On the Relationship between Chain Queries and Linear Datalog Programs

29 March 1997

Foto Afrati
National Technical University of Athens
Department of Electrical and Computing Engineering
Division of Computer Science
15773 Zographou, Athens, Greece
afrati@softlab.ece.ntua.gr

Francesca Toni
Imperial College of Science, Technology and Medicine
Department of Computing
180 Queen's Gate, London SW7 2BZ, UK
ft@doc.ic.ac.uk

1 Introduction

First-order (algebraic) query languages lack recursion and, as a consequence, have limited expressive power. Datalog, the language of Horn logic without function symbols, embeds recursion and therefore allows to express a far wider class of queries. However, queries expressed in Datalog are harder to evaluate than classical first-order queries. As a consequnce, in the (deductive) database community, many efforts have been devoted to detect special classes of Datalog programs for which efficient evaluation methods and optimisation techniques exist.

Linear Programs are special Datalog programs which belong to NC^2. [1] Since it is conjectured that $NC^2 \subset P$ (in the same way that it is conjectured that $P \subset NP$), linear programs are simpler to evaluate than generic Datalog programs, belonging to P.

We study the relationship between linear programs and a special class of (recursive) queries, referred to in the literature as *chain queries*. It is known that there exist chain queries which do not correspond to any linear program [1]. Our research aims to establish linearisability of chain queries of special kinds. We identify two classes of linearisable chain queries, i.e. regular and *pseudo-regular* chain queries. To the best of our knowledge, the class of pseudo-regular chain queries has not been studied elsewhere in the literature.

[1]The complexity class NC^2 consists of all problems solvable in time bounded by $log^2|X|$, with X the size of the problems, using only a polynomial number of parallel machines.

2 Preliminaries

A *relational* (or *extensional*) *database* is a tuple $(D, r_1, \ldots, r_n)$ with D a finite set (the domain) and r_i, $i = 1, \ldots, n$, a relation over D. In logic programming terms, an extensional database is a (finite) set of unit clauses without function symbols. Under this formulation, the (implicit) domain is the Herbrand Universe of the set of unit clauses. The predicates occurring in an extensional database are referred to as *extensional predicates*.

A *query* over an extensional database is a set of relations over the domain of the database.

A *Datalog program* (or *intensional database*) is a set of Horn clauses without function symbols. The predicates occuring in the positive literals (heads) of the clauses in a Datalog program are referred to as *intensional* predicates.

Given an extensional database EDB and a Datalog program IDB, the *query corresponding to* an intensional predicate p in IDB is:
$Q_p^{IDB}(EDB) = \{d | p(d)$ belongs to the least Herbrand model of $EDB \cup IDB\}$ [2].

Example 2.1 Let $EDB = \{arc(a,b), arc(b,c), arc(c,d), arc(a,e)\}$ (with implicit domain $\{a,b,c,d,e\}$) and let IDB be
$path(X,Y) \leftarrow arc(X,Y)$
$path(X,Y) \leftarrow arc(X,Z) \wedge path(Z,Y)$
Then, the query corresponding to the (intentional) predicate $path$ is:
$Q_{path}^{IDB}(EDB) = \{(a,b), (b,c), (e,d), (a,e), (a,c), (a,d), (b,d)\}$.

A *linear program* is a Datalog program with at most one intentional predicate amongst the negative literals (body) of each clause. Note that the program IDB in example 2.1 is linear.

A *linearisable program* IDB is a Datalog program such that there exists a linear program IDB' satisfying the property that, for every extensional database EDB, the least Herbrand model of $EDB \cup IDB$ coincides with the least Herbrand model of $EDB \cup IDB'$ restricted to the predicates occuring in $EDB \cup IDB$. A *linearisable query* is a query corresponding to a linearisable program.

The class of "piecewise linear programs", defined below, is of special interest, since piecewise linear programs are linearisable [2]. In fact, piecewise linear programs can be transformed into linear program by applying standard program transformation techniques [4], which preserve the least Herbrand model semantics. A *piecewise linear program* is a Datalog program where in the body of each clause there is at most one atom whose predicate is "mutually recursive" with the predicate in the head of the clause, where a predicate p is *mutually recursive* with a predicate q if

[2] In general, d is a tuple of elements of the domain of EDB.

and only of p and q "depend on" each other, where p *depends on* q if and only of q or a predicate which depends on q occur in the body of a clause with p in the head.

