardinality of the initial binding set, and magic-sets can be worse than CRL by a factor of d, the number of tuples in the magic-set, assuming $f_e = O(f_o)$.

EXAMPLE 3.1 (Magic-sets is worse by a factor of *d*)
For the ancestor program P and query Qa, let $par = \{i(i+1), ia_{i+1} \mid 1 \leq i < n\}$ (Figure 3.1), so that $f_i = f_e = 1, f_o = 2$, and let $init = \{1, a_2, a_3, \ldots a_t\}$.

The magic program Ma makes $d' = t + 2n - 2$ deductions for $d = 2n - 1$ magic tuples, $m\_anc^{bf} = \{1\} \cup \{i, a_i \mid 1 < i \leq n\}$, and $h' = n^2 - n$ deductions for $h = n^2 - n$ ancestor tuples, $anc^{bf} = \{ia_j, ij \mid 1 \leq i < n, i < j \leq n\}$. Total Cost $= O(n^2)$.

The CRL program Ca makes $D' = t + 2n - 2$ deductions for $D = t + 2n - 2$ context magic tuples, $cm\_anc^{bf} = \{1j, 1a_j \mid 1 < j \leq n\} \cup \{a_k a_k \mid 1 < k \leq t\} \cup \{11\}$, and $H' = 2n - 2$ deductions for $H = 2n - 2$ ancestor tuples, $anc^{bf} = \{1j, 1a_j \mid 1 < j \leq n\}$. Total Cost $= O(t + n) = O(n)$.

Magic-sets is thus worse by a factor proportional to $n = \Omega(d)$. □

Figure 2: *par* (indicated by solid lines) and *royal_par* (indicated by dotted lines) relations for Example 3.2

EXAMPLE 3.2 (*CRL* **is worse by a factor of** t)
Consider the following program S to compute the ancestor relation for people having a royal ancestor. Program S differs from P in that the edb predicates in the recursive and exit rules are different.

$(S1)$: $r\_anc(X,Y)$:- $royal\_par(X,Y)$.
$(S2)$: $r\_anc(X,Y)$:- $par(X,Z)$ & $r\_anc(Z,Y)$.
(Q): ?- $init(X)$ & $r\_anc(X,Y)$.

Let $par = \{i(i+1) \mid 1 \leq i < n\}$ and $royal\_par = \{n(\text{akbar})\}$ (Figure 3.2), so that $f_i = f_o = f_e = 1$. Let $init = \{1, 2, \ldots t\}$.

The magic program will make $d' = n$ deductions for $d = n$ magic tuples, $m\_r\_anc^{bf} = \{1, 2, \ldots, n\}$, and $h' = n$ deductions for $h = n$ ancestor tuples, $r\_anc^{bf} = \{i(\text{akbar}) \mid 1 \leq i \leq n\}$. Total Cost $= O(n)$.

The CRL program will make $D' = nt - (t^2 - t)/2$ deductions for $D = nt - (t^2 - t)/2$ context magic tuples, $cm\_r\_anc^{bf} = \{ij \mid 1 \leq i \leq t, i \leq j \leq n\}$, and $H' = t$ deductions for $H = t$ ancestor tuples, $r\_anc^{bf} = \{i(\text{akbar}) \mid 1 \leq i \leq t\}$. Total Cost $= O(nt)$.

CRL is thus worse by a factor proportional to t.

Another example is the same generation program with both arguments bound (?- sg^{bb}). The CRL program could deduce $O(tn^2)$ tuples compared to $O(n^2)$ tuples deduced by magic-sets. □

EXAMPLE 3.3 (*CRL* **is worse by a factor of** t)
This example illustrates the effect of left-linear computation of context magic set in CRL vs a right-linear computation of answer tuples in magic-sets.

Consider the ancestor program P and query Qa of Example 3.1, but let $par = \{a_i b, bc_j, c_j d_k \mid 1 \leq i, j, k \leq n\}$ (Figure 3.3), so that $f_i = f_o = f_e = n$, and let $init = \{a_1, a_2, \ldots a_t\}$.

The magic program will make $d' = n^2 + n + 2t$ deductions for $d = 2n + t + 1$ magic tuples, $m\_anc^{bf} = \{a_i, b, c_j, d_k\}$[3], and $h' = 2n^2 + 2tn + n + t$ deductions for $h = n^2 + 2tn + 2n + t$ ancestor tuples, $anc^{bf} = \{a_i b, a_i c_j, a_i d_k, bc_j, bd_k, c_j d_k\}$. Total Cost $= O(n^2)$.

