Acta Informatica 7, 305—317 (1977)

Monotone Data Flow Analysis Frameworks*

John B. Kam and Jeffrey D. Ullman

Received March 24, 1975

Summary. We consider a generalization of Kildall's lattice theoretic approach to data flow analysis, which we call *monotone data flow analysis frameworks*. Many flow analysis problems which appear in practice meet the monotonicity condition but not Kildall's condition called *distributivity*. We show that the maximal fixed point solution exists for every instance of every monotone framework, and that it can be obtained by Kildall's algorithm. However, whenever the framework is monotone but not distributive, there are instances in which the desired solution—the "meet over all paths solution"—differs from the maximal fixed point. Finally, we show the nonexistence of an algorithm to compute the meet over all paths solution for monotone frameworks.

1 Introduction

Performing compile time optimization requires solving a class of problems, called global data flow analysis problems (abbreviated as gdfap's), involving determination of information which is distributed throughout the program.

Thus far, work has been done only for a restricted subclass of gdfap's for which the meet over all paths solution[1] to individual programs can be obtained efficiently by using interval analysis [1–5, 8, 12] or by an iterative approach [6, 9, 10, 14, 16]. In these gdfap's, called *distributive* gdfap's, the MOP solution can be characterized as a maximum fixed point solution to a set of simultaneous equations.

In this paper, a more general class of gdfap's called *monotone data flow analysis frameworks* (abbreviated as *framework*), will be examined. We first illustrate several problems not belonging to the restricted class of distributive gdfap's. The paper also shows that for monotone frameworks, the MOP solution for an individual program does not necessarily coincide with the maximum fixed point solution to the corresponding set of simultaneous equations. Several methods for approaching this class of frameworks will be discussed. We conclude the paper by showing that there exists no algorithm which, when given an arbitrary monotone framework, will compute the MOP for each program.

2. Background

We assume the reader has some familiarity with the lattice theoretic formulation of data flow analysis, as discussed in [9, 10, 15], for example. We refer to these papers for the proper motivation for the subject to be discussed here.

* Work supported by NSF grant GJ-1052.

1 Given a gdfap, *the meet over all paths (MOP)* solution for a program can be interpreted informally as the calculation for each statement in the program of the maximum information, relevant to the gdfap, which is true along every possible execution path from the starting point of the program to that particular statement.

Definition: A *flow graph* is a triple $G=(N, E, n_0)$, where:

(1) N is a finite set of *nodes*.

(2) E is a subset of $N \times N$ called the *edges*. The edge (x, y) *enters* node y and *leaves* node x. We say that x is a *predecessor* of y, and y a *successor* of x.

(3) n_0 in N is the *initial node*. There is a path[2] from n_0 to every node.

Definition: A *semilattice* is a set L with a binary *meet* operation $\wedge$ such that for all $a, b, c \in L$:

$$a \wedge a = a \qquad \text{(idempotent)}$$

$$a \wedge b = b \wedge a \qquad \text{(commutative)}$$

$$a \wedge (b \wedge c) = (a \wedge b) \wedge c \qquad \text{(associative)}$$

Definition: Given a semilattice L and elements, $a, b \in L$, we say that

$$a \geqq b \quad \text{iff } a \wedge b = b$$

$$a > b \quad \text{iff } a \wedge b = b \text{ and } a \neq b$$

also $a \leqq b$ means $b \geqq a$ and $a < b$ means $b > a$. We extend the notation of the meet operation to arbitrary finite sets by saying

$$\bigwedge_{1 \leqq i \leqq n} x_i = x_1 \wedge x_2 \wedge \ldots x_n$$

Definition: A semilattice L is said to have a *zero element 0*, if for all $x \in L$, $0 \wedge x = 0$. L is said to have a *one element 1*, if $1 \wedge x = x$ for all $x \in L$. We assume from here on that every semilattice has a zero element, but not necessarily a one element.