In the remainder of this report we will restrict our attention to extensional databases with binary relations, referred to as *graphs*. Note that the extensional databases EDB in example 2.1 is a graph.

Let EDB be a graph and Σ be the (finite) alphabet with a letter R_i for each relation r_i in EDB. Then, the *chain query* for a language $L \subseteq \Sigma^*$ is:

$Q_L(EDB) = \{(u,v) |$ there exists a path in EDB from u to v spelling a word in $L\}$,

where a *path from u to v spelling a word* $R_{i_1} \ldots R_{i_l} \in \Sigma^+$ is a sequence $u = u_1, \ldots, u_{l+1} = v$ of elements of the domain of EDB such that $r_{i_j}(u_j, u_{j+1}) \in EDB$, for each $j = 1, \ldots, l$, and a *path spelling the empty word*, $\mathcal{E}$, is a sequence u, u, for every element u of the domain of EDB.

Example 2.2 Let EDB as in example 2.1. Then $\Sigma = \{Arc\}$. Let $L = \{Arc^i | i \geq 1\}$. Then $Q_L(EDB) = \{(a,b),(b,c),(e,d),(a,e),(a,c),(a,d),(b,d)\}$ ($= Q^{IDB}_{path}$ in example 2.1). The spelled words are Arc, Arc, Arc, Arc, Arc^2, Arc^3, Arc^2, respectively.

Example 2.3 Let $EDB = \{r(a,b), r(b,c), s(c,d)\}$ (with implicit domain $\{a,b,c,d\}$). Then, $\Sigma = \{R, S\}$. Let $L = \{\ R^i S^j | i,j \geq 0, i+j > 0\ \}$. Then, $Q_L(EDB) = \{(a,b),(a,c),(a,d),(b,c),(b,d),(c,d)\}$. The spelled words are R, RR, RRS, R, RS, S, respectively.

3 Linearisable Chain Queries

We consider chain queries for regular, context-free and "pseudo-regular" languages (the latter are defined below in section 3.3), and analyse their linearisability. Whereas it is known that chain queries for context-free languages are not linearisable [1], we prove that chain queries for regular and pseudo-regular languages are linearisable.

3.1 Regular languages

Regular languages are generated by grammars with production rules of the form:

$I \rightarrow RJ \,|\, R$ or $I \rightarrow JR \,|\, R$

where I, J are non-terminal symbols and R is a terminal symbol. The terminal symbols are elements of Σ, the (finite) alphabet for the language $L(G)$ generated by the grammar G.

The *Datalog program $IDB(G)$ corresponding to a regular grammar G* is constructed as follows:

- each (terminal or non-terminal) symbol is mapped onto a binary (extensional or intensional, respectively) predicate;
- let (non-terminal) symbols I, J be mapped onto the intensional predicates i, j and (terminal) symbol R be mapped onto the extensional predicate r; then each production rule $I \to RJ$ is mapped onto a clause

  $i(X,Y) \leftarrow r(X,Z) \wedge j(Z,Y)$

  each production rule $I \to JR$ is mapped onto a clause

  $i(X,Y) \leftarrow j(X,Z) \wedge r(Z,Y)$

  and each production rule $I \to R$ is mapped onto a clause

  $i(X,Y) \leftarrow r(X,Y)$.

Then, it can be proved that the query for a regular language $L(G)$ coincides with the query corresponding to the predicate i in $IDB(G)$ on which the initial symbol I in G is mapped:

Theorem 3.1 Given a regular grammar G with initial symbol I, let I be mapped onto the intensional predicate i in $IDB(G)$. For every extensional database EDB for the extensional predicates in $IDB(G)$
$Q_{L(G)}(EDB) = Q_i^{IDB(G)}(EDB)$.

