A GRAPHICAL QUERY LANGUAGE SUPPORTING RECURSION†

*Isabel F Cruz**
Alberto O Mendelzon
Peter T Wood

Computer Systems Research Institute
University of Toronto
Toronto, Canada M5S 1A4

ABSTRACT

We define a language G for querying data represented as a labeled graph G By considering G as a relation, this graphical query language can be viewed as a relational query language, and its expressive power can be compared to that of other relational query languages We do not propose G as an alternative to general purpose relational query languages, but rather as a complementary language in which recursive queries are simple to formulate The user is aided in this formulation by means of a graphical interface The provision of regular expressions in G allows recursive queries more general than transitive closure to be posed, although the language is not as powerful as those based on function-free Horn clauses However, we hope to be able to exploit well-known graph algorithms in evaluating recursive queries efficiently, a topic which has received widespread attention recently

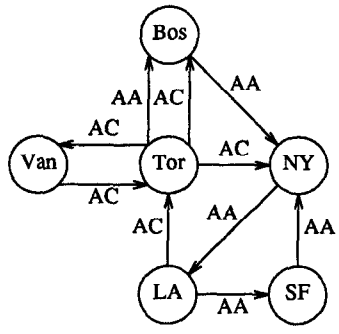
1 INTRODUCTION

It is often the case that the data comprising an application can be represented most naturally in the form of a graph structure In order to extract information from such a representation, users need a suitable query language One method of providing this service would be to transform the graph into a relation and use a relational query language such as SQL However, this solution suffers from two disadvantages Firstly, the graphical nature of the data is no longer apparent, and secondly, there are useful queries, such as finding the transitive closure of a graph, that cannot be expressed in traditional relational query languages [Aho79] As a result, our approach is different In this paper we define a graphical query language G tailored to querying data which is represented as a graph This language has sufficient expressive power to enable users to pose queries, including transitive closure, which are not expressible in relational query languages Furthermore, the formulation of such queries by the user is facilitated by means of a graphical interface, through which the user constructs and manipulates both query and answer graphs

Recently, there have been a number of proposals for more powerful relational query languages [Daya86], many of them based on Horn clauses [Hens84, Chan85, Ullm85] However, efficient evaluation algorithms for such languages have been difficult to obtain and seem highly data dependent [Banc86, Sacc86] We hope that by restricting the query language slightly and exploiting existing graph algorithms, we will be able to evaluate graphical queries efficiently

The graphs over which our graphical queries are defined are labeled directed multigraphs The node labels are distinct values drawn from some domain, while edge labels are tuples of domain values

EXAMPLE 1 The following graph represents the flight information of various airlines Each node is labeled by the name of a city, while each edge is labeled by an airline name



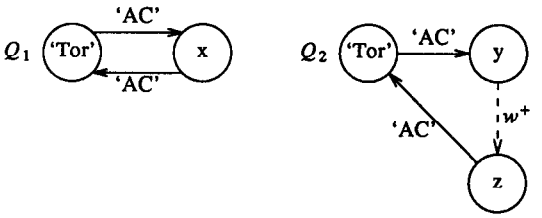
A directed edge from node 'Tor' to node 'Bos' with label 'AC' denotes the fact that Air Canada has a flight from Toronto to Boston □

A *graphical query* Q on a graph G is a set of labeled directed multigraphs, in which the node labels of Q may be either variables or constants, and the edge labels are regular expressions defined over n-tuples of variables and constants An edge which is labeled by a regular expression containing the positive closure operator (+) is drawn as a dashed edge in Q

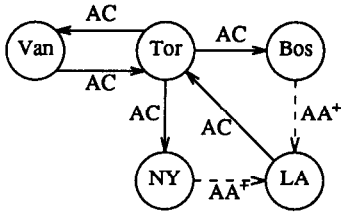
† This research was supported by the Natural Sciences and Engineering Research Council of Canada
* Research supported by an Invotan grant and a World University Service of Canada scholarship

This is done to emphasize that such edges correspond to paths of arbitrary length in G, while solid edges in Q (those whose labels contain no +) correspond to paths of fixed length The *value of Q with respect to G* is the union of all query graphs of Q which "match" subgraphs of G A formal definition of the semantics of G is postponed until Section 2 2

EXAMPLE 2 Given the graph G of Example 1, the following query $Q = \{Q_1, Q_2\}$ finds the first and last cities visited in all round trips from Toronto, in which the first and last flights are with Air Canada and all other flights (if any) are with the same airline



The following graph is the value of Q with respect to the graph G



The node x in Q_1 matches 'Van' in G, while the edge from y to z in Q_2 matches the paths <'NY','LA'> and <'Bos','NY','LA'> in G The concatenated edge labels of both these paths satisfy the regular expression 'AA$^+$' Because this query requires the computation of the transitive closure of G, it is not expressible in relational algebra [Aho79] Furthermore, the requirement that cities be linked by the same (unspecified) airline means that the query is also not expressible in the algebra extended with a transitive closure operator [Vard82] □