Definition: Given a semilattice L, a sequence of elements $x_1, x_2, \ldots x_n$ in L forms a *chain* if $x_i > x_{i+1}$ for $1 \leqq i < n$. L is said to be *bounded* if for each $x \in L$ there is a constant b_x such that each chain beginning with x has length at most b_x.

If L is bounded, then we can take meets over countably infinite sets if we define $\bigwedge_{x \in S} x$, where $S = \{x_1, x_2, \ldots\}$, to be $\lim_{n \to \infty} \bigwedge_{1 \leqq i \leqq n} x_i$. The fact that L is bounded assures us there is an integer m such that $\bigwedge_{x \in S} x = \bigwedge_{1 \leqq i \leqq m} x_i$.

3. Monotone Data Flow Analysis Frameworks

Definition: Given a bounded semilattice L, a set of functions F on L is said to be a *monotone function space associated with* L if the following conditions are satisfied:

[M1] Each $f \in F$ satisfies the *monotoncity* condition,

$$(\forall x, y \in L)(\forall f \in F)[f(x \wedge y) \leqq f(x) \wedge f(y)].$$

[M2] There exists an identity function i in F, such that

$$(\forall x \in L)[i(x) = x].$$

2 A *path* from n_1 to n_k is a sequence of nodes $n_1, n_2, \ldots, n_k$ such that (n_i, n_{i+1}) is in E for $1 \leqq i \leqq k-1$. The *path length* is $k-1$.

[M3] F is closed under composition, i.e. $f, g \in F \Rightarrow fg \in F$, where

$$(\forall x, y \in L)[fg(x) = f(g(x))].$$

[M4] L is equal to the closure of $\{0\}$ under the meet operation and application of functions in F.

Observation 1. Given a semilattice L, let f be a function on L, then

$$(\forall x, y \in L)[f(x \wedge y) \leqq f(x) \wedge f(y) \Leftrightarrow (\forall x, y \in L)[x \leqq y \text{ implies } f(x) \leqq f(y)].$$

The above condition was also observed by Graham and Wegman [5].

Observation 2. For any bounded semilattice L and any countable set $S \subseteq L$, if for all $x \in S$ we have $x \geqq y$, then $\bigwedge_{x \in S} x \geqq y$.

Definition: A *Monotone data flow analysis framework* is a triple $D = (L, \wedge, F)$, where

(1) L is a bounded semilattice with meet $\wedge$.

(2) F is a monotone function space associated with L.

A *particular instance* of a monotone data flow analysis framework is a pair $I = (G, M)$, where

(1) $G = (N, E, n_0)$ is a flow graph.

(2) $M: N \to F$ is a function which maps each node in N to a function in F.

Previous study has been done by Kildall [9] on those monotone data flow analysis frameworks $D = (L, \wedge, F)$ which satisfy the condition:

$$(\forall x, y \in L)(\forall f \in F)[f(x \wedge y) = f(x) \wedge f(y)] \quad (\textit{distributivity}).$$

That is, each f in F is a homomorphism on L. Recently, Graham and Wegman [5], Tennenbaum [3], and Wegbreit [17] have also considered models similar to monotone frameworks. However, there are many interesting gdfap's which are monotone data flow analysis frameworks but which do not satisfy the distributivity property. The following are some examples.

Constant Propagation can be formalized [9] as a monotone data flow analysis framework CONST $= (L, \wedge, F)$. Here $L \subset 2^{V \times R}$, where $V = \{A_1, A_2, \ldots\}$ is an infinite set of variables and R is the set of all real numbers.

L is the set of functions from finite subsets of V to R.

$0 \in L$ is the function which is undefined for all $A_i \in V$.

The meet operation on L is set intersection[3].

Intuitively, $z \in L$ stands for the information about variables which we may assume at certain points of the program flow graph. $(A, r) \in z$ implies the variable A has value r.

We define a notation for functions in F based on the sequence of assignments whose effect they are to model.