Proof: By definition, $Q_i^{IDB(G)}(EDB) = \{(u,v) | i(u,v)$ belongs to the least Herbrand model of $EDB \cup IDB(G)\}$. Since SLD resolution is complete with respect to the least Herbrand model semantics [3], $Q_i^{IDB(G)}(EDB) = \{(u,v) |$ there is an SLD refutation for $i(u,v)$ in $EDB \cup IDB(G)\}$.
It is easy to prove, by induction, that there is a one-to-one correspondence between derivation trees for words $R_{i_1} \dots R_{i_l}$ of $L(G)$ and SLD derivations, in $IDB(G)$ (and therefore in $EDB \cup IDB(G)$) from goals $\leftarrow i(X,Y)$ to goals $\leftarrow r_{i_1}(X,Z_1) \wedge \dots \wedge r_{i_l}(Z_{l-1},Y)$, for some distinct variables $Z_1, \dots, Z_{l-1}$. Indeed, the application of a production rule $I \to JR$ to a non-terminal symbol I corresponds to a step of resolution between the goal $\leftarrow \dots \wedge i(X,Y) \wedge \dots$ and the clause $i(X,Y) \leftarrow j(X,Z) \wedge r(Z,Y)$ (similarly for the other kinds of production rules).
Then, for any such word $R_{i_1} \dots R_{i_l}$ of $L(G)$, assume $r_{i_j}(u_j, u_{j+1}) \in EDB$, for $j = 1, \dots, l$, i.e. $(u_1, u_{l+1}) \in Q_{L(G)}(EDB)$. Then, trivially there is a refutation for $\leftarrow r_{i_1}(X,Z_1) \wedge \dots \wedge r_{i_l}(Z_{l-1},Y)$ in EDB (and therefore in $EDB \cup IDB(G)$) returning the substitution $\{X/u_1, Z_1/u_2, \dots Z_{l-1}/u_l, Y/u_{l+1}\}$, i.e. $(u_1, u_{l+1}) \in Q_i^{IDB(G)}(EDB)$.
Conversely, for any derivation from $\leftarrow i(X,Y)$ to $\leftarrow r_{i_1}(X,Z_1) \wedge \dots \wedge r_{i_l}(Z_{l-1},Y)$, assume there is a refutation for $\leftarrow r_{i_1}(X,Z_1) \wedge \dots \wedge r_{i_l}(Z_{l-1},Y)$ in EDB (and therefore in $EDB \cup IDB(G)$) returning the substitution $\{X/u_1, Z_1/u_2, \dots Z_{l-1}/u_l, Y/u_{l+1}\}$,

i.e. $(u_1, u_{l+1}) \in Q_i^{IDB(G)}(EDB)$. Then, trivially $r_{i_j}(u_j, u_{j+1}) \in EDB$, for $j = 1, \ldots, l$, i.e. $(u_1, u_{l+1}) \in Q_{L(G)}(EDB)$. *q.e.d*

Trivially, for every regular grammar G, $IDB(G)$ is linear and therefore linearisable. As a consequence, the chain queries for regular languages are linearisable.

3.2 Context-free languages

Chain queries for some context-free languages are linearisable:

Example 3.1 Let $L = \{R_1^i R_2^i R_3^j R_4^j | i, j \geq 0\} \subseteq \Sigma^*$ with $\Sigma = \{R_1, R_2, R_3, R_4\}$. L can be generated by the (context-free) grammar with production rules:
$I \to I_1 I_2$
$I_1 \to R_1 I_1 R_2 | \mathcal{E}$
$I_2 \to R_3 I_2 R_4 | \mathcal{E}$
I, I_1, I_2 non-terminal symbols and I initial symbol.
Assume R_i, $i = 1, \ldots, 4$, is mapped onto the extensional predicate r_i, and I, I_1, I_2 are mapped onto the intensional predicates i, i_1, i_2, respectively. It is easy to see (as for regular languages, theorem 3.1) that the chain query $Q_L(EDB)$, for any EDB for r_1, r_2, r_3, r_4, coincides with the query corresponding to the predicate i in the Datalog program:
$i(X, Y) \leftarrow i_1(X, Z), i_2(Z, Y)$
$i_1(X, Z) \leftarrow r_1(X, Y_1) \wedge i_1(Y_1, Y_2) \wedge r_2(Y_2, Z)$
$i_1(X, X)$
$i_2(Z, Y) \leftarrow r_2(Z, X_1) \wedge i_2(X_1, X_2) \wedge r_4(X_2, Y)$
$i_2(X, X)$
obtained as for regular languages. This program is not linear, but is piecewise linear and therefore linearisable (see section 2).

However, there are chain queries for context-free languages which are not linearisable, as, for example, the chain queries for the languages:
$\{W | W$ has the same number of occurences of R_1 and $R_2\}$ and
$L(G)$ generated by the grammar G with production rules:
$I \to I R_1 I (R_2 I R_1 I)^j | \mathcal{E}$, with $j \geq 1$.
Both languages, as well as the proof of their non-linearisability, are given in [1].

