# A Unified Approach to Path Problems

ROBERT ENDRE TARJAN

*Stanford University, Stanford, California*

ABSTRACT. A general method is described for solving path problems on directed graphs. Such path problems include finding shortest paths, solving sparse systems of linear equations, and carrying out global flow analysis of computer programs The method consists of two steps First, a collection of regular expressions representing sets of paths in the graph is constructed This can be done by using any standard algorithm, such as Gaussian or Gauss–Jordan elimination. Next, a natural mapping from regular expressions into the given problem domain is applied. The mappings required to find shortest paths are exhibited, sparse systems of linear equations are solved, and global flow analysis is carried out. The results provide a general-purpose algorithm for solving any path problem and show that the problem of constructing path expressions is in some sense the most general path problem.

KEY WORDS AND PHRASES: code optimization, compiling, Gaussian elimination, Gauss–Jordan elimination, global flow analysis, graph algorithm, linear algebra, path problem, regular expression, semilattice, shortest path, sparse matrix

CR CATEGORIES: 4 12, 4.34, 5 14, 5 22, 5.25, 5 32

## 1. *Introduction*

A fundamental problem in numerical analysis is the solution of a system of linear equations $Ax = b$, where $A$ is an $n \times n$ matrix of coefficients, $x$ is an $n \times 1$ vector of variables, and $b$ is an $n \times 1$ vector of constants. Efficient methods for solving $Ax = b$, such as Gaussian and Gauss–Jordan elimination, have long been known. These methods have been repeatedly rediscovered and applied in other contexts. For example, Floyd's shortest path algorithm [7], which is based on Warshall's transitive closure algorithm [34], is a version of Gauss–Jordan elimination. Kleene's method for converting a finite automaton into a regular expression [21] is a form of Gauss–Jordan elimination; Gaussian elimination also solves Kleene's problem [3]. In all these situations the problem of interest can be formulated as the solution of a system of linear equations defined not over the field of real numbers but over some other algebra.

In this paper we provide a unified setting for such problems. Our goal is to show that a solution to one of them can be used to solve them all. One approach to this task is to develop a minimal axiom system for which elimination techniques work (see, e.g., Aho, Hopcroft, and Ullman [1] and Lehmann [22]) and show that the problems of interest satisfy the axioms. Our approach is somewhat different and resembles that taken by Backhouse and Carré [3]; we believe that the proper setting



This research was partially supported by the National Science Foundation under Grant MCS 75-22870-A02, by the Office of Naval Research under Contract N00014-76-C-0688, by a Guggenheim Fellowship, and by Bell Laboratories

Author's present address. Bell Laboratories, Murray Hill, NJ 07974.

for such problems is the algebra of regular expressions, which is simple, well understood, and general enough for our purposes.

We use a graph-theoretic approach rather than a matrix-theoretic one because we are interested mainly in sparse problems (problems in which the coefficient matrix $A$ contains mostly zeros). Let $G$ be a directed graph with a distinguished *source vertex* $s$. The *single-source path expression problem* is to find, for each vertex $v$ in $G$, a regular expression $R(s, v)$ representing the set of all paths from $s$ to $v$. The *all-pairs path expression problem* is to find, for each pair of vertices $v$, $w$, a regular expression $R(v, w)$ representing the set of all paths from $v$ to $w$. We show that it is possible to use solutions to the single-source and all-pairs path expression problems to find shortest paths in $G$, solve systems of linear equations defined on $G$, and solve global flow problems defined on $G$. We solve these problems by providing natural homomorphisms that map the regular expressions representing path sets into the algebras in which the given problems are expressed. We define these mappings by reinterpreting the $\cup$, $\cdot$, and $*$ operations used to construct regular expressions. The technical part of our work is in showing that these mappings are indeed homomorphisms.

This paper contains nine sections. Section 2 reviews the properties of regular expressions that we use. Section 3 considers shortest path problems. Section 4 examines the solution of systems of linear equations over the real numbers. Sections 5–8 discuss various kinds of global flow analysis problems. Section 9 contains some additional remarks. The appendix contains the graph-theoretic definitions used in the paper.

## 2. *Regular Expressions and Path Expressions*

Let $\Sigma$ be a finite alphabet disjoint from $\{\Lambda, \emptyset, (,)\}$. A *regular expression* over $\Sigma$ is any expression built by applying the following rules:

(1a) "$\Lambda$" and "$\emptyset$" are *atomic* regular expressions; for any $a \in \Sigma$, "$a$" is an atomic regular expression.

(1b) If $R_1$ and $R_2$ are regular expressions, then $(R_1 \cup R_2)$, $(R_1 \cdot R_2)$, and $(R_1)^*$ are *compound* regular expressions.

In a regular expression, $\Lambda$ denotes the empty string, $\emptyset$ denotes the empty set, $\cup$ denotes set union, $\cdot$ denotes concatenation, and $*$ denotes reflexive, transitive closure (under concatenation).[1] Thus each regular expression $R$ over $\Sigma$ defines a set $\sigma(R)$ of strings over $\Sigma$ as follows:

(2a) $\sigma(\Lambda) = \{\Lambda\}$; $\sigma(\emptyset) = \emptyset$; $\sigma(a) = \{a\}$ for $a \in \Sigma$.

(2b) $\sigma(R_1 \cup R_2) = \sigma(R_1) \cup \sigma(R_2) = \{w \mid w \in \sigma(R_1) \text{ or } w \in \sigma(R_2)\}$; $\sigma(R_1 \cdot R_2) = \sigma(R_1) \cdot \sigma(R_2) = \{w_1 w_2 \mid w_1 \in \sigma(R_1) \text{ and } w_2 \in \sigma(R_2)\}$; $\sigma(R_1^*) = \bigcup_{k=0}^{\infty} \sigma(R_1)^k$, where $\sigma(R_1)^0 = \{\Lambda\}$ and $\sigma(R_1)^t = \sigma(R_1)^{t-1} \cdot \sigma(R_1)$.

Two regular expressions $R_1$ and $R_2$ are *equivalent* if $\sigma(R_1) = \sigma(R_2)$. A regular expression $R$ is *simple* if $R = \emptyset$ or $R$ does not contain $\emptyset$ as a subexpression. We can transform any regular expression $R$ into an equivalent simple regular expression by repeating the following transformations until none is applicable: (i) replace any subexpression of the form $\emptyset \cdot R_1$ or $R_1 \cdot \emptyset$ by $\emptyset$; (ii) replace any subexpression of the form $\emptyset \cup R_1$ or $R_1 \cup \emptyset$ by $R_1$; (iii) replace any subexpression of the form $\emptyset^*$ by $\Lambda$.

[1] Note that the symbol $\Lambda$ represents both the regular expression "$\Lambda$" and the empty string. Henceforth we shall avoid using quotation marks and allow the context to resolve this ambiguity; similarly for $\emptyset$, $\cup$, $\cdot$, and $*$. Our definition requires that regular expressions be completely parenthesized, but we shall freely omit parentheses in regular expressions when the meaning is clear; we assume the standard operator precedence $*$ over $\cdot$ over $\cup$

A regular expression $R$ is *unambiguous* if each string in $\sigma(R)$ is represented uniquely in $R$. A more precise definition is as follows:

(3a) $\Lambda$, $\varnothing$, and $a$ for $a \in \Sigma$ are unambiguous.
(3b) Let $R_1$ and $R_2$ be unambiguous.
  (i) $R_1 \cup R_2$ is unambiguous if $\sigma(R_1) \cap \sigma(R_2) = \varnothing$.
  (ii) $R_1 \cdot R_2$ is unambiguous if each $w \in \sigma(R_1 \cdot R_2)$ is uniquely decomposable into $w = w_1 w_2$ with $w_1 \in \sigma(R_1)$ and $w_2 \in \sigma(R_2)$.
  (iii) $R_1^*$ is unambiguous if $R_1 \neq \Lambda$ and each nonempty $w \in \sigma(R_1^*)$ is uniquely decomposable into $w = w_1 w_2 \cdots w_k$ with $w_i \in \sigma(R_1)$ for $1 \leq i \leq k$.

Note that if $\Lambda \in \sigma(R)$, then $R^*$ is ambiguous.

Let $G = (V, E)$ be a directed graph. We can regard any path in $G$ as a string over $E$, but not all strings over $E$ are paths in $G$. A *path expression* $P$ of *type* $(v, w)$ is a simple regular expression over $E$ such that every string in $\sigma(P)$ is a path from $v$ to $w$. Every subexpression of a path expression is a path expression, whose type can be determined as follows.

(4) Let $P$ be a path expression of type $(v, w)$.
  (i) If $P = P_1 \cup P_2$, then $P_1$ and $P_2$ are path expressions of type $(v, w)$.
  (ii) If $P = P_1 \cdot P_2$, there must be a unique vertex $u$ such that $P_1$ is a path expression of type $(v, u)$ and $P_2$ is a path expression of type $(u, w)$.
  (iii) If $P = P_1^*$, then $v = w$ and $P_1$ is a path expression of type $(v, w) = (v, v)$.