(1) There are functions denoted $\langle A := B \theta C \rangle$ and $\langle A := r \rangle$ in F, for each A, B and C in V, $r \in R$ and $\theta \in \{+, -, *, /\}$

Let $z \in L$. Then

3 Let W be a finite subset of V. Recall that $f: W \to R$ is a set of pairs (A, c) with $A \in W$, and $c \in R$. We shall henceforth treat members of L as finite subsets of $V \times R$.

(i) $\langle A := B\theta C\rangle(z)=z'$, where $z'(X)=z(X)$ for all $X\in V-\{A\}$; $z'(A)$ is undefined unless $z(B)=b$ and $z(C)=c$ for some b and c in R, in which case $z'(A)=b\theta c$.

(ii) $\langle A := r\rangle(z)=z'$ where $z'(X)=z(X)$ for all $X\in V-\{A\}$ and $z'(A)=r$.

(2) $i\in F$, where $i(z)=z$ for all $z\in L$.

(3) If $f, g\in F$ then $fg\in F$.

Lemma 1. Let L be a semilattice and $f_1, f_2, \ldots, f_n$ be functions on L. If it is true that $(\forall x, y\in L)(\forall 1\leqq i\leqq n)[f_i(x\wedge y)\leqq f_i(x)\wedge f_i(y)]$, then $f_1 f_2 \ldots f_n(x\wedge y)\leqq f_1 f_2\ldots f_n(x)\wedge f_1 f_2\ldots f_n(y)$.

Proof. $f_n(x\wedge y)\leqq f_n(x)\wedge f_n(y)$ (by assumption). Suppose $f_i\ldots f_n(x\wedge y)\leqq f_i\ldots f_n(x)\wedge f_i\ldots f_n(y)$, then $f_{i-1}(f_i\ldots f_n(x\wedge y))\leqq f_{i-1}(f_i\ldots f_n(x)\wedge f_i\ldots f_n(y))$ (by Observation 1). $f_{i-1}(f_i\ldots f_n(x)\wedge f_i\ldots f_n(y))\leqq f_{i-1}\ldots f_n(x)\wedge f_{i-1}\ldots f_n(y)$ (by assumption). So by simple backward induction on i, the lemma follows. □

Theorem 1. CONST$=(L, \wedge, F)$ is a monotone data flow analysis framework. Furthermore there exists $z, z'\in L$ and $f\in F$ such that $f(z\wedge z')<f(z)\wedge f(z')$.

Proof. The fact that L is a bounded semilattice with a 0 element is obvious. Furthermore, for any element $z\in L$, $z=f_1 f_2\ldots f_n(0)$ for some integer n, where f_i is of the form $\langle A_i=r\rangle$. So to show that F is a monotone function space associated with L, it suffices ,by Lemma 1, to show that for all z, $z'\in L$ and all functions in F of the form $\langle A := B\theta C\rangle$ or $\langle A := r\rangle$,

$$\langle A := B\theta C\rangle(z\wedge z')\leqq\langle A := B\theta C>(z)\wedge\langle A := B\theta C\rangle(z'),$$

and

$$\langle A := r\rangle(z\wedge z')\leqq\langle A := r\rangle(z)\wedge\langle A := r\rangle(z').$$

Observe that since $\wedge$ is intersection on L, the $\leqq$ relation is set inclusion.

(i) Suppose we are given $z, z'\in L$ and $\langle A := B\theta C\rangle\in F$.
Let $y=\langle A := B\theta C\rangle(z\wedge z')$. Then for all $X\in V-\{A\}$, if $(X, r)\in y$ then $(X, r)\in z$ and $(X, r)\in z'$. Hence $(X, r)\in\langle A := B\theta C\rangle(z)$ and $(X, r)\in\langle A := B\theta C\rangle(z')$.

If A is undefined in y, then we are done. Suppose however, that $(A, r)\in y$. Then $\{(B, r_1), (C, r_2)\}$ is a subset of z and is also a subset of z', for some r_1 and r_2 such that $r=r_1\,\theta\, r_2$. This implies that $(A, r)\in\langle A := B\theta C\rangle(z)$ and $(A, r)\in\langle A := B\theta C\rangle(z')$

(ii) Suppose we are given $z, z'\in L$ and $\langle A := r\rangle\in F$. It is straightforward to show that $\langle A := r\rangle(z\wedge z')=\langle A := r\rangle(z)\wedge\langle A := r\rangle(z')$. Hence the first part of the lemma follows.