Apart from those query languages based on Horn clauses, other extended relational query languages have concentrated on the transitive closure operator QBE [Zloo76] allows transitive closure to be computed, but only with respect to information which can be represented as a tree or a forest Both the approaches of [Clem81] and G-Whiz [Hell85] support recursive views, but neither of them can query cyclic information The Probe project [Daya86, Rose86] is closest to our approach Probe is an extension of G-Whiz which allows cyclic structures to be queried Transitive closure is generalized to include additional information about the set of paths between any two attribute values over which transitive closure is defined However, it is not clear whether the query of Example 2 can be expressed in the Probe language In any event, we believe that the use of regular expressions makes such queries easier to express in our language The provision of various operators in Probe permits queries such as finding the shortest path to be expressed, which cannot be achieved in our present formulation However, we are in the process of adding suitable operators to our language in order to gain this additional expressive power

The remainder of this paper is divided into four main sections In the next section, the syntax and semantics of the graphical query language G are defined Section 3 compares the expressive power of G with that of H, the language of Horn clause programs [Chan85] An initial implementation of G, in which queries are translated to Prolog programs [Cloc81], is discussed briefly in Section 4 Finally, a number of further research issues are suggested in Section 5

2 GRAPHICAL QUERIES

The syntax and semantics of the graphical query language G are defined in this section Before discussing the syntax and semantics of G, it is necessary to give a more precise definition of the graphs over which the expressions of G are defined

A *labeled directed (multi-) graph* G is an ordered quintuple

$$(N_G, E_G, \psi_G, \nu_G, \varepsilon_G),$$

where N_G is a set of *nodes* E_G is a set of *directed edges*, ψ_G is the *incidence function* that associates with each edge of G an ordered pair of nodes of G, ν_G is a one-to-one *node labeling function* that associates with each node a distinct value drawn from domain D_0, and ε_G is an *edge labeling function*, which associates with each edge an n-tuple of values drawn from domains $D_1, \ldots, D_n$ In Example 1, D_0 = {'Bos','Van','Tor','NY','LA','SF'} and D_1 = {'AA','AC'} If $e = (x,y)$ is an edge in E_G, then x is the *tail* of e and y is the *head* of e Given two edges e_i and e_j in E_G such that $\psi_G(e_i) = \psi_G(e_j)$, then $\varepsilon_G(e_i) \neq \varepsilon_G(e_j)$ We will call this the *distinct edge label property* In addition, there are no isolated nodes in G From now on, G will be referred to simply as a graph, and directed edges will simply be called edges

2 1 Syntax

Given a graph $G = (N_G, E_G, \psi_G, \nu_G, \varepsilon_G)$, an expression of G, that is, a graphical query, is a set $\{Q_1, \ldots, Q_p\}$ of labeled directed (multi-) graphs Let

$$Q = (N_Q, E_Q, \psi_Q, \nu_Q, \varepsilon_Q)$$

be one of these graphs, and let $X = \{x_1, x_2, \ldots\}$ be a set of *variables* Every node in N_Q must be the head or tail of some edge in E_Q The node labeling function ν_Q maps each node in N_Q to an element of $D_0 \cup X$, that is, a node is labeled either by a *constant* $a_i \in D_0$ or by a variable $x_i \in X$ The edge labeling function ε_Q associates with each edge in E_Q a regular expression of simple edge labels A *simple edge label* is an n-tuple $(l_1, \ldots, l_n)$ of constants, variables and underscores, such that for any constant b appearing in the i'th component of an edge label, $b \in D_i$ The *empty edge label* is also a simple edge label, it is used only when querying graphs which have no edge labels

A *sequence* of edge labels is defined as follows Each edge label $(l_1, \ldots, l_n)$ is a sequence $<(l_1, \ldots, l_n)>$ of edge labels If x and y are sequences of edge labels, then so is the *concatenation* $<x, y>$ of x and y Let S_1 and S_2 be sets of sequences of labels The set S_1S_2, called the *concatenation* of S_1 and S_2, is

$$\{ <x, y> \mid x \in S_1 \text{ and } y \in S_2 \}$$

If S is a set of sequences of labels, define $S^{i+1} = SS^i$ for $i \geq 1$, and the *positive closure* of S as the set

$$S^+ = \bigcup_{i=1}^{\infty} S^i$$

Let L be the set of simple edge labels The *regular expressions over L* and the sets that they denote are defined recursively as follows [Aho74]

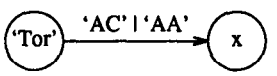
1 For each label l in L, l is a regular expression and denotes the set $\{l\}$

2 If s_1 and s_2 are regular expressions denoting the sets S_1 and S_2 respectively, then the *alternation* of s_1 and s_2, written $s_1 | s_2$, and the sequence $<s_1, s_2>$ are regular expressions that denote the sets $S_1 \cup S_2$ and $S_1 S_2$ respectively

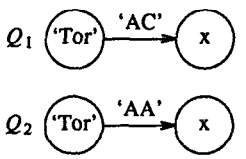
3 If s is a regular expression denoting the set S, then the *positive closure* of s, written s^+, is a regular expression denoting the set S^+