It is easy to verify (4) using the fact that $P$ is simple.

## 3. *Shortest Paths*

Let $G = (V, E)$ be a directed graph with an associated real-valued *cost* $c(e)$ for each edge $e$. A *shortest path* from $v$ to $w$ is a path $p = e_1, e_2, \ldots, e_k$ from $v$ to $w$ such that $\sum_{i=1}^{k} c(e_i)$ is minimum over all paths from $v$ to $w$. If $G$ contains no cycles of negative total cost, there is a shortest path from $v$ to $w$ if there is any path from $v$ to $w$. The *single-source shortest path problem* is to find, for each vertex $v$, the cost of a shortest path from $s$ to $v$, where $s$ is a distinguished source vertex. The *all-pairs shortest path problem* is to find the cost of a shortest path from $v$ to $w$ for all vertex pairs $v$, $w$.

We can use path expressions to solve shortest path problems by means of two mappings, *cost* and *shortest path*, defined as follows.

(5a) $cost(\Lambda) = 0$, $shortest\ path(\Lambda) = \Lambda$;
  $cost(\varnothing) = \infty$, $shortest\ path(\varnothing) =$ no path;
  $cost(e) = c(e)$, $shortest\ path(e) = e$ for $e \in E$.
(5b) $cost(P_1 \cup P_2) = \min\{cost(P_1), cost(P_2)\}$,
    $shortest\ path(P_1 \cup P_2) =$ **if** $cost(P_1) \leq cost(P_2)$ **then** $shortest\ path(P_1)$ **else** $shortest\ path(P_2)$;
  $cost(P_1 \cdot P_2) = cost(P_1) + cost(P_2)$,
    $shortest\ path(P_1 \cdot P_2) = shortest\ path(P_1) \cdot shortest\ path(P_2)$;
  $cost(P_1^*) =$ **if** $cost(P_1) < 0$ **then** $-\infty$ **else** 0,
    $shortest\ path(P_1^*) =$ **if** $cost(P_1) < 0$ **then** no shortest path **else** $\Lambda$.

LEMMA 1. *Let $P$ be a path expression of type $(v, w)$. If $cost(P) = \infty$, there is no path in $\sigma(P)$. If $cost(P) = -\infty$, there are paths of arbitrarily small cost in $\sigma(P)$. Otherwise, $shortest\ path(P)$ is a minimum cost path in $\sigma(P)$, and its cost is $cost(P)$.*

PROOF. Straightforward by induction on the number of operation symbols in $P$. □

THEOREM 1. *Let $P(v, w)$ be a path expression representing all paths from $v$ to $w$. If $cost(P(v, w)) = \infty$, there is no path from $v$ to $w$. If $cost(P(v, w)) = -\infty$, there are paths of arbitrarily small cost from $v$ to $w$. Otherwise, shortest path$(P(v, w))$ is a shortest path from $v$ to $w$, and its cost is $cost(P(v, w))$.*

PROOF. Immediate from Lemma 1. □

THEOREM 2. *Let $P_1(v, w)$ be a path expression such that $\sigma(P_1(v, w))$ contains at least all the simple paths from $v$ to $w$. If there is a shortest path from $v$ to $w$, shortest path$(P(v, w))$ gives one such path, and its cost is $cost(P(v, w))$.*

PROOF. Any shortest path is simple. □

By applying Theorem 1 we can use a solution to the single-source (or all-pairs) path expression problem to solve the single-source (or all-pairs) shortest path problem. By Theorem 2 it is sufficient to use path expressions representing only the simple paths if we have a separate test for negative cycles. The following theorem provides such a test.

THEOREM 3. *Let $s$ be a distinguished source vertex in $G$, and suppose that every vertex in $G$ is reachable from $s$. For every vertex $v$ let $P_1(s, v)$ be a path expression such that $\sigma(P_1(s, v))$ contains at least all the simple paths from $s$ to $v$. Then $G$ contains a negative cycle if and only if either $cost(P_1(s, v)) = -\infty$ for some $v$ or there is some edge $e$ such that $cost(P_1(s, h(e))) + c(e) < cost(P_1(s, t(e)))$.*

PROOF. Straightforward. See Ford and Fulkerson [10]. □

Although we have treated shortest path problems in a section of their own, such problems can also be formulated as continuous bounded data flow problems (see Sections 5 and 7).

## 4. *Systems of Linear Equations*

The next problem to which we apply our technique is the solution of a system $Ax = b$ of linear equations over the set $\mathbb{R}$ of real numbers [11]. This problem has pitfalls not present in the other problems we examine. The system $Ax = b$ does not always have a solution; even if it does, the solution need not be unique. Furthermore, the standard algorithms for finding a solution, such as Gaussian elimination, may not succeed even if a unique solution exists. (To deal with this difficulty, numerical analysts have devised more complicated algorithms, such as Gaussian elimination with pivoting [11].) We bypass these issues by proposing a method that either gives a solution or divides by zero.

We begin by rewriting $Ax = b$ as $b + (I - A)x = x$, where $I$ is the $n \times n$ identity matrix. Let $x_0$ be a new variable; then the system $b + (I - A)x = x$ is equivalent to

$$\begin{pmatrix} 1 \\ \bar{0} \end{pmatrix} + A' \begin{pmatrix} x_0 \\ x \end{pmatrix} = \begin{pmatrix} x_0 \\ x \end{pmatrix}, \qquad \text{where} \quad A' = \begin{pmatrix} 0 & \bar{0} \\ b & I-A \end{pmatrix},$$

and $\bar{0}$ denotes a zero matrix of the appropriate size. Let $G = (V, E)$ be the graph having $n + 1$ vertices (one for each variable $x_i$) and $m$ edges (one for each nonzero entry in $A'$) such that there is an edge $e$ with $h(e) = v_j$ and $t(e) = v_i$ if and only if the entry in row $i$ and column $j$ of $A'$ is nonzero; let $a(e)$ be the value of this entry. Then the system of equations takes the form

$$x(s) = 1; \qquad x(v) = \sum \{a(e)x(h(e)) \mid e \in E \text{ and } t(e) = v\} \quad \text{if} \quad v \neq s, \tag{1}$$

where $s = v_0$.

We solve this system by extending the mapping $a$ to regular expressions over $E$ as follows.

(6a) $a(\Lambda) = 1$; $a(\varnothing) = 0$.
(6b) $a(R_1 \cup R_2) = a(R_1) + a(R_2)$;
$a(R_1 \cdot R_2) = a(R_1)a(R_2)$;
$a(R_1^*) = 1/(1 - a(R_1))$.

Note that $a(R_1^*)$ is defined if and only if $a(R_1) \neq 1$. If $R$ is a regular expression over $E$, then $a(R)$ is a rational function of $a(e_1)$, $a(e_2)$, ..., $a(e_m)$. The following lemma gives a more precise statement of these observations. Because the operation of addition into which union is mapped is *not* idempotent, we are forced to deal only with unambiguous regular expressions.

LEMMA 2. *If $R$ is any unambiguous regular expression over $E$, there are polynomials $p$ and $q$ in variables $a(e_1)$, $a(e_2)$, ..., $a(e_m)$ such that, for particular values of $a(e_1)$, $a(e_2)$, ..., $a(e_m)$, $a(R)$ is defined if and only if $q \neq 0$, and $a(R) = p/q$ if both sides are defined. The polynomials $p$ and $q$ can be chosen so that the constant term in $q$ is 1, and the constant term in $p$ is 0 if $\Lambda \notin \sigma(R)$, 1 if $\Lambda \in \sigma(R)$.*

PROOF. By induction on the number of operation symbols in $R$. For $R = \Lambda$, $p = q = 1$ satisfies the lemma. For $R = \varnothing$, $p = 0$, $q = 1$ satisfies the lemma. For $R = e_i$, $p = a(e_i)$, $q = 1$ satisfies the lemma. Suppose $R_1$ and $R_2$ are unambiguous regular expressions for which $p_1$, $q_1$ and $p_2$, $q_2$, respectively, satisfy the lemma. If $R_1 \cup R_2$ is unambiguous, then the choice of $p = p_1q_2 + p_2q_1$, $q = q_1q_2$ satisfies the lemma for $R_1 \cup R_2$; note that $\sigma(R_1) \cap \sigma(R_2) = \varnothing$, which means that either $p_1$ or $p_2$ has a constant term of zero. If $R_1 \cdot R_2$ is unambiguous, then the choice of $p = p_1p_2$, $q = q_1q_2$ satisfies the lemma. Finally, if $R_1^*$ is unambiguous, then the choice of $p = q_1$, $q = q_1 - p_1$ satisfies the lemma. Note that $R_1^*$ unambiguous implies $\Lambda \notin \sigma(R_1)$, which means that the constant term of $p_1$ is zero. □

LEMMA 3. *If $R_1$ and $R_2$ are two equivalent unambiguous regular expressions over $E$, then $a(R_1) = a(R_2)$ whenever both $a(R_1)$ and $a(R_2)$ are defined.*

Lemma 3 is the hardest result in this paper, and we shall postpone its proof.