The CRL program will make $D' = tn^2 + tn + 2t$ deductions for $D = 2tn + 2t$ context magic tuples, $cm\_anc^{bf} = \{a_i a_i, a_i b, a_i c_j, a_i d_k\}$, and $H' = tn^2 + tn + t$ deductions for $H = 2tn + t$ ancestor tuples, $anc^{bf} = \{a_i b, a_i c_j, a_i d_k\}$. Total Cost $= O(tn^2)$.

CRL is thus worse by a factor of t.

[3] Unless otherwise stated, assume the following ranges for indices: $1 \leq i \leq t, 1 \leq j, k \leq n$.

Figure 3: *par* relation for Example 3.3

It is interesting to note that if we reverse the edges and let $init = \{d_1, d_2, \ldots d_t\}$, Cost($CRL$) $= O(tn)$ and Cost(Magic-sets) $= O(tn^2)$ □

Examples 3.2 and 3.3 refute the claim of Kemp *et. al.* [KRS90] that for large values of t, the cost of CRL will never be much greater than that of magic-sets. CRL will usually perform better than magic-sets in the following contexts. If the number of initial values (t) is small, CRL will outperform magic-sets since the degree of replication of the deductions for magic and answer tuples will be small. Again, if there are few common ancestors between the different initial values, little replication will result.

We now identify the reasons why magic-sets sometimes performs better than CRL, and propose a simpler cost metric to abstract away one of the factors. The simpler cost metric will be a reflection of the true cost in case the database graph has a small fan-in, or if we use specialized transitive closure algorithms. In Section 4 we define a class of programs for which CRL strongly dominates magic-sets under the simpler cost metric.

3.3 Number of Distinct Derived Tuples as a Cost Metric

Magic-sets performs better than CRL in both Examples 3.2 and 3.3. However, the reason for better performance are different in the two examples.

In Example 3.2, CRL actually computes more tuples than magic-sets. CRL derives a lot of extra context magic tuples that eventually do not contribute to the answer. The extra work done in computing the context magic set is usually paid for by reduced work in computing the answer tuples, but in this example, there are very few answer tuples. Thus, we learn that if the number of answer tuples is not large enough to compensate for the work done in deducing context magic tuples, CRL may not be a viable alternative to magic-sets.

In Example 3.3, CRL actually computes fewer tuples than magic-sets, but still loses because it repeats deductions due to a high indegree ($f_i = n$). The bulk of the work done by CRL is in computing the context magic set, and CRL does this computation using a left-linear recursion. Magic-sets does less work during the left-linear traversal, putting off work till a later phase when it computes a larger number of answer tuples using a right-linear recursion. It is known that the left linear traversal of a graph can be worse than a right-linear traversal by a factor of the indegree of the graph. Therefore it should be no surprise that CRL lost to magic-sets in Example 3.3. It is also known that the right linear traversal of a graph can be worse than a left-linear traversal by a factor of the outdegree of the graph. Thus, given a graph, it is usually difficult to determine which traversal will be preferable. In Example 3.3, we made CRL outperform magic-sets by simply reversing the direction of the edges.

Many common database applications have bounded degree graphs. For example, in bill-of-material databases, f_i is about 10 or 100, much less than the number of nodes in the database graph. The parental database has an outdegree of 2, and an indegree bounded by the maximum number of children a couple may have. If we restrict ourselves to such databases, the number of deductions is of the same order as the number of distinct derived tuples. In general, the number of deductions is more than the number of distinct derived tuples since for each derived tuple, there might be many deductions. For instance, the magic rule may derive the same magic value z from f_i magic values x_1, x_2, x_{f_i} if z has an indegree f_i with an edge coming in from each of the x_j values. In our ancestor example, a parent magic node may be derived from many children magic nodes. Thus, the number of deductions can be greater than the number of distinct derived tuples by a factor of f_i. If $f_i = O(1)$, Number of deductions $= O$(number of distinct derived tuples).

Thus, for bounded degree graphs, we could compare CRL and magic-sets by comparing the number of distinct tuples they derive. Using the notation of Section 3.2 we have Cost(magic-sets) $= O(d + dH)$ and Cost(CRL) $= O(td + H)$.

We want to use this cost metric to derive sufficient conditions under which CRL dominates magic-sets. There are certain performance advantages of CRL (one recursion compared to two recursions in the magic program) that are not captured by the number of distinct derived tuples or by the number of deductions. This doesn't bother us. If the cost model underestimates the cost of magic-sets, we will only derive stronger sufficient conditions, not incorrect weaker conditions.