Edges whose labels are formed by using only the first two rules above are called *solid edges*, while those whose labels are constructed using rule 3 are called *dashed edges*

EXAMPLE 3 Referring back to the graph of Example 1, the following query Q will return those cities reachable from Toronto using only a single Air Canada or American Airlines flight



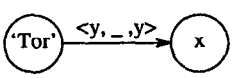
This is equivalent to the following set $\{Q_1, Q_2\}$ of queries



□

The underscore can be viewed as a shorthand notation for the alternation of all relevant domain constants appearing in the graph G That is, if an underscore appears as the i'th component of an n-tuple, it denotes the regular expression $d_1 | \quad | d_m$, where $D_i = \{d_1, \quad, d_m\}$ As a result, the positive closure of the n-tuple of underscores denotes the set of all sequences of simple labels which contain only constants

EXAMPLE 4 The query to find the cities reachable from Toronto in a sequence of three flights such that the first and last flights are with the same airline could be expressed as follows



The underscore is shorthand for 'AC' | 'AA' □

EXAMPLE 5 The dashed edge label given by the regular expression $AC^+ | <AC, AA>^+$ would match the edge labels on paths where either all the flights were with Air Canada or the flights alternated between Air Canada and American Airlines □

2.2 Semantics

We shall now define the value of a graphical query Q with respect to a graph G Given an expression Q of G which is a set $\{Q_1, \quad, Q_p\}$ of query graphs, the *value of Q with respect to G* is simply

$$Q(G) = Q_1(G) \cup \quad \cup Q_p(G),$$

where $Q_i(G)$ is the value of Q_i with respect to G The graph union operator is defined in such a way that it preserves the distinct edge label property Next, we will define the semantics when Q is a single query graph

The concept of a valuation is used to define a mapping from the variables in Q to values in the domains of G Let $Q = (N_Q, E_Q, \psi_Q, v_Q, \varepsilon_Q)$ be a query graph which is to be evaluated with respect to the graph $G = (N_G, E_G, \psi_G, v_G, \varepsilon_G)$, whose nodes labels are defined over D_0, and whose edge labels are defined over $D_1 \times \quad \times D_n$ A *valuation* ρ of Q is a pair (ρ_1, ρ_2) of mappings The *node valuation* ρ_1 is a one-to-one mapping from node labels to elements of the domain D_0, such that if c is a constant, then $\rho_1(c) = c$ The *edge valuation* ρ_2 is a mapping from the constants and variables that appear in edge labels to domain values such that (1) if c is a constant, then $\rho_2(c) = c$, and (2) if x is a variable appearing in the i'th component of a tuple in an edge label, then $\rho_2(x) \in D_i$ The mapping ρ_2 can be extended to map simple edge labels to tuples of domain values In addition, given an edge e in E_Q, let $\rho_2(\varepsilon_Q(e))$ denote the result of applying ρ_2 to each simple label appearing in the regular expression $\varepsilon_Q(e)$, and let $S(\rho_2, \varepsilon_Q, Q, e)$ be the set of sequences of simple labels denoted by $\rho_2(\varepsilon_Q(e))$

EXAMPLE 6 A valuation $\rho = (\rho_1, \rho_2)$ for the query of Example 4 is given by

ρ_1('Tor') = 'Tor', ρ_1(x) = 'LA',
ρ_2(y) = 'AA', ρ_2('AC') = 'AC', ρ_2('AA') = 'AA'

From now on, when defining valuations we will usually omit the definitions for constant labels □

The semantics of the graphical query language is defined using a simplified form of mapping between graphs known as a subgraph homeomorphism [LaPa78, Fort80] This mapping seems to capture our intention that the user should think of the edges in a query being matched to paths in the graph being queried It is first necessary to define a simple path in a graph A *simple path P* in a graph G is a sequence

$$<v_1, e_1, v_2, e_2, \quad, e_{n-1}, v_n>,$$

where $v_i \in N_G$, $v_i \neq v_j$, $1 \leq i, j \leq n$, and $e_k \in E_G$, $1 \leq k \leq n-1$, such that $\psi_G(e_k) = (v_k, v_{k+1})$, $1 \leq k \leq n-1$ The *edge label sequence induced by P* is given by

$$<\varepsilon_G(e_1), \quad, \varepsilon_G(e_{n-1})>$$

An *edge-independent subgraph homeomorphism* between a query graph Q and a graph G is defined as a pair $\mu = (\mu_1, \mu_2)$ of one-to-one mappings, where μ_1 maps nodes of Q to nodes of G, and μ_2 maps edges of Q to simple paths in G The traditional definition of subgraph homeomorphism requires that the paths in G to which the edges of Q map are pairwise node-disjoint [Fort80] We use the term "edge-independent" in our definition since each edge in Q can be mapped to any simple path in Q, independently of the other edges in Q Some justification of this choice for the semantics of G is given towards the end of this section From now on, we will refer to edge-independent subgraph homeomorphisms simply as homeomorphisms