THEOREM 4. *For each vertex $v$, let $P(s, v)$ be an unambiguous path expression representing all paths from $s$ to $v$. If $a(P(s, v))$ is defined for all $v$, then the mapping $x$ defined by $x(v) = a(P(s, v))$ satisfies eqs.* (1).

PROOF. The only path from $s$ to $s$ in $G$ is the empty path; by Lemma 3, $x(s) = a(P(s, s)) = a(\Lambda) = 1$. If $v \neq s$, then $\bigcup\{P(s, h(e))\cdot e \mid e \in E \text{ and } t(e) = v\}$ is an unambiguous regular expression representing the set of all paths from $s$ to $v$. By Lemma 3,

$$\begin{aligned} x(v) &= a(P(s, v)) = a(\bigcup\{P(s, h(e))\cdot e \mid e \in E \text{ and } t(e) = v\} \\ &= \sum \{a(e)x(h(e)) \mid e \in E \text{ and } t(e) = v\}. \end{aligned}$$ □

Thus the mapping $a$, when defined, gives a solution to eqs. (1). It remains for us to prove Lemma 3. We employ Salomaa's method for showing the completeness of an axiom system for regular expressions [25]. We use the notation $Q \equiv R$ to denote that $\sigma(Q) = \sigma(R)$ and $a(Q) = a(R)$ whenever both $a(Q)$ and $a(R)$ are defined. An unambiguous regular expression $Q$ is *equationally characterized* in terms of unambiguous regular expressions $Q_1$, $Q_2$, ..., $Q_q$ if $Q = Q_1$ and

$$Q_i \equiv \left( \bigcup_{j=1}^{m} Q_j \cdot e_j \right) \cup D(Q_i), \tag{2}$$

where $D(Q_i) \in \{\emptyset, \Lambda\}$ and $Q_{ij} \in \{Q_k \mid 1 \le k \le q\}$ for all $j$.

LEMMA 4. *Every unambiguous regular expression over $E$ is equationally characterized.*

PROOF. By induction on the number of operation symbols in the regular expression. We have the following equivalences:

$$\emptyset \equiv \left(\bigcup_{j=1}^{m} \emptyset \cdot e_j\right) \cup \emptyset; \qquad \Lambda \equiv \left(\bigcup_{j=1}^{m} \emptyset \cdot e_j\right) \cup \Lambda;$$

$$e_j \equiv \emptyset \cdot e_1 \cup \cdots \cup \Lambda \cdot e_j \cup \cdots \cup \emptyset \cdot e_m \cup \emptyset \qquad \text{for} \quad 1 \le j \le m.$$

Thus every atomic regular expression is equationally characterized.

Suppose $Q$ and $R$ are equationally characterized. Let $Q_1, \ldots, Q_q$ be unambiguous regular expressions such that $Q = Q_1$ and eq. (2) holds. Let $R_1, \ldots, R_r$ be unambiguous regular expressions such that $R = R_1$ and the following equivalence holds:

$$R_i \equiv \left(\bigcup_{j=1}^{m} R_{ij} \cdot e_j\right) \cup D(R_i), \tag{3}$$

where $D(R_i) \in \{\emptyset, \Lambda\}$ and $R_{ij} \in \{R_k \mid 1 \le k \le r\}$ for all $j$. We shall equationally characterize $Q \cup R$, $Q \cdot R$, and $Q^*$, assuming they are unambiguous.

Let $1 \le u \le q$, $1 \le v \le r$, and suppose $Q_u \cup R_v$ is unambiguous. Combining eqs. (2) and (3) we obtain

$$\begin{aligned} Q_u \cup R_v &\equiv \left(\bigcup_{j=1}^{m} (Q_{uj} \cup R_{vj}) \cdot e_j\right) \cup D(Q_u) \cup D(R_v) \\ &\equiv \left(\bigcup_{j=1}^{m} (Q_{uj} \cup R_{vj}) \cdot e_j\right) \cup D(Q_u \cup R_v), \end{aligned} \tag{4}$$

since if $\sigma(Q_u) \cap \sigma(R_v) = \emptyset$, then $D(Q_u) = \emptyset$ or $D(R_v) = \emptyset$. Furthermore, $Q_{uj} \cup R_{vj}$ is unambiguous for $1 \le j \le m$. Thus if $Q \cup R$ is unambiguous, the set of equations (4) such that $Q_u \cup R_v$ is unambiguous equationally characterizes $Q \cup R = Q_1 \cup R_1$.

Let $1 \le v \le r$, $s \ge 0$, and $1 \le u_1 < u_2 < \cdots < u_s \le q$. Suppose $Q \cdot R_v \cup (\bigcup_{i=1}^{s} Q_{u_i})$ is unambiguous. If $D(R_v) = \emptyset$, we obtain from eqs. (2) and (3) that

$$\begin{aligned} Q \cdot R_v \cup \left(\bigcup_{i=1}^{s} Q_{u_i}\right) &\equiv \left(\bigcup_{j=1}^{m} \left(Q \cdot R_{vj} \cup \left(\bigcup_{i=1}^{s} Q_{u_i j}\right)\right) \cdot e_j\right) \cup \left(\bigcup_{i=1}^{s} D(Q_{u_i})\right) \\ &\equiv \left(\bigcup_{j=1}^{m} \left(Q \cdot R_{vj} \cup \left(\bigcup_{i=1}^{s} Q_{u_i j}\right)\right) \cdot e_j\right) \cup D\left(Q \cdot R_v \cup \left(\bigcup_{i=1}^{s} Q_{u_i}\right)\right). \end{aligned} \tag{5}$$

Furthermore, $Q \cdot R_{vj} \cup (\bigcup_{i=1}^{s} Q_{u_i j})$ is unambiguous for $1 \le j \le m$. If $D(R_v) = \Lambda$, we obtain from eqs. (2) and (3) that

$$\begin{aligned} Q \cdot R_v \cup \left(\bigcup_{i=1}^{s} Q_{u_i}\right) &\equiv \left(\bigcup_{j=1}^{m} \left(Q \cdot R_{vj} \cup Q_{1j} \cup \left(\bigcup_{i=1}^{s} Q_{u_i j}\right)\right) \cdot e_j\right) \\ &\qquad \cup D(Q_1) \cup \left(\bigcup_{i=1}^{s} D(Q_{u_i})\right) \\ &\equiv \left(\bigcup_{j=1}^{m} \left(Q \cdot R_{vj} \cup Q_{1j} \cup \left(\bigcup_{i=1}^{s} Q_{u_i j}\right)\right) \cdot e_j\right) \\ &\qquad \cup D\left(Q \cdot R_v \cup \left(\bigcup_{i=1}^{s} Q_{u_i}\right)\right). \end{aligned} \tag{6}$$

Furthermore, $Q \cdot R_{vj} \cup Q_{1j} \cup (\bigcup_{i=1}^{s} Q_{u_i j})$ is unambiguous for $1 \le j \le m$. It follows that if $Q \cdot R$ is unambiguous, we can equationally characterize $Q \cdot R = Q \cdot R_1$ in terms of

$$\left\{ Q \cdot R_v \cup \left( \bigcup_{i=1}^{s} Q_{u_i} \right) \mid 1 \le v \le r,\ s \ge 0,\ 1 \le u_1 < u_2 < \cdots < u_s \le q, \text{ and } Q \cdot R_v \cup \left( \bigcup_{i=1}^{s} Q_{u_i} \right) \text{ is unambiguous} \right\}.$$

Finally we must consider $Q^*$. Suppose $Q^*$ is unambiguous. Then $D(Q) = \emptyset$. From eq. (2) we obtain

$$Q^* \equiv \left( \bigcup_{j=1}^{m} Q_{1j} \cdot e_j \right)^* \equiv \left( \bigcup_{j=1}^{m} Q^* \cdot Q_{1j} \cdot e_j \right) \cup \Lambda. \tag{7}$$

Furthermore, $Q^* \cdot Q_{1j}$ is unambiguous for $1 \le j \le m$.

Let $s \ge 1$ and $1 \le u_1 < u_2 < \cdots < u_s \le q$. Suppose $Q^* \cdot (\bigcup_{i=1}^{s} Q_{u_i})$ is unambiguous. If $D(Q_{u_i}) = \emptyset$ for $1 \le i \le s$, then

$$Q^* \cdot \left( \bigcup_{i=1}^{s} Q_{u_i} \right) \equiv \left( \bigcup_{j=1}^{m} Q^* \cdot \left( \bigcup_{i=1}^{s} Q_{u_i j} \right) \cdot e_j \right) \cup \emptyset, \tag{8}$$

where $Q^* \cdot (\bigcup_{i=1}^{s} Q_{u_i j})$ is unambiguous for $1 \le j \le m$.