For a counterexample showing that CONST is not distributive, consider the flow chart of Figure 1. There we see that $\langle C := A+B\rangle(z\wedge z')=\emptyset$, while $\langle C := A+B\rangle(z)\wedge\langle C := A+B\rangle(z')=\{(C, 5)\}$. □

It should be noted that in Figure 1, C really does have the value 5, so the CONST framework fails to detect at compile time a constant relationship which holds at runtime.

We shall also mention that Theorem 1 can be generalized to any framework whose lattice elements associate "values" with variables, whose meet operation is intersection, and whose functions reflect the application of "operators" on those values and assignment of values to variable. The framework will be monotone in all cases, but will be distributive only if the interpretation of the opera-

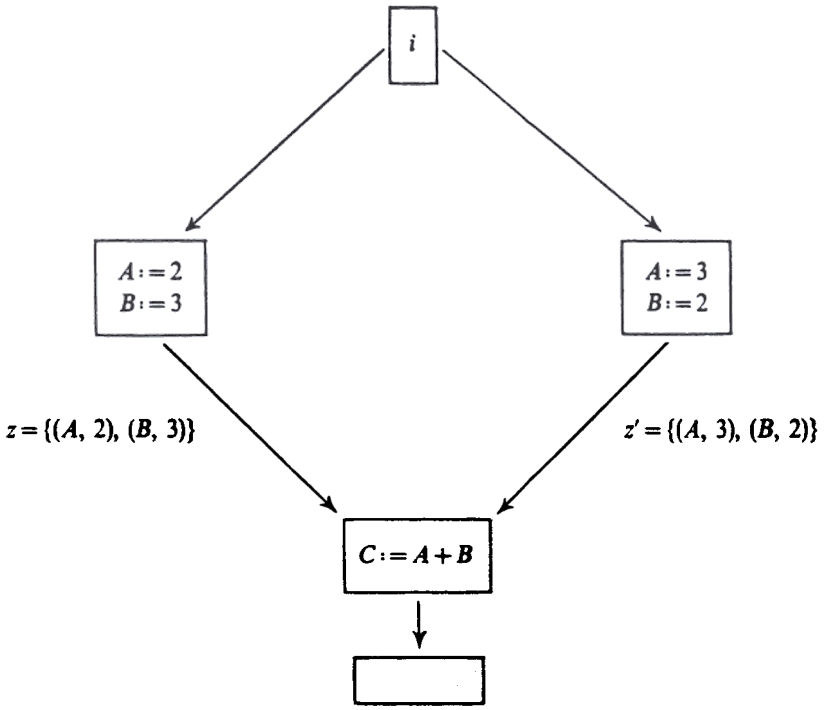

Fig. 1. Counter example to distributivity of CONST

tors is "free", that is, the effect of applying k-ary operator θ to two different k-tuples of values is never the same.

Numerous additional examples of monotone but not distributive frameworks can be found in the literature. Examples are the "structured partition" framework from [9], Tennenbaum's type checking [13] and Schwartz's framework for detecting the liveness of computed values in SETL [11].

4. Approaches to Solving Monotone Data Flow Analysis Problems

It appears generally true that what one seaches for in a data flow problem is what we shall call the *meet over all paths* (MOP) solution. That is, let PATH(n) denote the set of paths from the initial node to node n in some flow graph. Then we really want $\bigwedge_{P \in \mathrm{PATH}(n)} f_P(0)$ for each n. It is this function, the MOP solution that, in any practical data flow problem we can think of, expresses the desired information. For example, in Figure 1, the MOP solution would have $C=5$ at the point following the assignment $C := A+B$ because both paths to that point set C to 5.