The number of distinct tuples derived can be a meaningful cost metric even if the database graph does not have a bounded degree. The trick is to view the computation required to compute the CRL program as a transitive closure over the underlying database graph. Similarly, the computation required to compute the magic program can be viewed as a reachability problem followed by a transitive closure over the underlying database graph. The transitive closure can potentially be done more efficiently than either the right-linear or left-linear traversal. Jåkobsson [Jak91] presents such an algorithm. The algorithm as presented in [Jak91] is for the complete transitive closure, and a t source transitive closure algorithm is under development. Now, if we can show that, on a given program P, CRL requires you to compute a transitive closure for a subset of the sources required by magic-sets, and that every transitive closure tuple derived by CRL corresponds to a transitive closure tuple derived by magic-sets, we can be sure that CRL will dominate magic-sets on program P. Such a condition on the transitive closure depends on the number of distinct tuples derived by CRL and magic-sets.

4 Stable Right-Linear Programs

In this section, we use the number of distinct derived tuples as our cost metric.

We first develop a simple semantic condition under which CRL strongly dominates magic-sets, and then give a sufficient syntactic condition. Let a query Q on program P have D answer tuples, where D is also the number of context magic tuples in the CRL transformation of (P, Q). The magic-sets program for (P, Q) must also produce these D answer tuples. Thus even if the magic-sets program saves on computing magic tuples, the total number of tuples computed by the magic-sets program is at least half the number of tuples computed by the CRL program.

Lemma 4.1 *Given a right-linear program P, let the following condition hold for every query Q and every edb E :*

B1: *If the CRL transformed program for (P, Q), when evaluated on edb E, derives D context magic tuples, then the query Q, evaluated on program P and edb E, has at least $(D - t)$ answer tuples, where t is the number of initial values in query Q.*

Then, for program P, CRL can never be worse than magic-sets by a factor of more than 2. □

Proof: Using the notation of Section 3.2, Cost(CRL) $= (D+H)$. It is given that $H > (D-t)$, say $H = D - t + \Delta$. Thus, Cost(CRL) $= (2D + \Delta - t)$, $\Delta \geq 0$.
Cost(magic-sets) $= (d+h) \geq (d+H) = (d + (D - t + \Delta)) = ((d-t) + D + \Delta) \geq (D + \Delta)$, since $h \geq H$ and $d \geq t$.
It follows that Cost(CRL) $< 2 \star$ Cost(magic-sets). □

EXAMPLE 4.1 Consider the ancestor program P on Page 3 for an anc^{bf} adornment. Example 3.1 on page 8 described a query and an edb for which CRL performed better than magic-sets by $O(n)$.

For any query Q and edb E, let CRL generate D context magic tuples. If we discard the t tuples derived from the t initial values, we are left with $D-t$ context magic tuples of the form $cm\_anc^{bf}(U, Z)$ where U is an initial value, and Z is an ancestor of U. (U, Z) is therefore also an answer tuple, so $D-t$ answer tuples must be computed. By Lemma 4.1, it follows that, for program P, CRL can never do worse than magic-sets by more than a constant factor.

We conclude that CRL strongly dominates magic-sets for the ancestor program P for the anc^{bf} adornment. □

We generalize the observation in Example 4.1 to a class of programs for which CRL dominates magic-sets. The idea is that each context magic tuple derived by the CRL program must lead to an answer tuple. The definitions of stability that follow capture the condition $B2$:

B2: For a given $\overline{x}$ value, if we derive the binding $\overline{z}$ from the conjunctive query $first(\overline{X}, \overline{Z})$ in a recursive rule, then from the same $\overline{x}$ value we must derive a binding $\overline{y}$ from the conjunctive query $exit(\overline{X}, \overline{Y})$ in some nonrecursive rule, where $\overline{z}$ is the projection of $\overline{y}$ on a given projection list.

Definition 4.1 Projection Containment: Let $Q(X_1, \ldots, X_k, Z_1, \ldots, Z_m)$ and $S(X_1, \ldots, X_k, Y_1, \ldots, Y_l)$, $m \leq l$ be two conjunctive queries, and let $(i_1, \ldots, i_m)$ be a permutation of m numbers between 1 and l. Then, if there exists a permutation such that the following conjunctive query containment ([Ull89], Chapter 14) holds,

$$Q(X_1, \ldots, X_k, Z_1, \ldots, Z_m) \subseteq \prod_{X_1, \ldots, X_k, Y_{i_1}, \ldots, Y_{i_m}} S(X_1, \ldots, X_k, Y_1, \ldots, Y_l)$$

we say that query Q is *projection-contained* in query S with respect to arguments $X_1, \ldots, X_k$. We write this as:

$$Q(X_1, \ldots, X_k, Z_1, \ldots, Z_m) \sqsubseteq_{X_1, \ldots, X_m} S(X_1, \ldots, X_k, Y_1, \ldots, Y_l)$$