Given a valuation $\rho = (\rho_1, \rho_2)$ of Q, the homeomorphism $\mu = (\mu_1, \mu_2)$ is said to *preserve* ρ if for each node x in Q,

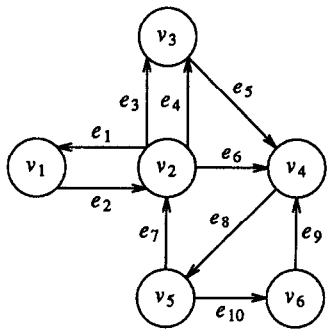
$$\rho_1(v_Q(x)) = v_G(\mu_1(x)),$$

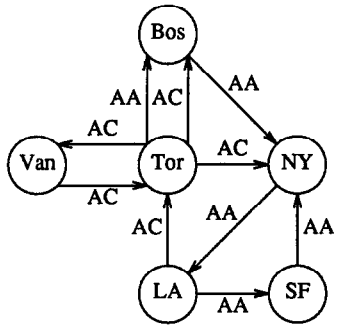
and for each edge e in Q, the edge label sequence induced by the simple path $\mu_2(e)$ in G is in the set $S(\rho_2, \varepsilon_Q, Q, e)$, that is, the set denoted by the regular expression $\rho_2(\varepsilon_Q(e))$

The *value of* Q *with respect to* G, denoted $Q(G)$, is the union of the set of graphs

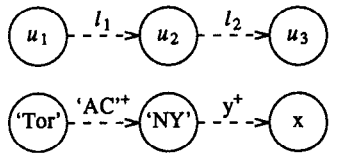
{ $\rho(Q)$ | ρ is a valuation of Q and there is a homeomorphism between Q and G which preserves ρ }

EXAMPLE 7 Let us return to the graph G of Example 1 The following two graphs provide definitions for the structure of G as well as for the labeling functions





Consider the following query Q, where only the second of the two graphs (that specifying the labels) would actually be input by the user



Let a valuation $\rho=(\rho_1,\rho_2)$ of Q be given by

$$\rho_1(x)=\text{'LA'},\ \rho_2(y)=\text{'AA'}$$

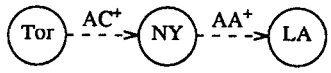
One homeomorphism $\mu=(\mu_1,\mu_2)$ from Q to G is given by

$$\mu_1(u_1)=v_2,\ \mu_1(u_2)=v_4,\ \mu_1(u_3)=v_5,$$

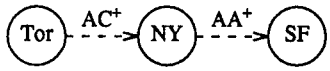
and

$$\mu_2(l_1)=\langle v_2,e_6,v_4\rangle,\ \mu_2(l_2)=\langle v_4,e_8,v_5\rangle$$

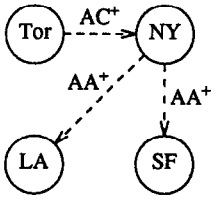
The mapping μ_1 preserves the node valuation ρ_1 since $\rho_1(v_Q(u_i))=v_G(\mu_1(u_i))$ for all nodes $u_i\in N_Q$ For example, $\rho_1(v_Q(u_3))=\text{'LA'}=v_G(\mu_1(u_3))$ The edge label sequence induced by $\mu_2(l_1)$ is <'AC'>, and that induced by $\mu_2(l_2)$ is <'AA'> Since each of these is in the set denoted by the edge valuation ρ_2 applied to the corresponding regular expression in Q, μ_2 preserves ρ_2 So μ preserves ρ, and $\rho(Q)$ which is



is a subgraph of the answer to Q Another valuation which is preserved by a homeomorphism from Q to G is ρ', which is identical to ρ except that $\rho_1'(x)=$'SF' The homeomorphism μ' which preserves ρ' is the same as μ, except that $\mu_1'(u_3)=v_6$ and $\mu_2'(l_2)=\langle v_4,e_8,v_5,e_{10},v_6\rangle$ The valuation ρ' is preserved since <'AA','AA'> satisfies 'AA'+ Therefore, the following graph $\rho'(Q)$ is another subgraph of the answer

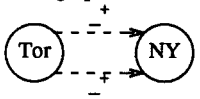


The paths $\langle v_2,e_3,v_3,e_5,v_4\rangle$ and $\langle v_2,e_4,v_3,e_5,v_4\rangle$ do not preserve any valuation of Q (since $\varepsilon_G(e_5)=$'AA' and $\varepsilon_Q(l_1)=$'AC'+), so l_1 cannot be mapped to either of these paths Since no other subgraphs contribute to the answer, $Q(G)$ is given by



□

We now provide some justification for our choice of a homeomorphism different from the traditional mapping For this purpose, it is useful to consider the answer of the following query Q with respect to the graph G of Example 7