The people solving bit vector data flow analysis problems, such as [1, 2, 6, 8, 12, 14], or problems based on distributive frameworks [9] obtain the MOP solution by finding maximum fixed point of a set of equations. As [9] shows, this fixed point is always the MOP solution in this case. However, the MOP solution is not the maximum fixed point of the equations in the case of a general monotone framework, and this fact explains why Kildall's method "fails" on the framework CONST discussed in Theorem 1.

We shall here consider what of Kildall's theory remains true in the context of general monotone frameworks. Our first approach is to consider what happens when the algorithm of [9] is applied to a monotone framework.

In order to make the algorithm below simple to read, for each $D=(L, \wedge, F)$ if L does not contain a one element 1, we introduce an artificial element 1, such that

$$(\forall f \in F)(\forall x \in L)[1 \wedge x = x \wedge 1 = x \quad \text{and} \quad f(1)=1]$$

Algorithm 1 (Essentially Kildall's Algorithm [9] applied to a monotone framework)

Input. A particular instance $I=(G, M)$ of $D=(L, \wedge, F)$, where $G=(N, E, n_0)$ is a flow graph.

Initialization.

$$(\forall n \in N) \qquad A[n] \begin{cases} 0 & \text{if } n = n_0 \\ 1 & \text{otherwise} \end{cases}$$

Iteration Step. Visit nodes other than n_0 in order $n_1, n_2, \ldots$ (with repetitions, and not fixed in advance). We *visit* node n by setting

$$A[n] = \bigwedge_{p \in \mathrm{PRED}(n)} f_p(A[p])$$

where $\mathrm{PRED}(n)=\{p \mid (p, n) \in E\}$ The sequence $n_1, n_2, \ldots$ has to satisfy the following condition:

If there exists a node $n \in N-\{n_0\}$ such that $A[n] \neq \bigwedge_{p \in \mathrm{PRED}} f_p(A[p])$ after we have visited node n_s in the sequence, then there exists integer $t>s$ such that $n_t=n$. Also, if after visiting node n_s, $A[n]=\bigwedge_{p \in \mathrm{PRED}} f_p(A[p])$ for all $n \neq n_0$, then the sequence will eventually end.

Convention. Given instance $I=(G, M)$ of framework $D=(L, \wedge, F)$, if we apply Algorithm 1 to I with the sequence $n_1, n_2, \ldots$, we say that the j-th step of Algorithm 1 has been applied after we have visited nodes $n_1, n_2, \ldots, n_j$. Let n be a node in G. We let $A^{(m)}[n]$ denote the value of $A[n]$ right after step m of Algorithm 1 has been applied.

Convention. Given a particular instance $I=(G, M)$ of $D=(L, \wedge, F)$, we let f_n denote $M(n)$, the function in F which is associated with node n. Let $P=n_1, n_2, \ldots, n_m, n_{m+1}$ be a path in G. Then we may use $f_P(.)$ for $f_{n_m}(f_{n_{m-1}}(\ldots f_{n_1}(.)\ldots))$. Note that $f_{n_{m+1}}$ is not in the composition.

Lemma 2. Given an instance $I=(G, M)$ of a monotone data flow analysis framework $D=(L, \wedge, F)$, if we apply Algorithm 1 to I, the algorithm will eventually halt.

Proof. It is a simple proof by induction on m, the number of steps applied in Algorithm 1, that $A^{(m+1)}[n] \leqq A^{(m)}[n]$, for all nodes in G. According to the condition on the sequence of nodes being visited, after we have applied the k-th step of Algorithm 1, either there exists an integer j such that $A^{(k+j+1)}[n] < A^{(k+j)}[n]$ for some node n in G, or the sequence will halt. The facts that L is bounded and that G has only finitely many nodes guarantee that the sequence ends and the algorithm will eventually halt. □

Theorem 2. Given an instance $I=(G, M)$ of framework $D=(L, \wedge, F)$, after we have applied Algorithm 1 to I, we have $(\forall n \in N)\left[A[n] \leqq \bigwedge_{P \in \mathrm{PATH}(n)} f_P(0)\right]$, where $\mathrm{PATH}(n)=\{P \mid P$ is a path in G from n_0 to $n\}$.

