Research Report

Extending the Right-Linear Transformation

Inderpal Singh Mumick

Computer Science Department
Stanford University
Stanford, CA 94305.

Hamid Pirahesh

IBM Research Division
Almaden Research Center
San Jose, California 95120-6099.

Research Division
Yorktown Heights, New York • San Jose, California • Zurich, Switzerland

Extending the Right-Linear Transformation

Inderpal Singh Mumick

Computer Science Department
Stanford University
Stanford, CA 94305.

Hamid Pirahesh

IBM Research Division
Almaden Research Center
San Jose, California 95120-6099.

ABSTRACT: We present an optimization technique for a useful subclass of recursive queries in databases. We concentrate on right-linear queries, a class that includes the transitive closure and bill-of-material queries. When such a query is to be solved for a single initial value, a special optimization technique, the *right-linear transformation* has been proposed by Naughton *et al* [NRSU89a] as an efficient alternative to the magic-sets transformation. We present an *Extended Right-Linear transformation* that is applicable when the query contains multiple initial values, and is more efficient than magic-sets in many cases. The same idea can be used to extend the left-linear transformation.

We observe that right-linear and left-linear programs are duals of each other. Every right-linear program can be written as an equivalent left-linear program, and vice versa. The extended right-linear transformation can be viewed as a translation of a right-linear program into a left-linear program.

1. Introduction

Recursive queries are becoming common as DBMS's are being used increasingly for new applications such as CAD, CAM, CASE, office systems, engineering systems, and biological systems. Extensions of SQL to include recursion [ISO89] is a further evidence of significance of recursion. Databases used for such applications could be very large, and optimization is a key factor in making processing of such queries possible. Lack of good optimization can lead to many orders of magnitude degradation in response time and computing resource consumption, and unacceptable processing of queries.

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.

Recursive queries are of two types: Linear and Nonlinear [BR88]. While magic-sets is a good optimization for general recursive queries, special optimization techniques [Nau87, Nau88, Agr87, NRSU89a, NRSU89b] often do better than magic-sets on certain classes of linear recursive queries, 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, and showed that RL *dominates*[1] magic-sets on all programs, due to reduction in the arity of the recursive predicate.

The RL transformation has a problem. It only applies when there is one initial value in the query. If the query has multiple initial values, and we want to find out the matching tuples for each of the initial values separately, RL cannot be applied. While right-linear queries appear to be very common in database applications, we believe that they will usually have multiple initial values. Let us look at a few linear queries commonly used in bill-of-material applications [COP85] to justify this claim.

Let's assume we have an edb or extensional database (the stored relations) predicate $par(X, Y)$, defining parent of X is Y. (Usually there is an index on both arguments of this edb, allowing efficient traversal from parent to child and vice versa.) This edb predicate defines the part structure, and is usually very large, containing different versions of part structure information. A right-linear program can be written to produce an idb or intensional database (derived relations, or view) predicate $anc(X, Y)$, defining ancestor of X is Y. The idb anc gives for each part all other parts that are components of this part, directly or indirectly. Queries over anc usually have some restrictions on the ancestors or children. One rarely does a full retrieval on such a large database. Hence, it is extremely important to take advantage of such restrictions for optimization. Two kinds of queries are typically run over this database: *long* and *short*. Restrictions associated with long queries are usually much weaker than for short queries. For instance, in a long query we may want the part structure for all the parts currently being manufactured. This typically results in an extensive search of the database. The results may be aggregated (e.g. for pricing), or even printed. Short queries typically involve search of a small portion of the database, hence they usually have strong restrictions on the ancestors or children. We may want the part structure for a specific set of parts,

[1] 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.

or sometimes for a single part. All long queries, and most of the short queries, thus involve multiple initial values for which the part structure is desired.

In the first part of the paper, we extend the right-linear transformation of [NRSU89a] to the case where a query has multiple initial values. We call our transformation *Extended Right-Linear Transformation* (ERL for short). Efficiency of ERL is discussed in [**?**]. We show that the left-linear transformation presented in [NRSU89a] can also be extended similarly.

In the second part of the paper, we look upon ERL 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 preliminary notation and definitions in Section 1.1, we present our extension to the right-linear transformation in Section 2. Then, in Section 3 we prove that right-linear and left-linear programs are duals of each other. Section 4 provides a comparison with related work.