If the semantics of G were based on the conventional definition of homeomorphism, this query would request all pairs of *disjoint* paths from Toronto to New York There are three paths in G from Toronto to New York,

$$p_1=\langle v_2,e_6,v_4\rangle,$$

$$p_2=\langle v_2,e_3,v_3,e_5,v_4\rangle, \text{ and}$$

$$p_3=\langle v_2,e_4,v_3,e_5,v_4\rangle,$$

and two disjoint pairs of paths, (p_1,p_2) and (p_1,p_3) If the answer to the query Q is the union of these pairs, then it is possible for the user to deduce incorrectly that p_2 and p_3 are disjoint, since this information is lost in forming the union An alternative is to present individual answers to the user one at a time, but this is both less elegant and would require additional processing to group answers where possible in order to try to avoid producing an exponential number of solutions We also feel that evaluating queries with these semantics may be more costly, although we have no results to support this conjecture It should be noted that, using our chosen semantics, one of the dashed edges in the above query is redundant

3 EXPRESSIVE POWER

In this section, the expressive power of G is compared to that of relational query languages, specifically the language H of Horn clause programs [Chan85] Before doing so, it is necessary to be able to view a query of G as a mapping between relations rather than graphs, that is, to provide relational semantics for G Given a graph G and a query Q on G, we show how to interpret both G and $Q(G)$ as relations

3 1. Relational semantics for G

Given a graph $G=(N_G, E_G, \psi_G, v_G, \varepsilon_G)$ in which edge labels are n-tuples of domain values, it is straightforward to construct a relation r corresponding to G Let the relation scheme for r be given by $R=(A_1, A_2, B_1, \ldots, B_n)$, where $dom(A_1)=dom(A_2)=D_0$ (the domain of the node labels in G) and $dom(B_i)=D_i$, $1\le i\le n$ For every edge $e\in E_G$ with $\psi_G(e)=(x,y)$, where $v_G(x)=v_1$, $v_G(y)=v_2$, and $\varepsilon_G(e)=(l_1, \ldots, l_n)$, there is a tuple $(v_1, v_2, l_1, \ldots, l_n)$ in r The distinct edge label property ensures that r is indeed a relation Conversely, given a relation r in which two attributes are defined over the same domain, it is also simple to produce a graph G corresponding to r

EXAMPLE 8 The relation r of the graph G given in Example 1 is shown below The relation scheme is *flight = (from, to, airline)*

from	to	airline
Tor	Van	AC
Tor	Bos	AA
Tor	Bos	AC
Tor	NY	AC
Van	Tor	AC
Bos	NY	AA
NY	LA	AA
LA	Tor	AC
LA	SF	AA
SF	NY	AA

□

In order to interpret $Q(G)$ as a relation, it is convenient to add a summary table to the syntax of G Given a set Q of p graphical queries $\{Q_1, \ldots, Q_p\}$, a *summary table* T is a set $\{t_1, \ldots, t_p\}$ of k-tuples of constants and variables from X, such that each variable x_i appearing in tuple t_j must label some node in Q_j or be a component of some edge label of Q_j Intuitively, each tuple t_i of T defines the output relation r_i for query Q_i, the value of Q being given by the union of the relations $r_1, \ldots, r_p$

Let Q be an expression of G, that is, Q is a set $\{Q_1, \ldots, Q_p\}$ of query graphs, and let $T=\{t_1, \ldots, t_p\}$ be the summary table for Q Given a relation r with scheme $R=(A_1, A_2, B_1, \ldots, B_n)$, the *value of Q with respect to r and T* is

$$Q(r,T)=Q_1(r,t_1)\cup \ldots \cup Q_p(r,t_p),$$

where each $Q_i(r,t_i)$, $1\le i\le p$, is a relation which is the value of query Q_i with respect to relation r and summary row t_i Let G be the graph of r and $Q(G)$ be the value of Q with respect to G

The value of a single query graph Q with respect to r and summary row t is defined as

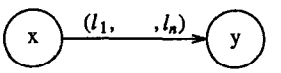
$$Q(r,t)=\{\rho(t) \mid \rho \text{ is a valuation of } Q \text{ and } \rho(Q) \text{ is isomorphic to a subgraph of } Q(G)\}$$

EXAMPLE 9 Returning to the query Q of Example 2, let the summary table $T=\{t_1, t_2\}$, where $t_1=(x,x,\text{'AC'})$ and $t_2=(y,z,w)$ The value of Q with respect to r and T is the following relation

Van	Van	AC
Bos	LA	AA
NY	LA	AA

□

EXAMPLE 10 Given a graph G, the identity query (shown below), with summary table consisting of the single tuple $(x,y,l_1, \ldots, l_n)$, yields the same relation as would be produced by the method outlined at the beginning of this section



□

3.2 Horn clause queries

It seems most appropriate to assess the expressive power of G by comparing it to the language H of *Horn clause queries* introduced in [Chan85] We will not repeat the definition of H here, but will only highlight some of the differences between H and the usual definition of function-free Horn clauses The predicate symbols of H are partitioned into *terminal relation symbols*, which correspond to base relations, and *nonterminal relation symbols* Since we are dealing with queries over a single relation, we will assume there is only a single terminal relation symbol R, apart from $=$ and $\neq$ which are special terminal relation symbols