□

Definition 4.2 Stable Right-Linear Programs: A right-linear program P for an adornment $\alpha = b^k f^l$ is stable, if for every recursive rule of the form:

$$(gr): \quad u^{\alpha}(X_1, \ldots, X_k, Y_1, \ldots, Y_l) :- first(X_1, \ldots, X_k, Z_1, \ldots, Z_k) \ \& \ u^{\alpha}(Z_1, \ldots, Z_k, Y_1, \ldots, Y_l).$$

there exists a nonrecursive rule of the form

$$(ge): \quad u^{\alpha}(X_1, \ldots, X_k, Y_1, \ldots, Y_l) :- exit(X_1, \ldots, X_k, Y_1, \ldots, Y_l).$$

where the conjunctive query $first$ is projection-contained in the conjunctive query $exit$ with respect to the argument list $X_1, \ldots, X_k$. If a $Z_i = X_j$, Z_i will not occur in $first$, and that is fine. However, each X_j must occur in $first$. □

EXAMPLE 4.2 $par(X_1, Z_1) \sqsubseteq_{X_1} par(X_1, Y_1)$. The ancestor program P is thus stable right-linear for the adornment anc^{bf}. However, for the adornment anc^{bb}, P is not stable right-linear. $first(X_1, X_2, Z_1) = par(X_1, Z_1)$, and $first(X_1, X_2, Z_1) \not\sqsubseteq_{X_1, X_2} par(X_1, X_2)$. In fact, an overbound right-linear program cannot be stable. Program S in Example 3.2 is not stable since $par(X, Z) \not\sqsubseteq_X royal\_par(X, Y)$.

Consider a version of the ancestor program where we want to trace paternal ancestry. The program,

$p\_anc(X, Y) :- par(X, Y).$
$p\_anc(X, Y) :- par(X, Z) \ \& \ father(Z) \ \& \ p\_anc(Z, Y).$

is stable for adornment p_anc^{bf} since

$$(par(X_1, Z_1), father(Z_1)) \sqsubseteq_{X_1} par(X_1, Y_1).$$

We note that transitive closure programs are stable for the bf adornment, and consequently many bill-of-material queries are stable right-linear. □

Theorem 4.1 *CRL dominates magic-sets for Stable Right-Linear Programs.* □

Proof: Let P be a stable right-linear program for adornment α. Due to the projection containment required by Definition 4.2, condition $B2$ is satisfied. Consider a context magic tuple $cm\_u^{\alpha}(\overline{u}, \overline{z})$, $\overline{u} \neq \overline{z}$. This tuple was derived by an application of a rule of the form $C2$ in Definition 1.6, using a $cm\_u^{\alpha}(\overline{u}, \overline{x})$ tuple derived earlier, such that $first(\overline{x}, \overline{z})$ is true. By condition $B2$, $\overline{z}$ can be obtained by a projection on some $\overline{y}$ for which $exit(\overline{x}, \overline{y})$ is true. Hence each context magic tuple $cm\_u^{\alpha}(\overline{u}, \overline{z})$, $\overline{u} \neq \overline{z}$, contributes the answer tuple $u^{\alpha}(\overline{u}, \overline{y})$. With $N-t$ answer tuples, Lemma 4.1 implies that CRL is never worse than magic-sets by more than a constant factor.
A database and a query where CRL beats Magic-sets by a non-constant fcator can be constructed in a manner similar to Example 3.1. □

Corollary 4.1 *CRL dominates magic-sets for transitive closure programs for the adornment bf.* □

[KRS90] conduct experiments to compare the magic-sets and CRL transformations on the ancestor program for the adornment anc^{bf}, using a database with fan-in $f_i = 1$. Stability of the ancestor program predicts that CRL will be better than magic-sets on any database with a small fan-in, and their experiments support our conclusion. Our work shows that there is little need for experimental comparison on stable programs for bounded degree databases. Experiments should concentrate on unstable programs or on databases with large degree, assuming the standard right-linear and left-linear traversal algorithms are used..

Given a stable program, CRL and magic-sets programs can both be viewed as restricted source transitive closure problems. The set of sources in the CRL program is a subset of the source set in the magic program. Further, projection containment guarantees that every distinct reachable node from a source will correspond to a distinct answer tuple. With such a problem, specialized transitive closure algorithms referred to in Section 3.3 will evaluate CRL more efficiently, regardless of the degree of the database graph.

5 Duality of Right-Linear and Left-Linear Programs

Ullman and Van Gelder [UVG88] define chain programs and show that with every chain program S we can associate a context free grammar $G(S)$. Right-linear and left-linear programs can be viewed as chain programs if we collapse the conjunctive queries in the recursive rules into one edb predicate. When the chain program is right-linear (left-linear), the associated grammar is also right-linear (left-linear). Since for every right-linear (left-linear) grammar, there is an equivalent left-linear (right-linear) grammar, does it follow that for every right-linear (left-linear) program, there is an equivalent left-linear (right-linear) program?