If $D(Q_{u_i}) = \Lambda$ for some (unique) $i$ such that $1 \le i \le s$, then

$$Q^* \cdot \left( \bigcup_{i=1}^{s} Q_{u_i} \right) \equiv \left( \bigcup_{j=1}^{m} Q^* \cdot \left( Q_{1j} \cup \left( \bigcup_{i=1}^{s} Q_{u_i j} \right) \right) \cdot e_j \right) \cup \Lambda, \tag{9}$$

where $Q^* \cdot (Q_{1j} \cup (\bigcup_{i=1}^{s} Q_{u_i j}))$ is unambiguous for $1 \le j \le m$. It follows that we can equationally characterize $Q^*$ in terms of

$$\{Q^*\} \cup \left\{ Q^* \cdot \left( \bigcup_{i=1}^{s} Q_{u_i} \right) \mid s \ge 1,\ 1 \le u_1 < u_2 < \cdots < u_s \le q, \text{ and } Q^* \cdot \left( \bigcup_{i=1}^{s} Q_{u_i} \right) \text{ is unambiguous} \right\}. \quad \square$$

We are now ready to prove Lemma 3. We extend $\cup$, $\cdot$, and $\equiv$ to ordered pairs of regular expressions by defining $(Q_1, R_1) \cup (Q_2, R_2) = (Q_1 \cup Q_2, R_1 \cup R_2)$, $(Q_1, R_1) \cdot (Q_2, R_2) = (Q_1 \cdot Q_2, R_1 \cdot R_2)$, $(Q_1, R_1) \equiv (Q_2, R_2)$ if and only if $Q_1 \equiv Q_2$ and $R_1 \equiv R_2$.

PROOF OF LEMMA 3. Suppose $Q$ and $R$ are unambiguous regular expressions such that $\sigma(Q) = \sigma(R)$. Let $Q$, $R$ be characterized in terms of $\{Q_i \mid 1 \le i \le q\}$ and $\{R_i \mid 1 \le i \le r\}$ by eqs. (2) and (3), respectively. We construct a set $X$ of pairs $(Q_u, R_v)$ such that $\sigma(Q_u) = \sigma(R_v)$. We begin with $X = \{(Q, R)\}$. We process pairs in $X$ and add new elements to $X$ until all pairs in $X$ are processed. We process a pair $(Q_u, R_v)$ as follows. By eqs. (2) and (3) we have

$$(Q_u, R_v) \equiv \left( \bigcup_{j=1}^{m} (Q_{uj}, R_{vj}) \cdot (e_j, e_j) \right) \cup (D(Q_u), D(R_v)).$$

Since $\sigma(Q_u) = \sigma(R_v)$, we have $D(Q_u) = D(R_v)$ and $\sigma(Q_{uj}) = \sigma(R_{vj})$ for $1 \le j \le m$. We add each pair $(Q_{uj}, R_{vj})$ for $1 \le j \le m$ to $X$ if it is not already present.

We obtain a set of pairs $X = \{(Q^{(1)}, R^{(1)}), \ldots, (Q^{(s)}, R^{(s)})\}$ such that $s \le qr$, $\sigma(Q^{(i)}) = \sigma(R^{(i)})$ for $1 \le i \le s$, and $(Q^{(i)}, R^{(i)}) \equiv \bigcup_{j=1}^{m} (Q_j^{(i)}, R_j^{(i)}) \cdot (e_j, e_j) \cup (D_i, D_i)$, where each pair $(Q_j^{(i)}, R_j^{(i)})$ appears in $X$.

Consider the system of equations $x_i = \sum_{j=1}^{m} a(e_j) x_{ij} + a(D_i)$, where $x_{ij} = x_k$ if $(Q_j^{(i)}, R_j^{(i)}) = (Q^{(k)}, R^{(k)})$. This system is satisfied by $x_i = a(Q^{(i)})$ if $a(Q^{(i)})$ is defined

for $1 \leq i \leq s$ and by $x_i = a(R^{(i)})$ if $a(R^{(i)})$ is defined for $1 \leq i \leq s$. We can rewrite this system as $x = Ax + b$, where each entry in $A$ is a linear combination of $a(e_1)$, $a(e_2)$, ..., $a(e_m)$, or equivalently as $(A - I)x = -b$. This system has a unique solution when the determinant of $A - I$ is nonzero, which is true if $|a(e_j)| < \epsilon$ for all $1 \leq j \leq m$, where $\epsilon$ is a sufficiently small positive constant. By Lemma 2, $\epsilon$ can be chosen sufficiently small so that if $|a(e_j)| < \epsilon$ for all $1 \leq j \leq m$, $a(Q^{(i)})$ and $a(R^{(i)})$ are defined for all $1 \leq i \leq s$. Thus $a(Q^{(i)}) = a(R^{(i)})$ for all $1 \leq i \leq s$ if $|a(e_j)| < \epsilon$ for all $1 \leq j \leq m$. In particular, $a(Q) = a(R)$ if $|a(e_j)| < \epsilon$ for all $1 \leq j \leq m$. Let $p_1$, $q_1$ and $p_2$, $q_2$ satisfy Lemma 2 for $Q$ and $R$, respectively. Then $p_1/q_1 = p_2/q_2$ if $|a(e_j)| < \epsilon$ for all $1 \leq j \leq m$; that is, $p_1q_2 = q_1p_2$. Two multivariate polynomials identical on a sphere centered at the origin are identical everywhere; thus $p_1q_2$ and $q_1p_2$ are identical polynomials, and $a(Q) = a(R)$ whenever $q_1 \neq 0$ and $q_2 \neq 0$; that is, $a(Q) = a(R)$ whenever both $a(Q)$ and $a(R)$ are defined. □

## 5. *Continuous Data Flow Problems*

Many problems in global code optimization can be formulated as path problems of the kind we are considering. The general setting is as follows. We represent a computer program by a flow graph $G = (V, E, s)$. Each vertex represents a *basic block* of the program (a block of consecutive statements having a single entry and a single exit). Each edge represents a possible transfer of control between basic blocks. The start vertex $s$ represents the start of the program. We are interested in determining, for each basic block, facts which must be true on entry to the block regardless of the actual path of program execution. Such facts can be used for various kinds of code optimization. See Aho and Ullman [2], Hecht [14], and Schaefer [26].

To represent the universe of possible program facts, we use a nonempty set $L$ having a commutative, associative, idempotent meet operation $\wedge$; such an algebraic structure is called a *lower semilattice*. If $x$ and $y$ are two possible program facts, $x \wedge y$ represents the information common to both. We can define a relation $\leq$ on $L$ by $x \leq y$ if and only if $x \wedge y = x$. The properties of $\wedge$ imply that $\leq$ is a partial order on $L$ [28]; we interpret $x \leq y$ to mean that fact $y$ contains more information than fact $x$. We assume that $L$ is *complete*, by which we mean that every nonempty subset $X \subseteq L$ has a greatest lower bound with respect to $\leq$; we denote this greatest lower bound by $\bigwedge X$. If $X = \{x_1, x_2, \ldots, x_n\}$, then $\bigwedge X = x_1 \wedge x_2 \wedge \cdots \wedge x_n$. We use $\perp$ to denote $\bigwedge L$, that is, the minimum element in $L$. For any functions $f$ and $g$ having common domain and range $L$, we define $f \leq g$ if and only if $f(x) \leq g(x)$ for all elements $x$ in the domain of $f$ and $g$.

To represent the effect of the program on the universe of facts, we associate with each edge $e$ a function $f_e$ such that if fact $x$ is true on entry to $h(e)$ and control passes through edge $e$, then $f_e(x)$ will be true on entry to $t(e)$. We can extend these functions to paths by defining $f_p(x) = x$ if $p$ is the empty path, $f_p(x) = (f_{e_k} \circ f_{e_{k-1}} \circ \cdots \circ f_{e_1})(x)$ if $p = e_1, e_2, \ldots, e_k$. What we want to compute is $\bigwedge \{f_p(\perp) \mid p \text{ is a path from } s \text{ to } v\}$ for each vertex $v$. (We assume that the minimum fact $\perp$ is true on entry to the program.) This discussion motivates the following definitions.

A *continuous data flow framework* $(L, F)$ is a complete lower semilattice $L$ with meet operation $\wedge$ and a set of functions $F: L \to L$ satisfying the following axioms:

(7a) (identity) $F$ contains the identity function $\iota$.

(7b) (closure) $F$ is closed under meet, function composition, and $*$, where $(f \wedge g)(x) = f(x) \wedge g(x)$ and $f^*(x) = \bigwedge \{f^i(x) \mid i \geq 0\}$.

(7c) (continuity) For every $f \in F$ and nonempty $X \subseteq L$, $f(\bigwedge X) = \bigwedge \{f(x) \mid x \in X\}$.