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. Built-in predicates can be introduced to enforce this constraint. We assume the reader is familiar with adornments, sips, magic-sets, conjunctive query containment and other concepts covered in [Ull88, Ull89].

Given an edb relation $par(X,Y)$, defining parent of X is Y, the following recursive program P[2] produces an idb relation $anc(X,Y)$, defining ancestor of X is Y.

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

We use subscripts to distinguish between the two occurrences of par. In the rest of this paper, we treat par_1 and par_2 as two distinct edb's. Assume that each argument of par_1 and par_2 can have n possible values. Queries over anc usually have some restrictions on the ancestors, children, or both. Restricting children in query Qa:

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

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 restriction on ancestors would lead to the adorned goal anc^{fb}, while a restriction on both children and ancestors would lead to the adorned goal anc^{bb}.

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 for adornment bf.

[2]We use a letter, such as P, to represent a program comprised of rules $P1, P2, \ldots$

Definition 1.1. 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): $u^\alpha(X_1, \ldots, X_k, Y_1, \ldots, Y_l) :- first(X_1, \ldots, X_k, Z_1, \ldots, Z_k)\ \&\ u^\alpha(Z_1, \ldots, Z_k, Y_1, \ldots, Y_l)$.

and each nonrecursive exit rule can be written as:

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

where

- $first$ 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 $first$. As an example, in rule $P2$ for adornment anc^{bf}, $first(X_1, Z_1) = par_1(X_1, Z_1)$. For adornment anc^{bb}, $first(X_1, X_2, Z_1, Z_2) = par_2(X_1, Z_1)\ \&\ Z_2 = X_2$. $Z_2 = X_2$ may both be dropped from $first$.
- Each Y variable appears exactly twice in the rule, once in the head and once in the recursive subgoal, in exactly the same argument position.

□

EXAMPLE 1.1. (Right-Linear Programs)
Program P is right-linear for adornments bf and bb. For adornment bf, the single recursive rule $P2$ can be written as:

$(P2')$: $anc^{bf}(X, Y) :- par_2(X, Z)\ \&\ anc^{bf}(Z, Y)$.

For adornment bb, rule $P2$ would be written as:

$(P2'')$: $anc^{bb}(X_1, X_2) :- par_2(X_1, Z_1)\ \&\ Z_2 = X_2\ \&\ anc^{bb}(Z_1, Z_2)$.

If a program is right-linear for an adornment α (say bf), then it is guaranteed to be right-linear for any adornment more bound than α (such as bb). □

[NRSU89a] also defines left-linear programs. Informally, a rule is left-linear if the head adornment can be preserved by placing the recursive subgoal *leftmost*. For example, in rule $P2$, if we adorn the head fb, and place the recursive anc subgoal *leftmost* in the body, it will be adorned fb. Rule $P2$ is therefore said to be left-linear for adornment fb.

EXAMPLE 1.2. (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).

$(Ma1)$: $anc^{bf}(X, Y) :- m\_anc^{bf}(X)\ \&\ par_1(X, Y)$.
$(Ma2)$: $anc^{bf}(X, Y) :- m\_anc^{bf}(X)\ \&\ par_2(X, Z)\ \&\ anc^{bf}(Z, Y)$.

$(Ma3)$: $m\_anc^{bf}(X) :- init(X)$.
$(Ma4)$: $m\_anc^{bf}(Z) :- m\_anc^{bf}(X)\ \&\ par_2(X, Z)$.

Given t initial values, program Ma could compute $O(n)$ magic tuples, and $O(n^2)$ anc tuples. □

Definition 1.2. 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 1.1 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 1.1 is transformed into:

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

□

2. Extended Right-Linear (ERL) Transformation

2.1. Motivation

Consider the query Qa on the ancestor program P. The magic-sets optimization produces the program Ma (Example 1.2). Program Ma has two recursions. The first recursion (rules $Ma3$ and $Ma4$) computes the ancestors of initial X values into the magic predicate, m_anc^{bf}. The second recursion (rules $Ma1$ and $Ma2$) then computes the ancestors of all magic values in m_anc^{bf}. But we are only interested in the ancestors of initial values, not in ancestors of the *ancestors of the initial values (m_anc^{bf})*. Program Ma thus generates many redundant anc^{bf} tuples.