3.3 Pseudo-regular languages

We identify a class of languages, containing all regular languages, such that all chain queries for languages in such class are linearisable. This is the class of all *pseudo-regular languages*, of the form:

$$\{\alpha_1^{k_1} \dots \alpha_n^{k_n} | \text{ for each } j = 1, \dots, n, \text{ either } k_j \text{ is an index and } k_j \geq 0 \\ \text{or } k_j \text{ is a natural number (different from 0)}\}$$

with $\alpha_j \in \Sigma^+$, $j = 1, \dots, n$, and $n > 0$.

We will refer to the chain queries for such languages as *pseudo-regular (chain) queries.*

Note that every regular language is trivially pseudo-regular. However, context-free languages might not be pseudo-regular, e.g. the language

$\{W|W$ has the same number of occurences of R_1 and $R_2\}$

is context-free but not pseudo-regular. Finally, pseudo-regular languages might not be context-free, e.g. the language

$\{R_1^k R_2^k R_3^k | k \geq 0\}$

is pseudo-regular but not context-free.

To prove that pseudo-regular queries are linearisable, we construct the corresponding Datalog programs and show that they are linearisable. We first illustrate the construction by means of two examples.

Example 3.2 Let $L = \{R_1^{i_1} R_2^{i_2} R_3^{i_1} R_4^2 | i_1, i_2 \geq 0\}$ (with $\Sigma = \{R_1, R_2, R_3, R_4\}$). The Datalog program corresponding to the query for L is:

$i_0(X, Y) \leftarrow i_1(X, Z, W, U) \wedge i_2(Z, W) \wedge i_3(U, Y)$
$i_1(X, Z, W, Y) \leftarrow r_1(X, X_1) \wedge r_3(W, W_1) \wedge i_1(X_1, Z, W_1, Y)$
$i_1(X, X, Y, Y)$
$i_2(Z, W) \leftarrow r_2(Z, Z_1) \wedge i_2(Z_1, W)$
$i_2(Z, Z)$
$i_3(U, Y) \leftarrow r_4(U, V) \wedge r_4(V, Y)$

This program is not linear but is piecewise linear and therefore linearisable (see section 2).

Example 3.3 Let $L = \{(R_1 R_2 R_3)^{i_1} R_4^{i_2} R_5^{i_1} | i_1, i_2 \geq 0\}$ (with $\Sigma = \{R_1, R_2, R_3, R_4, R_5\}$). The Datalog program corresponding to the query for L is:

$i_0(X, Y) \leftarrow i_1(X, Z, W, Y) \wedge i_2(Z, W)$
$i_1(X, Z, W, Y) \leftarrow r_1(X, X_1) \wedge r_2(X_1, X_2) \wedge r_3(X_2, X_3) \wedge r_5(W, W_1) \wedge i_1(X_3, Z, W_1, Y)$
$i_1(X, X, Y, Y)$
$i_2(Z, W) \leftarrow r_4(Z, Z_1) \wedge i_2(Z_1, W)$
$i_2(Z, Z)$

This program is not linear but is piecewise linear and therefore linearisable (see section 2).

In general, the *Datalog program* $IDB(L)$ *corresponding to a pseudo-regular language*

$\{\alpha_1^{k_1}\dots\alpha_n^{k_n} |$ for each $j = 1,\dots,n$, either k_j is an index and $k_j \geq 0$
or k_j is a natural number (different from 0)$\}$

for some $n > 0$ and $\alpha_j \in \Sigma^+$, $j = 1,\dots,n$, is constructed as follows. Let:

- m be the number of integer and distinct indexes amongst $k_1,\dots,k_n$ (trivially, $m \leq n$), and, after renaming them (for ease of reference) as $i_1,\dots,i_m$, according to the order in which they appear, let
- a_j be the number of factors with (the integer or index) i_j as exponent, $j = 1,\dots,m$. [3]

Then, the Datalog program corresponding to L has $m+1$ intensional predicates, $i_0, i_1,\dots,i_m$, with arity 2, $2a_1,\dots,2a_m$, respectively.

In example 3.2, there are two distinct indexes, i_1 and i_2, and an integer exponent, 2, and thus four intensional predicates, i_0, i_1, i_2 and i_3. The predicate i_1 has arity 4, since there are two factors ($R_1^{i_1}$ and $R_3^{i_1}$) with the index i_1 as exponent. The predicate i_2 has arity 2, since there is only one factor ($R_2^{i_2}$) with the index i_2 as exponent. The predicate i_3, corresponding to the integer exponent, has ariety 2.