The answer is yes. In fact, we can decompose the CRL transformation into two steps, first converting the given right-linear program into a left-linear one, and then commuting the selection with the left-linear recursion [AU79].

5.1 Right-Linear $\Rightarrow$ Left-Linear

EXAMPLE 5.1 Consider the ancestor program P for the adornment bf:

$(P1)$: $anc(X,Y)$:- $par_1(X,Y)$.
$(P2)$: $anc(X,Y)$:- $par_2(X,Z)$ & $anc(Z,Y)$.

We use indices to distinguish between the two occurences of par The associated right-linear grammar is $G(P,bf) = \{A \rightarrow p_1 \mid p_2 A\}$. G is equivalent to the left-linear grammar $G^l(P,bf) = \{A \rightarrow Mp_1; M \rightarrow \epsilon \mid Mp_2\}$. Introducing a predicate m for the grammar symbol M, we derive the following left-linear program corresponding to grammar G^l:

$(La1)$: $anc^{bf}(U,Y)$:- $m(U,X)$ & $par_1(X,Y)$.

$(La2)$: $m(X,X)$.
$(La3)$: $m(U,Z)$:- $m(U,X)$ & $par_2(X,Z)$.

Rule $La3$ for m is left-linear for adornment bf. Program La is not range-restricted,[4] but it can be made range-restricted by unfolding the usage of rule $La2$. Alternatively, the initial values in the query:

(Qa): ?- $init(X)$ & $anc^{bf}(X,Y)$.

restrict the first argument of m in rule $La1$, and can be pushed through the recursion into rule $La2$, obtaining

$(La2')$: $m(X,X)$:- $init(X)$.

The program consisting of rules $La1$, $La2'$, and $La3$ is identical to program Ca obtained by the CRL transformation in Example 1.3 (predicate $m \equiv$ predicate cm_anc^{bf}). □

EXAMPLE 5.2 Consider the ancestor program P for the adornment bb. The associated right-linear grammar is $G(T,bb) = \{A \rightarrow p_1 \mid p_2 A\}$, where p_2 is associated with the chain $< par_2(X_1,Z_1)$ & $Z_2 = X_2 >$. G is equivalent to the left-linear grammar $G^l(T,bb) = \{A \rightarrow Mp_1; M \rightarrow \epsilon \mid Mp_2\}$. The left-linear program corresponding to G^l is:

$(Lc1)$: $anc^{bb}(U_1,U_2)$:- $m(U_1,U_2,X_1,X_2)$ & $par_1(X_1,X_2)$.

$(Lc2)$: $m(U_1,U_2,U_1,U_2)$.
$(Lc3)$: $m(U_1,U_2,Z_1,Z_2)$:- $m(U_1,U_2,X_1,X_2)$ & $par_2(X_1,Z_1)$ & $Z_2 = X_2$.

Rule $Lc3$ for m is left-linear for adornment bb. Program Lc is not range-restricted, and cannot be made range-restricted by unfolding the usage of rule $Lc2$. However, the initial values in the query:

(Qc): ?- $init(X,Y)$ & $anc^{bb}(X,Y)$.

restrict the first two arguments of m in rule $Lc1$, and can be pushed through the recursion into rule $Lc2$, obtaining

$(Lc2')$: $m(U_1,U_2,U_1,U_2)$:- $init(U_1,U_2)$.

The program consisting of rules $Lc1$, $Lc2'$, and $Lc3$ is identical to program Cc obtained by the CRL transformation in Example 2.1 (predicate $m \equiv$ predicate cm_anc^{bb}). □

Lemma 5.1 *For every right-linear program, there exists an equivalent left-linear program.* □

Proof: Let program S be right-linear for adornment α, with recursive and exit rules of the form stated in Definition 1.2. $G(S,\alpha) = \{U \rightarrow f_1 U \mid f_2 U \mid \ldots \mid e_1 \mid e_2 \mid \ldots\}$ is the associated right-linear grammar. G is equivalent to the

[4] A rule is range-restricted if every variable in the head is restricted to a finite domain by virtue of appearing in an ordinary subgoal, or by being equated to a restricted variable.

left-linear grammar $G^l(S,\alpha) = \{U \rightarrow Me_1 \mid Me_2 \mid \ldots; M \rightarrow \epsilon \mid Mf_1 \mid Mf_2 \mid \ldots\}$. The chain program associated with G^l can be derived by (1) Introducing a predicate m of arity twice the number of bound arguments in adornment α, corresponding to the grammar symbol M, and (2) Writing a chain rule for each production rule in grammar G^l. Let program $\mathcal{L}(\mathcal{S})$ be the transformation of S obtained as above. $\mathcal{L}(\mathcal{S})$ is identical to the result of the CRL transformation (Definition 1.6), except for the initial rule for the context magic predicate, m, which now looks like:

$(E1)$: $\quad m(U_1, \ldots, U_k, U_1, \ldots, U_k).$

Predicate m in Program $\mathcal{L}(\mathcal{S})$ is left-linear for adornment α. □

It is profitable to convert right-linear programs into left-linear as then the selections can be commuted with the fix-point operation.