If there is only one initial value in the query, the RL transformation can be used with the following benefits:

1. RL eliminates the second recursion, and
2. RL does not compute redundant anc^{bf} tuples.

EXAMPLE 2.1. (Right-linear transformation)
Naughton, Ramakrishnan, Sagiv, and Ullman [NRSU89a] observed that if there is only one initial value (say 5), then rule $Ma1$ in the magic program Ma generates all the ancestors of 5, but does not link the ancestors to 5. The recursive rule $Ma2$ is responsible for linking the ancestors of 5 to 5, but to do so, it must first compute ancestors of ancestors of 5. They define a right-linear (RL) transformation that rewrites program P for query Qb as follows:

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

$(Rb1)$: $anc^{bf}(5,Y)$:- $m\_anc^{bf}(X)$ & $par_1(X,Y)$.

$(Rb2)$: $m\_anc^{bf}(5)$.
$(Rb3)$: $m\_anc^{bf}(Z)$:- $m\_anc^{bf}(X)$ & $par_2(X,Z)$.

Rule $Rb1$ is similar to rule $Ma1$, except that all the ancestors of 5 (the Y values) are directly linked to 5. The second recursion is eliminated, and ancestor tuples for ancestors of 5 are never computed, potentially reducing the number of computed tuples by $O(n)$. □

Definition 2.1. 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 1.1 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 1.1 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 $R3$. □

However, if we are given multiple initial nodes and we want to determine the (X, Y) pairs where X is an initial node, and Y is reachable from X, RL cannot be used. If we simply place all the initial node in the magic set, RL will derive all the nodes reachable from any of the initial nodes, but cannot link the reachable nodes to the correct initial node. The extended right-linear transformation proposed below will solve this problem.

2.2. Extension for multiple initial values

EXAMPLE 2.2. (Extended right-linear transformation)
Consider query Qa on the ancestor program P. After either the magic-sets or the RL transformation, the magic set m_anc^{bf} contains ancestors of all the initial values. However, there is no indication whether a magic value z is derived from an initial value x_1 or an initial value x_2. Rules $Ma1$ or $Rb1$ then generate all the answer values Y that are ancestors of some initial value, but cannot pinpoint the exact initial value to which each Y should be linked.

The problem can be solved if we keep track of the initial value from which each magic tuple is derived. ERL generates the program Ea:

$(Ea1)$: $anc^{bf}(U, Y)$:- $wm\_anc^{bf}(U, X)$ & $par_1(X, Y)$.

$(Ea2)$: $wm\_anc^{bf}(X, X)$:- $init(X)$.
$(Ea3)$: $wm\_anc^{bf}(U, Z)$:- $wm\_anc^{bf}(U, X)$ & $par_2(X, Z)$.

We define a widened magic predicate $wm\_anc^{bf}(U, X)$ that stores the initial U value from which the each magic X value is derived. Now, when we apply the exit rule $Ea1$, we can directly link each answer Y value to the correct initial U value since U is available in the widened magic tuple.

Program Ea has only one recursion, and does not generate any redundant anc^{bf} facts. □

The widened magic set wm_anc^{bf} is more expensive to compute than the magic set m_anc^{bf}, since the former is a binary predicate, and the later is a unary predicate. For example, given t initial values, Program Ea could compute $O(tn)$ widened magic tuples, while program Ma could compute $O(n)$ magic tuples. However, the extra cost in computing the widened magic set can be compensated by the reduced work in computing anc^{bf}. Program Ea is thus an attractive optimization for query Qa. [**?**] discusses performance characteristics of ERL.

Definition 2.2. Extended Right-Linear Transformation — ERL: The ERL 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:

  $(E1)$: $wm\_u^\alpha(U_1, \ldots, U_k, U_1, \ldots, U_k)$:- $init(U_1, \ldots, U_k)$.