In order to view a program P of H as representing a query, one of the nonterminal relation symbols of P is identified as the *carrier* that produces the result of the query The carrier will usually be denoted by S If S is of arity m, we define the *query represented by P* as

$$P(R)=\{(d_1, \ldots, d_m) \mid P \vdash S(d_1, \ldots, d_m)\}$$

A method for constructing a program of H from a query of G is given in [Mend86] Rather than reproducing the algorithm here, we will demonstrate some of its features by means of an example

EXAMPLE 11 Consider the graphical query Q of Example 7, with the summary row (x,y) The following program P, with carrier S, would be constructed from Q by the algorithm

C_1 $S(x,y) \leftarrow E_1(Tor,NY), E_2(NY,x,y), NY \neq x, x \neq Tor$

C_2 $E_1(Tor,NY) \leftarrow T_1(Tor,NY)$

C_3 $T_1(Tor,NY) \leftarrow E_{11}(Tor,z), T_1(z,NY), Tor \neq z, z \neq NY$

C_4 $T_1(Tor,NY) \leftarrow E_{11}(Tor,NY)$

C_5 $E_{11}(Tor,z) \leftarrow R(Tor,z,AC)$

C_6 $E_2(NY,x,y) \leftarrow T_2(NY,x,y)$

C_7 $T_2(NY,x,y) \leftarrow E_{21}(NY,z,y), T_2(z,x,y), NY \neq z, z \neq x$

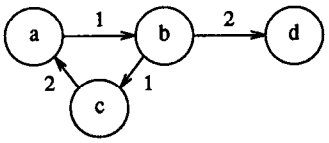
C_8 $T_2(NY,x,y) \leftarrow E_{21}(NY,x,y)$

C_9 $E_{21}(NY,x,y) \leftarrow R(NY,x,y)$

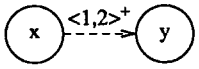
The right-hand side of C_1 contains a nonterminal literal for each edge in Q Subsequent clauses provide the definitions for these literals C_2 to C_5 define E_1, and C_6 to C_9 define E_2 A recursive clause is produced if an edge in Q is labeled by a regular expression containing the positive closure operator As a result, the definitions of both E_1 and E_2 contain recursive clauses (C_3 and C_7) The inequalities in P ensure that the assignment of values to those variables in P which correspond to node labels in G obey the restriction that such assignments are one-to-one □

We say that the translation of a query Q, with summary table T, to a program P, with carrier S, is correct if $P(R)=Q(r,T)$ Our construction of P from Q is correct if either Q is nonrecursive or the graph G corresponding to the relation r is *acyclic* If G is cyclic and Q is recursive, there is no guarantee that the translation will be correct The difficulty arises in enforcing that only simple paths in G are traversed by P, and adding inequalities to the appropriate clauses of P is not sufficient to prevent non-simple paths from being traversed The translation in the previous example is correct because, although non-simple paths may be examined by P, for every non-simple path from x to y which satisfies the restrictions of the query, there is a simple path from x to y which also satisfies them The next example, however, demonstrates that this is not always true

EXAMPLE 12 Consider the following graph G



and the query Q



with summary table (x,y) The value of $Q(G)$ is $\{(a,d),(b,a)\}$ However, the program P for Q would also produce the tuple (b,d), since there is no way of preventing (b,a) from combining with (a,d) to form (b,d), even using inequalities In this example, there is a non-simple path between b and d which satisfies the regular expression $<1,2>^+$, but no simple path between b and d which satisfies it □

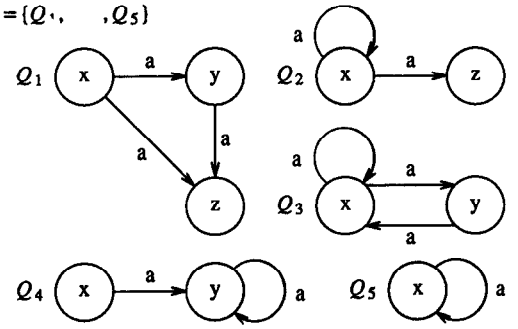
It is not hard to see that every nonrecursive program of **H** can be expressed as a graphical query Let P be a nonrecursive program of **H**, with carrier S Then P can be transformed to P', where no nonterminal symbol appears in the body of any clause of P', and the head of every clause in P' has the same predicate symbol, namely S For each clause C_i of P', construct a graphical query Q_i as follows Initially $N_{Q_i}=\emptyset$ and $E_{Q_i}=\emptyset$ Each atomic formula $R(v_1,v_2,l_1, \ldots ,l_n)$ in the body of C_i contributes an edge e to Q_i, so that $E_{Q_i}=E_{Q_i} \cup \{e\}$, $N_{Q_i}=N_{Q_i} \cup \{x,y\}$, $\psi_{Q_i}(e)=(x, y)$, $v_{Q_i}(x)=v_1$, $v_{Q_i}(y)=v_2$, and $\varepsilon_{Q_i}(e)=(l_1, \ldots ,l_n)$ If the head of C_i is $S(z_1, \ldots ,z_k)$, add the tuple $(z_1, \ldots ,z_k)$ to the summary table T The query Q consists of the set of all such queries Q_i produced in this way along with the summary table T