A *continuous data flow problem* consists of a flow graph $G = (V, E, s)$, a continuous data flow framework $(L, F)$, and a mapping from $E$ to $F$; we use $f_e$ to denote the function associated with edge $e$. The *meet over all paths (MOP) solution* to this problem is the mapping *mop* from $V$ to $L$ given by $mop(v) = \wedge \{f_p(\perp) \mid p \text{ is a path from } s \text{ to } v\}$.

We can use path expressions to solve continuous data flow problems by means of the mapping $f$ defined as follows.

(8a) $f(\Lambda) = \iota$;
$f(e) = f_e$.
(8b) $f(P_1 \cup P_2) = f(P_1) \wedge f(P_2)$;
$f(P_1 \cdot P_2) = f(P_2) \circ f(P_1)$;
$f(P_1^*) = f(P_1)^*$.

LEMMA 5. *Let $P \neq \emptyset$ be a path expression of type $(v, w)$. Then for all $x \in L$, $f(P)(x) = \wedge \{f_p(x) \mid p \in \sigma(P)\}$.*

PROOF. By induction on the number of operation symbols in $P$. The lemma is immediate if $P$ is atomic. Suppose the lemma is true for path expressions containing fewer than $k$ operation symbols, and let $P$ contain $k$ operation symbols. We have three cases.

Case 1. Suppose $P = P_1 \cup P_2$. Then

$$\begin{aligned} f(P)(x) &= f(P_1)(x) \wedge f(P_2)(x) \\ &= (\wedge \{f_p(x) \mid p \in \sigma(P_1)\}) \wedge (\wedge \{f_p(x) \mid p \in \sigma(P_2)\}) \\ &= \wedge \{f_p(x) \mid x \in \sigma(P_1) \cup \sigma(P_2)\} = \wedge \{f_p(x) \mid p \in \sigma(P)\}. \end{aligned}$$

Case 2. Suppose $P = P_1 \cdot P_2$. Then

$$\begin{aligned} f(P)(x) &= f(P_2)(f(P_1)(x)) = f(P_2)(\wedge \{f_{p_1}(x) \mid p_1 \in \sigma(P_1)\}) \\ &= \wedge \{f(P_2)(f_{p_1}(x)) \mid p_1 \in \sigma(P_1)\} \qquad \text{by continuity} \\ &= \wedge \{\wedge \{f_{p_1 p_2}(x) \mid p_2 \in \sigma(P_2)\} \mid p_1 \in \sigma(P_1)\} \\ &= \wedge \{f_{p_1 p_2}(x) \mid p_1 \in \sigma(P_1) \text{ and } p_2 \in \sigma(P_2)\} = \wedge \{f_p(x) \mid p \in \sigma(P)\}. \end{aligned}$$

Case 3. As in case 2 we can show that if $P_1$ has fewer than $k$ operation symbols, then $f(P_1)^i(x) = \wedge \{f_p(x) \mid p \in \sigma(P_1)^i\}$ for any $i \geq 0$. Suppose $P = P_1^*$. Then

$$\begin{aligned} f(P)(x) &= f(P_1)^*(x) = \wedge \{f(P_1)^i(x) \mid i \geq 0\} \\ &= \wedge \{\wedge \{f_p(x) \mid p \in \sigma(P_1)^i\} \mid i \geq 0\} \\ &= \wedge \{f_p(x) \mid p \in \sigma(P_1^*)\}. \end{aligned}$$

□

THEOREM 5. *For any vertex $v$ let $P(s,v)$ be a path expression representing all paths from $s$ to $v$. Then $mop(v) = f(P(s,v))(\perp)$.*

Thus we can use a solution to the single-source path expression problem to solve continuous data flow problems. For examples and extensive discussions of such problems see Cousot and Cousot [5], Fong, Kam, and Ullman [9], Graham and Wegman [13], Kam and Ullman [17, 18], Kildall [20], and Rosen [24]. Rosen's *flow cover* approach to data flow problems resembles the approach taken here.

## 6. *Monotone Data Flow Problems*

Many important global flow problems are not continuous [18]. For such problems there is in general no algorithm to compute the meet-over-all-paths solution [18], and we must be satisfied with less information than the MOP solution provides. In such

situations the following approach, essentially that of Graham and Wegman [13] and Rosen [24], is appropriate.

A *monotone data flow framework* $(L, F)$ is a complete lower semilattice $L$ with meet operation $\wedge$ and a set of functions $F: L \to L$ satisfying the following axioms. As in Section 5, we define the meet $f \wedge g$ of two functions $f$ and $g$ by $(f \wedge g)(x) = f(x) \wedge g(x)$, and the reflexive, transitive closure $f^*$ of a function $f$ by $f^*(x) = \wedge \{f^i(x) \mid i \geq 0\}$.

(9a) (identity) $F$ contains the identity function $\iota$.
(9b) (closure) $F$ is closed under meet and function composition.
(9c) (monotonicity) For every $f \in F$ and $x, y \in L$, $x \leq y$ implies $f(x) \leq f(y)$.
(9d) (approximation to $f^*$) For every function $f \in F$, there is a function $f^{@} \in F$ such that

(i) $f^{@}(x) \leq f^i(x)$ for all $x \in L$, $i \geq 0$; and
(ii) if $x \in L$ satisfies $f(x) \geq x$, then $f^{@}(x) \geq x$.

Monotone frameworks generalize continuous frameworks by requiring only monotonicity (9c) in place of continuity (7c) and by requiring only a pseudotransitive closure function. Note that $f^*$ is the maximum function satisfying (9d), although we do *not* require that $f^* \in F$.

A *monotone data flow problem* consists of a flow graph $G = (V, E, s)$, a monotone data flow framework $(L, F)$, and a mapping from $E$ to $F$ whose values we denote by $f_e$ for $e \in E$. A *fixed point* for this problem is a mapping $Z: V \to L$ such that

$$z(s) = \bot \quad \text{and} \quad f_e(z(h(e))) \geq z(t(e)) \quad \text{for any} \quad e \in E. \tag{10}$$

A *safe solution* to the data flow problem is a mapping $x: V \to L$ such that

(10a) $x(v) \leq f_p(\bot)$ for any vertex $v$ and any path $p$ from $s$ to $v$; and
(10b) $x(v) \geq z(v)$ for any fixed point $z$ and any vertex $v$.

Thus a safe solution is a conservative approximation to the MOP solution which is at least as informative as any fixed point. It is easy to prove that any fixed point satisfies (10a); if the data flow problem is continuous, the MOP solution is the maximum fixed point [20].

We can use a slight variant of the mapping defined in Section 4 to compute a safe solution to a monotone data flow problem. Let $f$ be defined as in (8a) and (8b), except that $f(P_1^*) = f(P_1)^{@}$.

LEMMA 6. *Let $P \neq \emptyset$ be a path expression of type $(v, w)$. Then $f(P)(x) \leq f_p(x)$ for all $p \in S(P)$ and $x \in L$.*

PROOF. By induction on the number of operation symbols in $P$. The lemma is immediate if $P$ is atomic. Suppose the lemma is true for path expressions containing fewer than $k$ operation symbols, and let $p$ contain $k$ operation symbols. We have three cases.

Case 1. Suppose $P = P_1 \cup P_2$ and $p \in P$. If $p \in P_1$, then

$$f(P)(x) = f(P_1)(x) \wedge f(P_2)(x) \leq f(P_1)(x) \leq f_p(x)$$

by the induction hypothesis; similarly if $p \in P_2$.

Case 2. Suppose $P = P_1 \cdot P_2$ and $p = p_1 p_2$, with $p_1 \in P_1$, $p_2 \in P_2$. Then

$$\begin{aligned} f(P)(x) &= f(P_2)(f(P_1)(x)) \leq f(P_2)(f_{p_1}(x)) \\ &\leq (f_{p_2} \circ f_{p_1})(x) = f_p(x) \end{aligned}$$

by monotonicity and the induction hypothesis.

Case 3. Suppose $P = P_1^*$ and $p = p_1 p_2 \cdots p_k$, with $p_i \in P_1$ for $1 \le i \le k$. Then

$$\begin{aligned} f(P)(x) = f(P_1)^{@}(x) &\le f(P_1)^k(x) && \text{by (9d(i))} \\ &\le f_p(x) && \text{by monotonicity and the induction hypothesis, as above.} \end{aligned}$$ □

LEMMA 7. *Let $P \neq \emptyset$ be a path expression of type $(v, w)$. If $z$ is any fixed point, then $f(P)(z(v)) \ge z(w)$.*

PROOF. By induction. The lemma is immediate if $P$ is atomic. Suppose the lemma is true for path expressions containing fewer than $k$ operation symbols, and let P contain $k$ operation symbols. We have the usual three cases.

Case 1. Suppose $P = P_1 \cup P_2$. then $f(P)(z(v)) = f(P_1)(z(v)) \wedge f(P_2)(z(v)) \ge z(w)$ by the induction hypothesis.