- Every right-linear rule of the form gr in Definition 1.1 is transformed into:

$$(E2):\quad wm\_u^{\alpha}(U_1,\ldots,U_k,Z_1,\ldots,Z_k) \text{ :- } wm\_u^{\alpha}(U_1,\ldots,U_k,X_1,\ldots,X_k)\ \&\ first(X_1,\ldots,X_k,Z_1,\ldots,Z_k).$$

- Every exit rule of the form ge in Definition 1.1 is transformed into:

$$(E3):\quad u^{\alpha}(U_1,\ldots,U_k,Y_1,\ldots,Y_l) \text{ :- } wm\_u^{\alpha}(U_1,\ldots,U_k,X_1,\ldots,X_k)\ \&\ exit(X_1,\ldots,X_k,Y_1,\ldots,Y_l).$$

□

wm_u^{α} is the widened magic predicate, with the following interpretation: $wm\_u^{\alpha}(\overline{U},\overline{X})$ holds if $\overline{X}$ is a magic value by virtue of $\overline{U}$ appearing in the initial set. When computing the answers through rule $E3$, we can now link each answer $\overline{Y}$ value to the correct initial $\overline{U}$ value, since $\overline{U}$ is available in the widened magic tuple. Thus, only the u^{α} tuples required in the answer are computed by program E.

Theorem 2.1. *Given a program S that is right-linear for the query u^{α}, let E be the program obtained by the extended right-linear transformation. Then S is equivalent to E for the query u^{α}.*
□

EXAMPLE 2.3. Consider program P for adornment bb in query Qc:

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

Doing the extended right-linear transformation, we get program Ec:

$(Ec1)$: $anc^{bb}(U_1,U_2) \text{ :- } wm\_anc^{bb}(U_1,U_2,X_1,X_2)\ \&\ par_1(X_1,X_2)$.

$(Ec2)$: $wm\_anc^{bb}(U_1,U_2,U_1,U_2) \text{ :- } init(U_1,U_2)$.
$(Ec3)$: $wm\_anc^{bb}(U_1,U_2,Z_1,Z_2) \text{ :- } wm\_anc^{bb}(U_1,U_2,X_1,X_2)\ \&\ par_2(X_1,Z_1)\ \&\ Z_2 = X_2$.
□

2.3. Extended Left-Linear Transformation (ELL)

The left-linear transformation of [NRSU89a] can be extended for multiple initial values in a manner similar to ERL.

EXAMPLE 2.4. (ELL)
Consider query Qd on the ancestor program P:

(Qd): ?- $init(Y)\ \&\ anc^{fb}(X,Y)$.

Program P is left-linear with respect to adornment fb, and can be transformed to:

$(F1)$: $anc^{fb}(U,X) \text{ :- } wm\_anc^{fb}(U,Y)\ \&\ par_1(X,Y)$.
$(F2)$: $anc^{fb}(X,Y) \text{ :- } anc^{fb}(Z,Y)\ \&\ par_2(X,Z)$.

$(F3)$: $wm\_anc^{fb}(Y,Y) \text{ :- } init(Y)$.

The widened magic predicate consists only of the initial values, and can be reduced to the monadic magic predicate. □

ELL simply commutes the selection and least fixed point operators [AU79]. ELL is very similar to the magic-sets transformation — the only difference is that magic-sets introduces a redundant join with m_anc^{bf} in rule $F2$.

3. 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 ERL 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].

The following result follows from Lemma 3.1 and Lemma 3.2 in Sections 3.1 and 3.2.

Theorem 3.1. *Right-Linear and Left-Linear programs are duals of each other.* □

3.1. Right-Linear ⇒ Left-Linear

EXAMPLE 3.1. Consider the ancestor program P for the adornment bf. 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,[3] 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 Ea obtained by the ERL transformation (predicate $m \equiv$ predicate wm_anc^{bf}). □

EXAMPLE 3.2. 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 $< par_2(X_1, Z_1)$ & $Z_2 = X_2 >$.

[3]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.

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 Ec obtained by the ERL transformation (predicate $m \equiv$ predicate wm_anc^{bb}). □

Lemma 3.1. *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 1.1. $G(S, \alpha) = \{U \rightarrow f_1U \mid f_2U \mid \ldots \mid e_1 \mid e_2 \mid \ldots\}$ is the associated right-linear grammar. G is equivalent to the 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 ERL transformation (Definition 2.2), except for the initial rule for the widened magic predicate, m, which now looks like:

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

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.

3.2. Left-Linear ⇒ Right-Linear

EXAMPLE 3.3. 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.