5.2 Left-Linear ⇒ Right-Linear

EXAMPLE 5.3 Now consider the ancestor program P for the adornment fb, as in query Qd. To convert P into the standard form of a chain program, we must switch the arguments of anc, par_1, and par_2. We get the following program R and query Qd', with predicates $a(X,Y)$, $q_1(X,Y)$, and $q_2(X,Y)$ corresponding to $anc(Y,X)$, $par_1(Y,X)$ and $par_2(Y,X)$ respectively.

$(R1)$: $\quad a^{bf}(X,Y)$:- $q_1(X,Y)$.
$(R2)$: $\quad a^{bf}(X,Y)$:- $a^{bf}(X,Z)$ & $q_2(Z,Y)$.

(Qd'): $\quad$?- $init(Y)$ & $a^{bf}(Y,X)$.

Program R is left-linear for adornment bf. The associated left-linear grammar is $G(R,bf) = \{A \rightarrow q_1 \mid Aq_2\}$. G is equivalent to the right-linear grammar $G^r(R,bf) = \{A \rightarrow q_1H; H \rightarrow \epsilon \mid q_2H\}$. The right-linear program corresponding to G^r is:

$(G1)$: $\quad a^{bf}(X,Y)$:- $q_1(X,Z)$ & $h(Z,Y)$.

$(G2)$: $\quad h(Y,Y)$.
$(G3)$: $\quad h(X,Y)$:- $q_2(X,Z)$ & $h(Z,Y)$.

Rule $G3$ for h is right-linear for adornment bf. Program G is not range-restricted, but it can be made range-restricted by unfolding the usage of rule $G2$. However, the initial values in query Qd cannot restrict the first argument of h before doing the recursion. □

Lemma 5.2 *For every left-linear program, there exists an equivalent right-linear program.* □

Proof: Let program S be left-linear for adornment β. Let $G(S,\beta) = \{U \rightarrow Ul_1 \mid Ul_2 \mid \ldots \mid e_1 \mid e_2 \mid \ldots\}$ be the associated left-linear grammar. Then, G is equivalent to the right-linear grammar $G^r(S,\beta) = \{U \rightarrow e_1H \mid e_2H \mid \ldots; H \rightarrow \epsilon \mid l_1H \mid l_2H \mid \ldots\}$. The chain program associated with G^r can be derived by (1) Introducing a predicate h of arity twice the number of bound arguments in adornment β, corresponding to the grammar symbol H, and (2) Writing a chain rule for each production rule in grammar G^r. Let program $\mathcal{R}(\mathcal{S})$ be the transformation of S obtained as above. Predicate h in program $\mathcal{R}(\mathcal{S})$ is right-linear for adornment β. □

It is not usually profitable to convert left-linear programs into right-linear as then the selections can no longer be commuted with the fix-point operation.

5.3 Right-Linear ⇔ Left-Linear

The left-linear grammar $\{M \rightarrow \epsilon \mid Mf_1 \mid Mf_2 \mid \ldots\}$, is equivalent to the right-linear grammar $\{M \rightarrow \epsilon \mid f_1M \mid f_2M \mid \ldots\}$. A program S for predicate u that is right-linear for adornment α, can be represented by a grammar of the form $\{U \rightarrow Me_1 \mid Me_2 \mid \ldots\}$, where M is of the form given above (See proof for Lemma 5.1). Similarly, if program S for predicate u is left-linear for adornment β, it can be represented by a grammar of the form $\{U \rightarrow e_1M \mid e_2M \mid \ldots\}$, where M again is of the form given above (See proof for Lemma 5.2).

Consider the program for a predicate m corresponding to the grammar symbol M. If we use the right-linear grammar, the associated program Sr will be right-linear. If we use the left-linear grammar, the associated program Sl will be left-linear. We can show that $\mathcal{L}(Sr) = Sl$, $\mathcal{R}(Sl) = Sr$, $\mathcal{R}(\mathcal{L}(Sr)) = Sr$, and $\mathcal{L}(\mathcal{R}(Sl)) = Sl$.

We thus arrive at the duality between right- and left-linear programs.