Case 2. Suppose $P = P_1 \cdot P_2$. Let $u$ be the vertex such that $P_1$ is of type $(v, u)$ and $P_2$ is of type $(u, w)$. Then $f(P)(z(v)) = f(P_2)(f(P_1)(z(v))) \ge f(P_2)(z(u)) \ge z(w)$ by the induction hypothesis.

Case 3. Suppose $P = P_1^*$. By the induction hypothesis, $f(P_1)(z(v)) \ge z(v)$. By (9d(ii)), $f(P)(z(v)) = f(P_1)^{@}(z(v)) \ge z(v)$. □

THEOREM 6. *For each vertex $v$ let $P(s, v)$ be a path expression representing all paths from $s$ to $v$. Then the function $x: V \to L$ defined by $x(v) = f(P(s, v))(\perp)$ is a safe solution.*

PROOF. By Lemma 6, $x(v) = f(P(s, v))(\perp) \le f_p(\perp)$ for all $p \in S(P(s, v))$; thus $x$ satisfies (10a). Let $z$ be any fixed point. By Lemma 7, $x(v) = f(P(s, v))(\perp) = f(P(s, v))(z(s)) \ge z(v)$; thus $x$ satisfies (10b). □

## 7. *Bounded Data Flow Problems*

Most interesting data flow problems satisfy a stronger condition on $L$ than completeness, called the *descending chain condition*; every descending chain $x_1 > x_2 > x_3 > \cdots$ in $L$ is finite. For semilattices satisfying the descending chain condition, continuity is equivalent to *distributivity*: $f(x \wedge y) = f(x) \wedge f(y)$ for all $f \in F$ and $x, y \in L$. Our continuous data flow problems are thus a generalization of the distributive data flow problems considered by Kildall [20]. Although most global flow problems satisfy the descending chain condition, some, such as type checking [35], do not.

If the set of functions $F$ in a data flow framework satisfies a boundedness condition, then we can compute an approximation $f^{@}$ to $f^*$ for any function $f \in F$ using only function meet and composition. If the framework is continuous as well, it is possible to compute the MOP solution from a set of path expressions representing only some of the paths from the start vertex. We consider a hierarchy of boundedness axioms. For $k \ge 1$, a *$k$-bounded data flow framework* $(L, F)$ is a complete lower semilattice $L$ with meet operation $\wedge$ and a set of functions $F: L \to L$ satisfying identity (9a), closure (9b), monotonicity (9c), and

(9e) ($k$-boundedness) $f^k(x) \ge \wedge \{f^i(x) \mid 0 \le i \le k-1\}$ for all $f \in F$ and $x \in L$.

For $k \ge 1$, a *$k$-semibounded data flow framework* $(L, F)$ is a complete lower semilattice $L$ with meet operation $\wedge$ and a set of functions $F: L \to L$ satisfying (9a)–(9c), and

(9f) ($k$-semiboundedness) $f^k(x) \geq (\wedge \{f^i(x) \mid 0 \leq i \leq k-1\}) \wedge f^k(y)$ for all $f \in F$ and $x, y \in L$.

We define $k$-bounded and $k$-semibounded data flow problems in the obvious way. It is easy to show that $k$-boundedness implies $k$-semiboundedness and $k$-semiboundedness implies $(k+1)$-boundedness. Boundedness, being a property of $F$ and not of $L$, is neither stronger nor weaker than the descending chain condition. The $k$-bounded and $k$-semibounded data flow problems include some, but not all, of the global flow problems mentioned in the literature. Problems that use bit vectors, such as finding available expressions [32] and finding live variables [19], are 1-semibounded but not 1-bounded. Problems that use "structured partition lattices," such as common subexpression detection [9, 17, 20], are 2-bounded but not 1-semibounded. Type checking [35] is not $k$-bounded unless some bound is artificially imposed.

LEMMA 8. *In a $k$-bounded data flow framework $(L, F)$, $f^* = \wedge \{f^i \mid 0 \leq i \leq k-1\}$ for all $f \in F$.*

PROOF. We prove by induction on $j$ that if $j \geq k$, $f^j(x) \geq \wedge \{f^i(x) \mid 0 \leq i \leq k-1\}$ for all $f \in F$ and $x \in L$. The claim is true for $j = k$ by $k$-boundedness. Suppose $j > k$ and the claim is true for $j-1$. Then

$$\begin{aligned} f^j(x) = f^{j-1}(f(x)) &\geq \wedge \{f^i(x) \mid 1 \leq i \leq k\} && \text{by the induction hypothesis} \\ &\geq \wedge \{f^i(x) \mid 0 \leq i \leq k-1\} && \text{by } k\text{-boundedness.} \end{aligned}$$

The lemma follows from the claim. □

LEMMA 9. *In a $k$-bounded data flow framework $(L, F)$, the function $f^@$ defined by $f^@ = (f \wedge \iota)^{k-1}$ for $f \in F$ satisfies (9d).*

PROOF. By repeated use of monotonicity, we obtain $f^@(x) = (f \wedge \iota)^{k-1}(x) \leq \wedge \{f^i(x) \mid 0 \leq i \leq k-1\}$, which implies (9d(i)) by Lemma 9. We prove by induction on $j$ that if $f(x) \geq x$, then $(f \wedge \iota)^j(x) \geq x$. The result is immediate for $j = 0$. Suppose $(f \wedge \iota)^{j-1}(x) \geq x$. Then $(f \wedge \iota)^j(x) \geq f(x) \wedge x \geq x$. Thus $f(x) \geq x$ implies $f^@(x) = (f \wedge \iota)^{k-1}(x) \geq x$, and (9d(ii)) holds. □

If $(L, F)$ is a $k$-bounded data flow framework and $f \in F$, we can compute $f^*$ using $O(k)$ function meets and compositions by Lemma 8. We can compute an approximation $f^@$ to $f^*$ in $O(\log k)$ function meets and compositions by Lemma 9. (We trade accuracy for time if we compute $f^@$ instead of $f^*$.) Theorem 6 thus gives a method for solving bounded data flow problems using only function meet, composition, and application.

Suppose $(L, F, G, f_e)$ is a data flow problem which is not only bounded but continuous. In this case $f^@ = f^*$, and we can compute the MOP solution using only function meet, composition, and application, with $O(\log k)$ such operations replacing each $*$. We can also use path expressions representing only some of the paths from $s$, as demonstrated by the following results.

LEMMA 10. *Let $(L, F, G, f_e)$ be a $k$-bounded continuous data flow problem. Let $v$ be a vertex in $G$ and $p$ be a path from $s$ to $v$ that is not $k$-simple. Then there is a set $S$ of paths from $s$ to $v$ such that each path in $S$ is shorter than $p$ and $f_p \geq \wedge \{f_q \mid q \in S\}$.*

PROOF. If $p$ is not $k$-simple, then $p$ contains some vertex $u$ at least $k+1$ times. Let $p = p_0 p_1 p_2 \cdots p_k p_{k+1}$, where each $p_i$ for $1 \leq i \leq k$ is a cycle from $u$ to $u$. (Both $p_0$ and $p_{k+1}$ may be the empty path.) Then

$$
\begin{aligned}
f_p &\geq f_{p_{k+1}} \circ (\wedge \{f_{p_i} \mid 1 \leq i \leq k\})^k \circ f_{p_0} && \text{by continuity} \\
&\geq f_{p_{k+1}} \circ \wedge \{(\wedge \{f_{p_i} \mid 1 \leq i \leq k\})^j \mid 0 \leq j \leq k-1\} \circ f_{p_0} && \text{by } k\text{-boundedness} \\
&\geq \wedge \{f_q \mid q = p_0 q_1 q_2 \cdots q_l p_{k+1} \text{ where } 0 \leq l \leq k-1 \\
&\qquad \text{and } q_j \in \{p_i \mid 1 \leq i \leq k\} \text{ for } 1 \leq j \leq l\}. && \square
\end{aligned}
$$

COROLLARY 1. *Let $(L, F, G, f_e)$ be a $k$-bounded continuous data flow problem. Let $v$ be a vertex in $G$ and $p$ be a path from $s$ to $v$. Then $f_p \geq \wedge \{f_q \mid q$ is a $k$-simple path from $s$ to $v\}$.*

PROOF. By induction on the length of $p$ using Lemma 10. □

THEOREM 7. *Let $(L, F, G, f_e)$ be a $k$-bounded continuous data flow problem. For each vertex $v$ let $P_k(s, v)$ be a path expression such that $\sigma(P_k(s, v))$ contains at least all the $k$-simple paths from $s$ to $v$. Then $mop(v) = f(P_k(s, v))(\perp)$, where $f$ is defined as in Section 5.*

PROOF. Immediate from Lemma 5 and Corollary 1. □

LEMMA 11. *Let $(L, F, G, f_e)$ be a $k$-semibounded continuous data flow problem. Let $v$ be a vertex in $G$ and $p$ be a path from $s$ to $v$ which is not $k$-semisimple. Then there is a set $S$ of paths from $s$ to $v$ such that each path in $S$ is shorter than $p$ and $f_p \geq \wedge \{f_q \mid q \in S\}$.*