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

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

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 $G^r(R, bf) = \{A \rightarrow q_1 H; H \rightarrow \epsilon \mid q_2 H\}$. The right-linear program corresponding to G^r is:

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

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

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. $\square$

Lemma 3.2. *For every left-linear program, there exists an equivalent right-linear program.* $\square$

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 $G^r(S, \beta) = \{U \rightarrow e_1 H \mid e_2 H \mid \ldots; H \rightarrow \epsilon \mid l_1 H \mid l_2 H \mid \ldots\}$. 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. $\square$

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.

3.3. Right-Linear $\Leftrightarrow$ Left-Linear

The left-linear grammar $\{M \rightarrow \epsilon \mid Mf_1 \mid Mf_2 \mid \ldots\}$, is equivalent to the right-linear grammar $\{M \rightarrow \epsilon \mid f_1 M \mid f_2 M \mid \ldots\}$. 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 3.1). Similarly, if program S for predicate u is left-linear for adornment β, it can be represented by a grammar of the form $\{U \rightarrow e_1 M \mid e_2 M \mid \ldots\}$, where M again is of the form given above (See proof for Lemma 3.2).

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. 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.

4. Related Work

Naughton, Ramakrishnan, Sagiv and Ullman defined right-, left- and mixed-linear programs in [NRSU89a] and gave program transformations when the query has only one initial value. In

[NRSU89b] they looked at these transformations as a special case of factoring predicates into two sets of arguments.

A recent paper by Kemp, Ramamohanarao and Somogyi [KRS90] presents work done concurrently with ours. They define context right-linear and context left-linear transformations that are identical to our extended right-linear and extended left-linear transformations. Context multi-linear and mixed-linear transformations are also defined in [KRS90]. Mumick and Pirahesh [?] point out an error in their definition of mixed-linear programs.

The performance of ERL has been compared with magic-sets both experimentally [KRS90] and analytically [?]. The latter defines a useful subclass of right-linear programs for which ERL dominates magic-sets.

The study of the duality between right-linear and left-linear programs is novel to our work.

5. Conclusions

We have developed on the work on right-linear programs of Naughton, Ramakrishnan, Sagiv and Ullman [NRSU89a]. We have two important results:

1. The *extended right-linear transformation* algorithm for right-linear programs in the presence of multiple initial values in the query.
2. Duality of right-linear and left-linear programs.

The first result enables us to take advantage of right-linear programs even when the query has multiple initial values, which is expected to be the common case.

The second result is of both theoretical and practical interest, telling us that selections on right-linear programs can always be applied before starting the recursion by first transforming the program into a left-linear one.

6. Acknowledgements

The Starburst project at IBM Almaden Research Center and the *NAIL!* project at Stanford University provided a stimulating environment for this work.

References.

[Agr87] Rakesh Agrawal. Alpha: An extension of relational algebra to express a class of recursive queries. In *Proceedings of the Third IEEE International Conference on Data Engineering*, pages 580–590, February 1987.

[AU79] Alfred Aho and Jeffrey D. Ullman. Universality of data retrieval languages. In *Proceedings of the Sixth Symposium on Principles of Programming Languiages (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.

[COP85] IBM. *COPICS: Bill of Material/online and batch utilities II; User's Guide and Reference Manual*, December 1985. Number: IBM SH31-0040.

[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.

[ISO89] ISO_ANSI. Iso-ansi working draft: Database language sql2 and sql3; x3h2; iso/iec jtc1/sc21/wg3, 1989.

[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.

[MP91] Inderpal Singh Mumick and Hamid Pirahesh. Overbound and Right-linear queries. In *Proceedings of the Tenth Symposium on Principles of Database Systems (PODS)*, Denver, CO, May 29-31 1991. ACM SIGACT-SIGMOD-SIGART.

[Nau87] Jeffrey F. Naughton. One-sided recursions. In *Proceedings of the Sixth Symposium on Principles of Database Systems (PODS)*, pages 340–348, San Diego, CA, 1987.

[Nau88] Jeffrey F. Naughton. Compiling separable recursions. In *Proceedings of ACM SIGMOD 1988 International Conference on Management of Data*, pages 312–319, Chicago, IL, 1988.

[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.

[Ull88] Jeffrey D. Ullman. `Principles of Database and Knowledge-Base Systems, Volume 1.` Computer Science Press, 1988.

[Ull89] Jeffrey D. Ullman. `Principles of Database and Knowledge-Base Systems, Volume 2.` Computer Science Press, 1989.

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