Similarly in example 3.3.

Moreover, for ease of reference,

- rename the bases of the a_j factors of exponent i_j, for every $j = 1,\dots,m$, as $\alpha_{j,1},\dots,\alpha_{j,a_j}$,
- assume each $\alpha_{j,e}$ ($j = 1,\dots,m$, $e = 1,\dots,a_j$) [4] be $R_{j,e,1},\dots,R_{j,e,i_{j,e}}$, for some $i_{j,e} \geq 1$ (given in the definition of L).

In example 3.2, $\alpha_{1,1} = \alpha_1 = R_1$ and $\alpha_{1,2} = \alpha_3 = R_3$ ((both corresponding to exponent i_1), $\alpha_{2,1} = \alpha_2 = R_2$ (corresponding to exponent i_2), and $\alpha_{3,1} = \alpha_4 = R_4$ (corresponding to integer exponent 2).

In example 3.3, $\alpha_{1,1} = \alpha_1 = R_1R_2R_3$ and $\alpha_{1,2} = \alpha_3 = R_5$ (both corresponding to exponent i_1), and $\alpha_{2,1} = \alpha_2 = R_4$ (corresponding to exponent i_2). Moreover, $R_{1,1,1} = R_1$, $R_{1,1,2} = R_2$, $R_{1,1,3} = R_3$ (for $\alpha_{1,1}$), $R_{1,2,1} = R_5$ (for $\alpha_{1,2}$), and $R_{2,1,1} = R_4$ (for $\alpha_{2,1}$).

Finally, let:

- $var_1(\alpha_i)$, $var_2(\alpha_i)$ be (distinct) variables associated to the factor with base α_i, for $i = 1,\dots,n$ (we use a functional representation of variables for ease of reference), such that, for $i = 1,\dots,n-1$, $var_2(\alpha_i) = var_1(\alpha_{i+1})$, and

[3] Note that, if i_j is an integer exponent, then $a_j = 1$.

[4] Note that, if i_j is an integer exponent, then $e = 1$.

- $vars_j = var_1(\alpha_{j,1}), var_2(\alpha_{j,1}), \ldots, var_1(\alpha_{j,a_j}), var_2(\alpha_{j,a_j})$, for $j = 1, \ldots, m$. [5]

In example 3.3, $var_1(\alpha_1) = X$, $var_2(\alpha_1) = Z = var_1(\alpha_2)$, $var_2(\alpha_2) = W = var_1(\alpha_3)$ and $var_2(\alpha_3) = Y$.
Then, $IDB(L)$ consists of a clause

$$
\begin{array}{l}
i_0(var_1(\alpha_1), var_2(\alpha_n)) \leftarrow \\
\quad i_1(vars_1) \wedge \ldots \wedge i_m(vars_m)
\end{array}
$$

of clauses (for $j = 1, \ldots, m$ and i_j index exponent):

$$
\begin{array}{l}
i_j(var_1(\alpha_{j,1}), var_2(\alpha_{j,1}), \ldots, var_1(\alpha_{j,a_j}), var_2(\alpha_{j,a_j})) \leftarrow \\
\quad var_1(\alpha_{j,1}) = var_2(\alpha_{j,1}) \wedge \ldots \wedge var_1(\alpha_{j,a_j}) = var_2(\alpha_{j,a_j}) \\
i_j(var_1(\alpha_{j,1}), var_2(\alpha_{j,1}), \ldots, var_1(\alpha_{j,a_j}), var_2(\alpha_{j,a_j})) \leftarrow \\
\quad r_{j,1,1}(var_1(\alpha_{j,1}), X_1^1) \wedge r_{j,1,2}(X_1^1, X_2^1) \wedge \ldots \wedge r_{j,1,i_{j,1}}(X_{i_{j,1}-1}^1, X_{i_{j,1}}^1) \wedge \\
\quad r_{j,2,1}(var_1(\alpha_{j,2}), X_1^2) \wedge r_{j,2,2}(X_1^2, X_2^2) \wedge \ldots \wedge r_{j,2,i_{j,2}}(X_{i_{j,2}-1}^2, X_{i_{j,2}}^2) \wedge \\
\quad \ldots \wedge \\
\quad r_{j,a_j,1}(var_1(\alpha_{j,a_j}), X_1^{a_j}) \wedge r_{j,a_j,2}(X_1^{a_j}, X_2^{a_j}) \wedge \ldots \wedge r_{j,a_j,i_{j,a_j}}(X_{i_{j,a_j}-1}^{a_j}, X_{i_{j,a_j}}^{a_j}) \wedge \\
\quad i_j(X_{i_{j,1}}^1, var_2(\alpha_{j,1}), \ldots, X_{i_{j,a_j}}^{a_j}, var_2(\alpha_{j,a_j}))
\end{array}
$$