PROOF. If $p$ is not $k$-semisimple, then $p$ can be partitioned into $p = p_0 p_1 p_2 p_3 \cdots p_{k+2} p_{k+3}$, where $p_1$ and $p_i$ for $3 \leq i \leq k+2$ are cycles and $p_0$, $p_2$, $p_{k+3}$ are possibly empty. Then

$$
\begin{aligned}
f_p &\geq f_{p_{k+3}} \circ (\wedge \{f_{p_i} \mid 3 \leq i \leq k+2\})^k \circ f_{p_0 p_1 p_2} && \text{by continuity} \\
&\geq f_{p_{k+3}} \circ \wedge \{(\wedge \{f_{p_i} \mid 3 \leq i \leq k+2\})^j \mid 0 \leq j \leq k-1\} \circ f_{p_0 p_1 p_2} \\
&\quad \wedge f_{p_{k+3}} \circ (\wedge \{f_{p_i} \mid 3 \leq i \leq k+2\})^k \circ f_{p_0 p_2} \\
&\qquad && \text{by } k\text{-semiboundedness and continuity} \\
&\geq (\wedge \{f_q \mid q = p_0 p_1 p_2 q_1 q_2 \cdots q_l p_{k+3} \text{ where } 0 \leq l \leq k-1 \\
&\qquad \text{and } q_j \in \{p_i \mid 3 \leq i \leq k+2\} \text{ for } 1 \leq j \leq l\}) \\
&\qquad \wedge (\wedge \{f_q \mid q = p_0 p_2 q_1 q_2 \cdots q_k p_{k+3} \text{ where } q_j \in \{p_i \mid 3 \leq i \leq k+2\} \\
&\qquad\quad \text{for } 1 \leq j \leq k\}). && \square
\end{aligned}
$$

COROLLARY 2. *Let $(L, F, G, f_e)$ be a $k$-semibounded continuous data flow problem. Let $v$ be a vertex in $G$ and $p$ be a path from $s$ to $v$. Then $f_p \geq \wedge \{f_q \mid q$ is a $k$-semisimple path from $s$ to $v\}$.*

PROOF. By induction on the length of $p$ using Lemma 11. □

THEOREM 8. *Let $(L, F, G, f_e)$ be a $k$-semibounded continuous data flow problem. For each vertex $v$, let $P'_k(s, v)$ be a path expression such that $S(P'_k(s, v))$ contains at least all the $k$-semisimple paths from $s$ to $v$. Then $mop(v) = f(P'_k(s, v))(\perp)$, where $f$ is defined as in Section 5.*

PROOF. Immediate from Lemma 5 and Corollary 2. □

Corollaries 1 and 2 require continuity; in fact, the MOP solution is not effectively computable in a general 2-bounded monotone data flow problem [18]. See Kam and Ullman [17] and Tarjan [30] for further discussion of the effect of boundedness on global flow analysis.

## 8. *An Idiosyncratic Data Flow Problem*

As a final application of our technique we consider a data flow problem that does not fit naturally into the semilattice framework but that can still be solved easily

using a mapping from path expressions. The problem arises in the optimization of very-high-level languages and has been studied by Fong [8].

Let $G = (V, E, s)$ be the flow graph of a program which contains occurrences of an expression $\mathscr{E}$. With each edge $e$ of the program is associated an *effect*, which has one of four values depending upon what flow of control through edge $e$ does to the value of $\mathscr{E}$.

$$effect(e) = \left\{ \begin{array}{l} gen \\ kill \\ injure \\ trans \end{array} \right\} \quad \text{if} \quad \left\{ \begin{array}{l} \text{the program recomputes } \mathscr{E}, \\ \text{the program makes a large change in the value of } \mathscr{E}, \\ \text{the program makes a small change in the value of } \mathscr{E}, \\ \text{the program does not affect the current value of } \mathscr{E}. \end{array} \right.$$

For any vertex $v$ we say $\mathscr{E}$ is *implicitly available* on entry to $v$ if there is a positive bound $b$ such that, for every path $p = e_1, e_2, \ldots, e_k$ from $s$ to $v$, there is an $i$ such that (i) $effect(e_i) = gen$, (ii) $effect(e_j) \neq kill$ for $i < j \leq k$, and (iii) the number of values $j$ such that $i < j \leq k$ and $effect(e_j) = injure$ is bounded by $b$. Note that the bound $b$ can depend upon the vertex $v$ but not upon the path $p$.

The problem we wish to solve is to determine from $\{effect(e) \mid e \in \mathscr{E}\}$ the vertices at which $\mathscr{E}$ is implicitly available. The idea is that if the most recently computed value of $\mathscr{E}$ can be injured only a bounded number of times before entering $v$, we can compute the value on entry to $v$ from the most recently computed value by performing a bounded number of updates. Otherwise we must completely recompute $\mathscr{E}$ to obtain its value on entry to $v$.

Fong [8] claims that this problem cannot be formulated within the semilattice framework, "At least in the only natural choice of semilattice." However, Fong observes that the problem can still be solved efficiently. We shall define a mapping from path expressions for this purpose.

Let $D = \{g, t_0, t_+, \omega\}$ be a set having operations $\wedge$, $\circ$, @ defined by the following tables:

| $\wedge$ | $g$ | $t_0$ | $t_+$ | $\omega$ |
|---|---|---|---|---|
| $g$ | $g$ | $t_0$ | $t_+$ | $\omega$ |
| $t_0$ | $t_0$ | $t_0$ | $t_+$ | $\omega$ |
| $t_+$ | $t_+$ | $t_+$ | $t_+$ | $\omega$ |
| $\omega$ | $\omega$ | $\omega$ | $\omega$ | $\omega$ |

| $\circ$ | $g$ | $t_0$ | $t_+$ | $\omega$ |
|---|---|---|---|---|
| $g$ | $g$ | $g$ | $g$ | $\omega$ |
| $t_0$ | $g$ | $t_0$ | $t_+$ | $\omega$ |
| $t_+$ | $g$ | $t_+$ | $t_+$ | $\omega$ |
| $\omega$ | $g$ | $\omega$ | $\omega$ | $\omega$ |

| | @ |
|---|---|
| $g$ | $t_0$ |
| $t_0$ | $t_0$ |
| $t_+$ | $\omega$ |
| $\omega$ | $\omega$ |

Let the mapping $f$ from path expressions to $D$ be defined as follows.

(11a) $f(\Lambda) = t_0$,

$$f(e) = \left\{ \begin{array}{l} g \\ \omega \\ t_+ \\ t_0 \end{array} \right\} \text{ if } effect(e) = \left\{ \begin{array}{l} gen \\ kill \\ injure \\ trans \end{array} \right\} \text{ for } e \in E;$$

(11b) $f(P_1 \cup P_2) = f(P_1) \wedge f(P_2)$,
$f(P_1 \circ P_2) = f(P_1) \circ f(P_2)$,
$f(P_1^*) = f(P_1)^{@}$.

We call a path $p = e_1, e_2, \ldots, e_k$ in $G$ a $t_i$-*path* if $effect(e_j) \in \{injure, trans\}$ for $1 \leq j \leq k$ and the number of edges $e_j$ such that $effect(e_j) = injure$ is $i$. We call a path $p$ a $g_i$-*path* if it can be partitioned into $p = p_1, e, p_2$, where $effect(e) = gen$ and $p_2$ is a $t_i$-path. We call a path $p$ an $\omega$-*path* if it can be partitioned into $p = p_1, e, p_2$, where $effect(e) = kill$ and $p$ is a $t_i$-path for some $i$.

LEMMA 12. *Let $P$ be a path expression. Then*

(*i*) $f(P) = g$ *if there is a bound* $b$ *such that every path in* $\sigma(P)$ *is a* $g_i$*-path with* $i \leq b$;
(*ii*) $f(P) = t_0$ *if there is a bound* $b$ *such that every path in* $\sigma(P)$ *is either a* $g_i$*-path with* $i \leq b$ *or a* $t_0$*-path and* $\sigma(P)$ *contains at least one* $t_0$ *path*;
(*iii*) $f(P) = t_+$ *if there is a bound* $b$ *such that every path in* $\sigma(P)$ *is either a* $g_i$*-path with* $i \leq b$ *or a* $t_i$*-path with* $i \leq b$ *and* $\sigma(P)$ *contains at least one* $t_i$*-path with* $i > 0$;
(*iv*) $f(P) = \omega$ *in all other cases. (For any bound* $b$, $\sigma(P)$ *contains either a* $g_i$*-path with* $i > b$, *a* $t_i$*-path with* $i > b$, *or an* $\omega$*-path.)*

PROOF. Straightforward but tedious, by induction on the number of operation symbols in $P$. □

THEOREM 9. *For each vertex* $v$ *in* $G$ *let* $P(s, v)$ *be a path expression representing all paths from* $s$ *to* $v$ *in* $G$. *Then* $\mathscr{E}$ *is implicitly available at* $v$ *if and only if* $f(P(s, v)) = g$.