For a query Q constructed by the above process alone, it may be the case that $Q(r,T) \subset P(R)$ This is a consequence of the one-to-one node mappings, which force node variables to be mapped to distinct values However, the problem can be solved by including additional queries Q_{ij} in the set Q by contracting edges (identifying nodes) of Q_i while preserving the distinct edge label property The summary row t_{ij} for Q_{ij} is the same as that for Q_i, except that those variables appearing in t_{ij} that correspond to nodes in Q_i which have been identified are equated If $|N_{Q_i}|=n$, there may be $O(2^n)$ such queries Q_{ij} generated, although for certain queries on acyclic graphs none of these additional queries is needed The following example illustrates the above process

EXAMPLE 13 Consider the following program P

$$S(x,y,z) \leftarrow R(x,y,a), R(y,z,a), R(x,z,a)$$

The above procedure produces the following graphical query $Q=\{Q_1, \ldots ,Q_5\}$



The summary table T for Q is

x	y	z
x	x	z
x	y	x
x	y	y
x	x	x

If the graph of R is acyclic, then Q_1 with summary table $\{t_1\}$ is equivalent to P □

A consequence of the above translation is that **G** has greater expressive power than both the conjunctive queries [Chan77] and the tableau queries [Aho79a] However, it is not obvious exactly which subset of the recursive queries expressible in **H** can be expressed in **G** In the present formulation of **G**, there are queries expressible in **H** which appear not to be expressible in **G**

EXAMPLE 14 Consider the flights relation scheme with two additional attributes giving the departure and arrival times of flights, that is, *flight = (from, to, dep, arr, airline)* The query Q that finds those cities connected by flights where the arrival time of one flight is equal to the departure time of the next flight, appears not to be expressible in **G** However, Q can be expressed in **H** as demonstrated by the following program

$$S(x_1,x_2) \leftarrow T(x_1,x_2,y_1,y_2)$$

$$T(x_1,x_2,y_1,y_2) \leftarrow R(x_1,x_3,y_1,y_3,z), T(x_3,x_2,y_3,y_2)$$

$$T(x_1,x_2,y_1,y_2) \leftarrow R(x_1,x_2,y_1,y_2,z) \square$$

The above query Q requires finding the transitive closure of two pairs of attributes simultaneously If **G** is modified to permit node labels to be defined over sets of attributes, then Q can be expressed in **G** by labeling nodes with *(from, dep)* and *(to, arr)* pairs, while labeling edges with airlines The various ways in which **G** might be extended and the additional expressive power gained through such extensions are currently under investigation

4 IMPLEMENTATION

We have written a prototype implementation of **G**, in which a graphical query is compiled into a C-Prolog program The compiler accepts queries written in an equivalent string representation of **G**, whose syntax is specified using a context-free grammar This allows the implementation to be independent of the graphical interface, which is currently under development on a Sun 3 workstation The UNIX† tools Lex and Yacc were used to develop a parser for the language Given a graphical query Q, the compiler constructs a parse tree which is traversed in pre-order to generate a Prolog program equivalent to Q

Certain non-Horn clause constructs available in Prolog are used to ensure that only simple paths are traversed by any program generated by the compiler This overcomes the problem raised by the query Q of Example 12, whose translation into a Prolog program P is given below

```
simple1(X, Y, Visited, [ X | Visited ]) -
        r(X, Y, 1),
        not member(X, Visited)

simple2(X, Y, Visited, [ X | Visited ]) -
        r(X, Y, 2),
        not member(X, Visited)

sequence(X, Y, Visited, NewVisited) -
        simple1(X, Z, Visited, NV),
        simple2(Z, Y, NV, NewVisited)

closure(X, Y, Visited, NewVisited) -
        sequence(X, Y, Visited, NewVisited)

closure(X, Y, Visited, NewVisited) -
        sequence(X, Z, Visited, NV),
        closure(Z, Y, NV, NewVisited)

s(X, Y) -
        closure(X, Y, [], Visited),
        not member(Y, Visited)
```

† UNIX is a trademark of Bell Laboratories

The carrier of P is s, while the rules for the nonterminal relation symbols *closure, sequence, simple1,* and *simple2* are generated while decomposing the regular expression $<1,2>^+$ By maintaining a list of visited nodes and testing for membership in this list (using the standard rules for the *member* predicate), the program ensures that no nodes of the graph are revisited Given the graph of Example 12, which is translated into the following set of Prolog facts

```
r(a, b, 1)
r(b, c, 1)
r(b, d, 2)
r(c, a, 2)
```

P produces the answer $\{(a,d), (b,a)\}$, as required

This Prolog implementation will be used to test and evaluate subsequent implementations which we anticipate will be more efficient as a result of employing graph algorithms for query evaluation