where X_e^k are fresh, distinct variables, and each $r_{e,k,i}$ is an (extensional) predicate symbol corresponding to the letter $R_{e,k,i}$ [6], and of clauses (for $j = 1, \ldots, m$, i_j integer exponent and $i_j = s$):

$$
\begin{array}{l}
i_j(var_1(\alpha_{j,1}), var_2(\alpha_{j,1})) \leftarrow \\
\quad r_{j,1,1}(var_1(\alpha_{j,1}), X_1^1) \wedge r_{j,1,2}(X_1^1, X_2^1) \wedge \ldots \wedge r_{j,1,i_{j,1}}(X_{i_{j,1}-1}^1, X_{i_{j,1}}^1) \wedge \\
\quad \ldots \wedge \\
\quad r_{j,1,1}(X_{i_{j,1}}^{s-1}, X_1^s) \wedge r_{j,1,2}(X_1^s, X_2^s) \wedge \ldots \wedge r_{j,1,i_{j,1}}(X_{i_{j,1}-1}^s, var_2(\alpha_{j,1}))
\end{array}
$$

where X_e^k are fresh, distinct variables, and each $r_{e,k,i}$ is an (extensional) predicate symbol corresponding to the letter $R_{e,k,i}$ [7].
It can be proved that the query for a pseudo-regular language L coincides with the query corresponding to the predicate i_0 in $IDB(L)$:

Theorem 3.2 Given a pseudo-regular language L, for every extensional database EDB for the extensional predicates in $IDB(G)$
$Q_L(EDB) = Q_{i_0}^{IDB(L)}(EDB)$.

[5]Note that, if i_j is an integer exponent, then $vars_j = var_1(\alpha_{j,1}), var_2(\alpha_{j,1})$.
[6]In example 3.3, $X_1^1 = X_1$, $X_2^1 = X_2$, $X_3^1 = X_3$ and $X_1^2 = Z_1$.
[7]In example 3.2, $X_1^1 = V$.

Proof: By definition, $Q_{i_0}^{IDB(L)}(EDB) = \{(u,v)|i_0(u,v)$ belongs to the least Herbrand model of $EDB \cup IDB(L)\}$. Since SLD resolution is complete with respect to the least Herbrand model semantics [3], $Q_{i_0}^{IDB(L)}(EDB) = \{(u,v)|$ there is an SLD refutation for $i_0(u,v)$ in $EDB \cup IDB(L)\}$.

It is not difficult to see that there is a one-to-one correspondence between words $\alpha_1^{k_1} \dots \alpha_n^{k_n}$, for concrete values of $k_1, \dots, k_n$, of L and SLD derivations, in $IDB(L)$ (and therefore in $EDB \cup IDB(L)$), from goals $\leftarrow i_0(X,Y)$ to goals (assume each $\alpha_j = R_{j,1} \dots R_{j,l_j}$, for some l_j, given in the definition of L)