PROOF. Immediate from Lemma 12. □

Actual occurrences of the implicit availability problem usually involve a number of expressions. We can perform the computation associated with Theorem 9 in parallel for all the expressions by using bit vector operations. Since $D$ contains four elements, we need two bit vectors for each value computed (rather than the three proposed by Fong [8]). By adding an additional element to $D$ we can compute the explicitly available expressions (those available with no injuries) in addition to the implicitly available ones.

## 9. *Remarks*

We have shown how to use path expressions to solve three kinds of path problems on directed graphs. Our results allow us to build a general algorithm for solving path problems on directed graphs; to solve a particular path problem, we merely interpret $\cup$, $\cdot$, and $*$ appropriately. We can base such an algorithm on Gaussian or Gauss–Jordan elimination [22]. Tarjan [31] discusses a decomposition algorithm which is especially efficient on reducible and almost reducible graphs [15, 29].

Our results serve to formally justify the empirical observation that the same algorithms work on many different path problems. There are of course algorithms that solve only a particular kind of path problem, such as Dijkstra's [6], Fredman's [12], Johnson's [16], and Wagner's [33] shortest path algorithms and Pan's improvement to Strassen's algorithm for solving linear equations [4, 23, 27]. However, any algorithm able to compute path expressions also solves all the path problems we have considered here.

Our ideas extend easily to matrix multiplication problems and to problems requiring the transitive closure of a matrix. See Aho, Hopcroft, and Ullman [1] and Lehmann [22] for discussions of such problems.

## *Appendix. Graph-Theoretic Definitions*

A *directed graph* $G = (V, E)$ is a finite set $V$ of *vertices* and a finite set $E$ of *edges* such that each edge $e$ has a *head* $h(e) \in V$ and a *tail* $t(e) \in V$. We regard the edge $e$ as leading from $h(e)$ to $t(e)$. A *path* $p = e_1, e_2, \ldots, e_k$ is a sequence of edges such that $t(e_i) = h(e_{i+1})$ for $1 \leq i \leq k - 1$. The path is *from* $h(e_1)$ *to* $t(e_k)$. The path *contains* edges $e_1, e_2, \ldots, e_k$ and vertices $h(e_1), h(e_2), \ldots, h(e_k), t(e_k)$, and *avoids* all other edges and vertices. There is a path of no edges from any vertex to itself. A *cycle* is a nonempty path from a vertex to itself.

If there is a path from a vertex $v$ to a vertex $w$, then $w$ is *reachable* from $v$. A *flow graph* $G = (V, E, s)$ is a graph containing a distinguished *start vertex* $s$ such that every vertex is reachable from $s$.

A *simple path p* is a path containing no vertex twice. For $k \geq 1$, a *k-simple path* is a path containing no vertex $k + 1$ times. Thus a 1-simple path is simple. A *k-semisimple path* is a path $p$ that can be partitioned as $p = p_1, e, p_2$, where $p_1$ is simple, $e$ is an edge, and $p_2$ is $k$-simple.

ACKNOWLEDGMENTS. My thanks to Fran Berman for extensive and illuminating discussions of these ideas, and to Roland Backhouse and Barry Rosen for their valuable comments on the manuscript.

REFERENCES

1. AHO, A.V., HOPCROFT, J.E., AND ULLMAN, J D. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, Mass., 1974, pp 195–206.
2. AHO, A V , AND ULLMAN, J.D. *Principles of Compiler Design* Addison-Wesley, Reading, Mass , 1977, pp 408–517.
3. BACKHOUSE, R.C., AND CARRÉ, B.A. Regular algebra applied to path-finding problems. *J. Inst. Math. Applic. 15* (1975), 161–186.
4 BUNCH, J.R., AND HOPCROFT, J E Triangular factorization and inversion by fast matrix multiplication. *Math. Comput. 28* (1974), 231–236.
5. COUSOT, P., AND COUSOT, R. Abstract interpretation. A unified lattice model for static analysis of programs by construction or approximation of fixpoints Conf. Rec. 4th ACM Symp. on Principles of Programming Languages, Los Angeles, Calif., 1977, pp 238–252.
6. DIJKSTRA, E W A note on two problems in connexion with graphs *Numer Math. 1* (1959), 269–271
7. FLOYD, R. Algorithm 97. Shortest path. *Commun. ACM 5*, 6 (June 1962), 345.
8. FONG, A C Generalized common subexpressions in very high level languages Conf. Rec 4th ACM Symp on Principles of Programming Languages, Los Angeles, Calif , 1977, pp 48–57.
9. FONG, A.C., KAM, J B., AND ULLMAN, J.D. Applications of lattice algebra to loop optimization Conf. Rec. 2nd ACM Symp on Principles of Programming Languages, Palo Alto, Calif , 1975, pp 1–9
10. FORD, L.R. JR., AND FULKERSON, D R. *Flows in Networks.* Princeton University Press, Princeton, N J., 1962, pp 130–134
11 FORSYTHE, G.E , AND MOLER, C.B. *Computer Solution of Linear Algebraic Equations.* Prentice-Hall, Englewood Cliffs, N.J., 1967, pp 27–36.
12 FREDMAN, M.L New bounds on the complexity of the shortest path problem. *SIAM J. Comput. 5* (1976), 83–89
13. GRAHAM, S L., AND WEGMAN, M A fast and usually linear algorithm for global flow analysis. *J. ACM 23*, 1 (Jan 1976), 172–202
14 HECHT, M.S. *Flow Analysis of Computer Programs.* Elsevier, New York, 1977
15 HECHT, M.S , AND ULLMAN, J.D. Flow graph reducibility *SIAM J. Comput. 1* (1972), 188–202
16. JOHNSON, D.B Efficient algorithms for shortest paths in sparse networks *J. ACM 24*, 1 (Jan. 1977), 1–13.
17. KAM, J B , AND ULLMAN, J D. Global data flow analysis and iterative algorithms. *J ACM 23*, 1 (Jan. 1976), 158–171
18 KAM, J.B., AND ULLMAN, J D. Monotone data flow analysis frameworks *Acta Inf 7* (1977), 305–317.
19. KENNEDY, K W Node listings applied to data flow analysis. Conf. Rec 2nd ACM Symp. on Principles of Programming Languages, Palo Alto, Calif , 1975, pp. 10–21
20 KILDALL, G A A unified approach to program optimization Conf. Rec ACM Symp on Principles of Programming Languages, Boston, Mass., 1973, pp 10–21.
21 KLEENE, S C. Representation of events in nerve nets and finite automata. In *Automata Studies,* C. E Shannon and J. McCarthy, Eds., Princeton University Press, Princeton, N J , 1956, pp. 3–40.
22 LEHMANN, D J Algebraic structures for transitive closure. *Theor. Comput Sci 4* (1977), 59–76.
23 PAN, V.Y Strassen's algorithm is not optimal. Trilinear technique of aggregating uniting and canceling for constructing fast algorithms for matrix operations. Proc. 19th Ann IEEE Symp on Foundations of Computer Science, Ann Arbor, Mich , 1978, pp 166–176.
24 ROSEN, B K. Monoids for rapid data flow analysis. *SIAM J Comput. 9* (1980), 159–196
25. SALOMAA, A. Two complete axiom systems for the algebra of regular events. *J. ACM*, 1 (Jan 1966), 158–169.
26 SCHAEFER, M *A Mathematical Theory of Global Program Optimization.* Prentice-Hall, Englewood Cliffs, N J , 1973

27. STRASSEN, V. Gaussian elimination is not optimal *Numer. Math 13* (1969), 354–356.
28. SZÁSZ, G *Introduction to Lattice Theory*, B. Balkay and G. Tóth, Trans., Academic Press, New York, 1963, pp 38–42, 59–60
29 TARJAN, R E. Testing flow graph reducibility *J. Comput Syst Sci. 9* (1974), 355–365
30 TARJAN, R.E Iterative algorithms for global flow analysis In *Algorithms and Complexity, New Directions and Recent Results*, J F Traub, Ed., Academic Press, New York, 1976, pp 91–102.
31. TARJAN, R E Fast algorithms for solving path problems, *J. ACM 28*, 3 (July 1981), 594–614
32 ULLMAN, J D Fast algorithms for the elimination of common subexpressions *Acta Inf. 2* (1973), 191–213
33. WAGNER, R.A A shortest path algorithm for edge-sparse graphs. *J ACM 23*, 1 (Jan 1976), 50–57.
34. WARSHALL, S A theorem on Boolean matrices. *J ACM 9*, 1 (Jan. 1962), 11–12.
35. WEGBREIT, B Property extraction in well-founded property sets. *IEEE Trans. Softw Eng. 1* (1975), 270–285.

RECEIVED APRIL 1979; REVISED MAY 1980, ACCEPTED JUNE 1980