5 CONCLUSION AND FURTHER RESEARCH

We have described a language **G** for querying data which can be represented as a labeled directed graph This representation includes relations (e g *parent*) over which useful recursive queries (e g finding the *ancestor* relation) can be defined We have provided a means for specifying recursive queries in **G**, which we believe is simpler to use than comparative formulations such as algebraic operators and Horn clauses The use of regular expressions in **G** allows queries to be formulated which are not expressible in relational algebra even when it is extended with a transitive closure operator

There are a number of topics for further research on the graphical query language It would be useful to increase the expressive power of the language further by adding operators to the language in a manner similar to [Rose86] These operators are defined over paths in the graph, and permit queries such as finding shortest paths to be computed We are currently investigating the use of graph algorithms as a means for evaluating graphical queries efficiently Related to this is the possibility that properties of the graph being queried (such as acyclicity) can be exploited during evaluation It is also sometimes the case that graphical queries can contain some redundancies This suggests the possibility of "optimizing" graphical queries, for example, by removing redundant edges

References

Aho74

A V AHO, J E HOPCROFT, AND J D ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974

Aho79a

A V AHO, Y SAGIV, AND J D ULLMAN, "Efficient Optimization of a Class of Relational Expressions," *ACM Trans on Database Syst*, vol 4, no 4, pp 435-454, 1979

Aho79

A V AHO AND J D ULLMAN, "Universality of Data Retrieval Languages," *Proc 6th ACM Symp on Principles of Programming Languages*, pp 110-120, 1979

Banc86

F BANCILHON, D MAIER, Y SAGIV, AND J D ULLMAN, "Magic Sets and Other Strange Ways To Implement Logic Programs," *Proc 5th ACM SIGACT—SIGMOD Symp on Principles of Database Systems*, pp 1-15, 1986

Chan85

A K CHANDRA AND D HAREL, "Horn Clause Queries and Generalizations," *J Logic Programming*, vol 2, no 1, pp 1-15, 1985 Originally appeared as "Horn Clauses and the Fixpoint Query Hierarchy", *Proc 1st ACM SIGACT—SIGMOD Symp on Principles of Database Systems*, pp 158-163, 1982

Chan77

A K CHANDRA AND P M MERLIN, "Optimal Implementation of Conjunctive Queries in Relational Data Bases," *Proc 9th ACM Symp on Theory of Computing*, pp 77-90, 1977

Clem81

E CLEMONS, "Design of an External Schema Facility to Define and Process Recursive Structures," *ACM Trans on Database Syst*, vol 6, no 2, pp 295-311, 1981

Cloc81

W F CLOCKSIN AND C S MELLISH, *Programming in Prolog*, Springer-Verlag, 1981

Daya86

U DAYAL AND J M SMITH, "PROBE A Knowledge-Oriented Database Management System," in *On Knowledge Base Management Systems Integrating Artificial Intelligence and Database Technologies*, ed M L Brodie and J Mylopoulos, pp 227-257, Springer-Verlag, 1986

Fort80

S FORTUNE, J HOPCROFT, AND J WYLLIE, "The Directed Subgraph Homeomorphism Problem," *Theor Comput Sci*, vol 10, pp 111-121, 1980

Heil85

S HEILER AND A ROSENTHAL, "G-WHIZ, a Visual Interface for the Functional Model with Recursion," *Proc 11th Conf on Very Large Data Bases*, 1985

Hens84

L J HENSCHEN AND S A NAQVI, "On Compiling Queries in Recursive First-Order Databases," *J ACM*, vol 31, no 1, pp 47-85, 1984

LaPa78

A S LAPAUGH AND R L RIVEST, "The Subgraph Homeomorphism Problem," *Proc 10th Ann ACM Symp on Theory of Computing*, pp 40-50, 1978

Mend86

A O MENDELZON AND P T WOOD, "A Graphical Query Language Supporting Recursion," Tech Report CSRI-183, Univ of Toronto, 1986

Rose86

A ROSENTHAL, S HEILER, U DAYAL, AND F MANOLA, "Traversal Recursion A Practical Approach to Supporting Recursive Applications," *Proc ACM SIGMOD Conf on Management of Data*, pp 166-176, 1986

Sacc86

D SACCA AND C ZANIOLO, "On the Implementation of a Simple Class of Logic Queries," *Proc 5th ACM SIGACT—SIGMOD Symp on Principles of Database Systems*, pp 16-23, 1986

Ullm85

J D ULLMAN, "Implementation of Logical Query Languages for Databases," *ACM Trans on Database Syst*, vol 10, no 3, pp 289-321, 1985 Originally appeared as Stanford Univ, Dept of Computer Science TR (May 1984)

Vard82

M Y VARDI, "The Complexity of Relational Query Languages," *Proc 14th Ann ACM Symp on Theory of Computing*, pp 137-146, 1982

Zloo76

M M ZLOOF, "Query by Example Operations on the Transitive Closure," IBM Research Report, RC5526, 1976