$$\begin{array}{l}
\leftarrow r_{1,1}(X, X_1^{1,1}) \wedge r_{1,2}(X_1^{1,1}, X_2^{1,1}) \wedge \dots \wedge r_{1,l_1}(X_{l_1-1}^{1,1}, X_{l_1}^{1,1}) \wedge \\
\quad r_{1,1}(X_{l_1}^{1,1}, X_1^{1,2}) \wedge \dots \quad \dots \quad \dots \quad \dots \wedge r_{1,l_1}(X_{l_1-1}^{1,2}, X_{l_1}^{1,2}) \wedge \\
\quad \dots \wedge \\
\quad r_{1,1}(X_{l_1}^{1,k_1-1}, X_1^{1,k_1}) \wedge \dots \quad \dots \quad \dots \quad \wedge r_{1,l_1}(X_{l_1-1}^{1,k_1}, X_{l_1}^{1,k_1}) \wedge \\
\quad r_{2,1}(X_{l_1}^{1,k_1}, X_1^{2,1}) \wedge \dots \quad \dots \quad \dots \quad \dots \wedge r_{2,l_2}(X_{l_2-1}^{2,1}, X_{l_2}^{2,1}) \wedge \\
\quad \dots \wedge \\
\quad r_{2,1}(X_{l_2}^{2,k_2-1}, X_1^{2,k_2}) \wedge \dots \quad \dots \quad \dots \quad \wedge r_{2,l_2}(X_{l_2-1}^{2,k_2}, X_{l_2}^{2,k_2}) \wedge \\
\quad \dots \wedge \\
\quad r_{n,1}(X_{l_{n-1}}^{n-1,k_{n-1}}, X_1^{n,1}) \wedge \dots \quad \dots \quad \dots \wedge r_{n,l_n}(X_{l_n-1}^{n,1}, X_{l_n}^{n,1}) \wedge \\
\quad \dots \wedge \\
\quad r_{n,1}(X_{l_n}^{n,k_n-1}, X_1^{n,k_n}) \wedge \dots \quad \dots \quad \dots \wedge r_{n,l_n}(X_{l_n-1}^{n,k_n}, Y)
\end{array}$$

for some distinct variables $X_1^{1,1}, \dots, X_{l_n-1}^{n,k_n}$ (in turn distinct from X and Y). We will refer to any such goal as $\leftarrow goal(\alpha_1^{k_1} \dots \alpha_n^{k_n})$. If $k_j = 0$, for some $j = 1, \dots, n$, then the conjuncts corresponding to α_j (i.e. the conjuncts in the predicates $r_{j,i}$, for $i = 1, \dots, l_j$) are missing and $X_{l_j}^{j,k_j} = X_{l_{j-1}}^{j-1,k_{j-1}}$. If $k_1 = \dots = k_n = 0$ then $\leftarrow goal(\alpha_1^{k_1} \dots \alpha_n^{k_n})$ is $\leftarrow X = Y$.

Then, for any such word $\alpha_1^{k_1} \dots \alpha_n^{k_n}$ ($\alpha_j = R_{j,1} \dots R_{j,l_j}$, for some l_j, given in the definition of L), assume $(u,v) \in Q_L(EDB)$. Then, trivially there is a refutation for $\leftarrow goal(\alpha_1^{k_1} \dots \alpha_n^{k_n})$ in EDB (and therefore in $EDB \cup IDB(L)$) returning the substitution $\{X/u, Y/v\}$, i.e. $(u,v) \in Q_{i_0}^{IDB(L)}(EDB)$.

Conversely, for any derivation from $\leftarrow i_0(X,Y)$ to $\leftarrow goal(\alpha_1^{k_1} \dots \alpha_n^{k_n})$, assume there is a refutation for $\leftarrow goal(\alpha_1^{k_1} \dots \alpha_n^{k_n})$ in EDB (and therefore in $EDB \cup IDB(L)$) returning the substitution $\{X/u, Y/v\}$, i.e. $(u,v) \in Q_{i_0}^{IDB(L)}(EDB)$. Then, trivially $(u,v) \in Q_L(EDB)$. *q.e.d*

It is easy to see that the Datalog program corresponding to any pseudo-regular language is piecewise linear and therefore linearisable (see section 2).

4 Conclusions and Future Work

We have investigated the linearisability of three classes of chain queries. To the best of our knowledge, the third class, of pseudo-regular chain queries, has not been studied elsewhere in the literature.

As a subject of future work, it seems possible to prove that the class of pseudo-regular chain queries is the biggest class of linearisable chain queries, in the sense that all linear programs corresponding to chain queries correspond to the union of pseudo-regular chain queries.

Acknowledgements

This research was supported by the EEC HCM Project no. CHRX-CT93-00414 "Logic Program Synthesis and Transformation".

References

[1] F. Afrati and S.S. Cosmadakis, Expressiveness of restricted recursive queries, *ACM* 1989.

[2] F. Afrati, M. Gergatsoulis and M. Katzouraki, On Transformations into linear database logic programs

[3] M.H. van Emden, R.A. Kowalski, The semantics of predicate logic as a programming language. *ACM* 23(4):733–742 (1976)

[4] A. Pettorossi, M. Proietti, Transformation of logic programs. *Journal of Logic Programming* 19/20:261–320 (1994)