Theorem 5.1 *Right-Linear and Left-Linear programs are duals of each other.* □

We can rewrite a right-linear program as a left-linear program. We can then transform the new left-linear predicate into right-linear, back into left-linear and so on, without introducing any new predicates or rules, recovering the original rules after every cycle.

6 Conclusions

We have improved on the work on right-linear programs of Naughton, Ramakrishnan, Sagiv and Ullman [NRSU89a], and the extensions to handle contexts in right-linear programs, proposed independently by Kemp, Ramamohanarao, and Somogyi [KRS90] and by Mumick and Pirahesh [MP91]. We have three important results:

1. Some right-linear queries have too many arguments that are bound, not all of which can be used to

restrict computation. Context right-linear (CRL) and right-linear (RL) transformations are inefficient for such *overbound* queries. We define overbound queries, and present techniques to optimize overbound queries.

2. Analytical comparison of the efficiency of context right-linear transformation with magic-sets shows that, in general, the two techniques are incomparable. However, for a very useful subclass of *stable right-linear* programs, CRL performs better than magic-sets.

3. Right-linear and left-linear programs are duals of each other.

The first result identifies a class of queries for which partial sips give better results than full sips. We are able to recognize that there are too many selection conditions, and that we must choose between traversing the graph from source or destination nodes. In case we decide to do the former, we have given algorithms that can perform better than the context right-linear and magic-sets transformations by an order of magnitude. To the best of our knowledge, *overbound right-linear* is the first class of programs for which an algorithm using partial sips (CORL) strongly dominates a similar algorithm using full sips (CRL). The idea of using partial sips to gain performance needs to be explored further. For example, the query ?- sg^{bb} is right-linear. It is not overbound right-linear, but could benefit from partial sips.

The second result lets us choose CRL over magic-sets for many of the common applications, with confidence that we are not selecting a bad evaluation strategy. We have defined a class of stable programs, and shown that for stable programs on graphs with bounded degree, CRL performs better than magic-sets. We have also learned that for unstable programs (Example 3.2), or for stable programs on graphs with unbounded degree (Example 3.3), CRL can be much worse than magic-sets, and so a choice between CRL and magic-sets depends on the query and the edb characteristics. Our analysis shows that fan-in and fan-out of the database graph are important characteristics. Intuitively, CRL is likely to be better than magic-sets if the fan-in of the database graph going from sources to destinations is low. However, experimental performance studies are needed for unbounded degree graphs and for unstable programs to better understand the effect of edb characteristics. The experimental study in Kemp *et. al.* [KRS90] is on a stable program for a bounded degree graph, and supports our conclusions derived from an analytical model.

The trade-offs between magic-sets and CRL seem to be applicable to a larger class of queries, such as summarized and indented explosion/implosion queries in bill-of-material applications. These queries involve aggregation, and a CRL style computation that computes the answers by traversing the graph once from sources to destinations is an alternative to magic-sets. The trade-offs involved are very similar to the ones we have discussed in this paper.

The third result tells us that for every right-linear program, there is an equivalent left-linear program, and for every left-linear program, there is an equivalent right-linear program. In fact, we can view the context right-linear transformation as the conversion of a right-linear program into a left-linear program, followed by a simple predicate pushdown into a nonrecursive rule.

Closely related to right-linear programs and the right-linear transformation are the multi-linear and mixed-linear programs and transformations defined by Naughton *et. al.* [NRSU89a]. Kemp *et. al.* [KRS90] define context multi-linear and context mixed-linear transformations. They claim that their context mixed-linear transformation works on *pseudo left-linear* rules. The following example contradicts their claim.

EXAMPLE 6.1 (Context Mixed-linear for pseudo left-linear rules)
Consider the following program S:

$(S1)$: $p(X,Y)$:- $b(X,Y)$.
$(S2)$: $p(X,Y)$:- $p(X,Z)$ & $c(Z,Y,X)$.
$(S3)$: $p(X,Y)$:- $a(X,Z)$ & $p(Z,Y)$.

(Q): ?- $init(U)$ & $p(U,Y)$.

Rule $S2$ is pseudo left-linear and rule $S3$ is right-linear, so that program S is mixed-linear. Kemp *et. al.* [KRS90] propose doing the following transformation:

(C_S1): $p(U,Y)$:- $cm\_p(U,X)$ & $b(X,Y)$.
(C_S2): $p(U,Y)$:- $p(U,Z)$ & $c(Z,Y,U)$.

(C_S3): $cm\_p(U,U)$:- $init(U)$.
(C_S4): $cm\_p(U,Y)$:- $cm\_P(U,X)$ & $a(X,Z)$.

Given $a = \{24\}, b = \{48\}, c = \{894\}, init = \{2,4\}$, program S computes $p = \{28, 29, 48, 49\}$, while program C_S computes $p = \{28, 48, 49\}$. The context mixed-linear transformation is therefore incorrect. □

The ideas of stability and overboundness can be applied to multi-linear and mixed-linear programs.

Naughton *et. al.* ([NRSU89b]) view the right-linear transformation as the result of factoring the answer predicate in the magic-sets transformation. CRL cannot be obtained by factoring. However, $CORL$ can be viewed as the result of an operation similar to factoring the context magic predicate in the CRL program. Ross [Ros91] presents tail recursion elimination as a generalization of the context right-linear transformation.

7 Acknowledgements

We thank Shuky Sagiv and Håkan Jakobsson for many insightful discussions on cost analysis. Raghu Ramakrishnan and Jeff Ullman provided valuable comments. The Starburst project at IBM Almaden Research Center and the *NAIL!* project at Stanford University provided a stimulating environment for this work.

References

[AU79] Alfred Aho and Jeffrey D. Ullman. Universality of data retrieval languages. In *Proceedings of the Sixth Symposium on Principles of Programming Languages (POPL)*, pages 110–120, January 1979.

[BR88] Francois Bancilhon and Raghu Ramakrishnan. Performance evaluation of data intensive logic programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 439–518, Washington D.C., 1988. Morgan Kaufmann.

[HCL+90] L.M. Haas, W. Chang, G.M. Lohman, J. McPherson, P.F. Wilms, G. Lapis, B. Lindsay, H. Pirahesh, M. Carey, and E. Shekita. Starburst mid-flight: As the dust clears. *IEEE Transactions on Knowledge and Data Engineering*, pages 143–160, March 1990. Also available as IBM Research Report RJ7278, San Jose, CA, Jan. 1990.

[Jak91] Håkan Jakobsson. Mixed-approach algorithms for transitive closure. In *Proceedings of the Tenth Symposium on Principles of Database Systems (PODS)*, Denver, CO, May 29-31 1991. ACM SIGACT-SIGMOD-SIGART.

[KRS90] David B. Kemp, Kotagiri Ramamohanarao, and Zoltan Somogyi. Right-, left- and multi-linear rule transformations that maintain context information. In *Proceedings of the Sixteenth International Conference on Very Large Databases (VLDB)*, Brisbane, Australia, August 13-16 1990.

[MFPR90] Inderpal Singh Mumick, Sheldon J. Finkelstein, Hamid Pirahesh, and Raghu Ramakrishnan. Magic is relevant. In *Proceedings of ACM SIGMOD 1990 International Conference on Management of Data*, pages 247–258, Atlantic City, NJ, May 23-25 1990.

[MNS+87] Katherine A. Morris, Jeffrey F. Naughton, Yatin Saraiya, Jeffrey D. Ullman, and Allen Van Gelder. Yawn! (yet another window on nail!). In *Data Engineering 10:4*, pages 28–43, 1987.

[MP91] Inderpal Singh Mumick and Hamid Pirahesh. Extending the right-linear transformation. Research report, RJ 7938, IBM Research Division, Computer Science, Almaden Research Center, San Jose, California 95120-6099, January 22 1991.

[NRSU89a] Jeffrey F. Naughton, Raghu Ramakrishnan, Yehoshua Sagiv, and Jeffrey D. Ullman. Efficient evaluation of right-, left-, and multi-linear rules. In *Proceedings of ACM SIGMOD 1989 International Conference on Management of Data*, Portland, OR, 1989.

[NRSU89b] Jeffrey F. Naughton, Raghu Ramakrishnan, Yehoshua Sagiv, and Jeffrey D. Ullman. Arity reduction through factoring. In *Proceedings of the Fifteenth International Conference on Very Large Databases (VLDB)*, Amsterdam, Holland, 1989.

[Ros91] Kenneth A. Ross. Modular acyclicity and tail recursion in logic programs. In *Proceedings of the Tenth Symposium on Principles of Database Systems (PODS)*, Denver, CO, May 29-31 1991. ACM SIGACT-SIGMOD-SIGART.

[Ull89] Jeffrey D. Ullman. `Principles of Database and Knowledge-Base Systems, Volumes 1 and 2.` Computer Science Press, 1988-89.

[UVG88] Jeffrey D. Ullman and Allen Van Gelder. Parallel complexity of logical query programs. *Algorithmica*, 3:5–42, January 1988.

[Yan90] Mihalis Yannakakis. Graph-theoretic methods in database theory. In *Proceedings of the Ninth Symposium on Principles of Database Systems (PODS)*, pages 230–242, Nashville, TN, April 2-4 1990. ACM SIGACT-SIGMOD-SIGART.