Proof. We want to prove by induction on l that $(\forall n \in N)\left(A[n] \leqq \bigwedge_{P \in \mathrm{PATH}_l(n)} f_P(0)\right)$, where $\mathrm{PATH}_l(n)=\{P \mid P$ is a path of length l from n_0 to n$\}$.

Basis. $(l=0)$ n_0 is the only node that has a path from n_0 of zero length. Since $A[n_0]$ is assigned 0 initially and not changed afterwards, the basis holds.

Inductions step. $(l>0)$ If $n=n_0$, we are done. Suppose $n \neq n_0$. We have $A[n]=\bigwedge_{p \in \mathrm{PRED}(n)} f_p(A[p])$, and $(\forall p \in \mathrm{PRED}(n))\left(A[p] \leqq \bigwedge_{Q \in \mathrm{PATH}_{l-1}(p)} f_Q(0)\right)$, by hypothesis. Thus $A[n] \leqq \bigwedge_{p \in \mathrm{PRED}(n)} f_p\left(\bigwedge_{Q \in \mathrm{PATH}_{l-1}(p)} f_Q(0)\right)$ by monotonicity and Observation 1. By mononicity again, we have $A[n] \leqq \bigwedge_{\substack{p \in \mathrm{PRED}(n) \\ Q \in \mathrm{PATH}_{l-1}(p)}} f_p(f_Q(0))=\bigwedge_{P \in \mathrm{PATH}_l} f_P(0)$. By Observation 2, we have for all $n \in N$

$$A[n] \leqq \bigwedge_{P \in \mathrm{PATH}(n)} f_P(0). \quad \square$$

Theorem 3. Given an instance $I=(G, M)$ of a monotone framework $D=(L, \wedge, F)$, after we have applied Algorithm 1, the solution $A[n]$'s we get is the maximum fixed point solution of the set of simultaneous equations

$$\begin{gathered} X[n_0]=0 \\ (\forall n \in N-\{n_0\})\left(X[n]=\bigwedge_{p \in \mathrm{PRED}(n)} f_p(X[p])\right) \end{gathered} \tag{*}$$

Proof. It is obvious that, after Algorithm 1 halts, the $A[n]$'s satisfy the Equations (*). Now suppose we are given any solution $B[n]$'s to the Equations (*). We want to prove by induction on m, the number of steps applied in Algorithm 1, that after the m-th step $B[n] \leqq A^{(m)}[n]$ for all $n \in N$.

Basis. $(m=0)$ Obvious.

Inductions step. $(m>0)$ At the m-th step, we have $A^{(m)}[n_m]:=\bigwedge_{p \in \mathrm{PRED}(n_m)} f_p(A^{(m-1)}[p])$. Since we have $(\forall p \in \mathrm{PRED}(n_m))(B[p] \leqq A^{(m-1)}[p])$ by the induction hypothesis, we have $B[n_m]=\bigwedge_{P \in \mathrm{PATH}(n_m)} f_P(B[p]) \leqq A^{(m)}[n_m]$ by monotonicity. For the rest the nodes $n \in N-\{n_m\}$, $A^{(m)}[n]=A^{(m-1)}[n]$.

The theorem then follows from the fact that Algorithm 1 will eventually halt. $\square$

Corollary. Given an instance $I=(G, M)$ of a framework $D=(L, \wedge, F)$, as input to Algorithm 1, the $A[n]$'s we get after Algorithm 1 halts is unique independent of the sequence in which nodes are visited, provided the sequence satisfies the condition stated in the algorithm.

Theorem 4. Given a monotone framework $D=(L, \wedge, F)$, suppose $(\exists x, y \in L) \cdot (\exists f \in F)[f(x \wedge y)<f(x) \wedge f(y)]$, i.e. D is not distributive. Then there exists an instance $I=(G, M)$ such that after we apply Algorithm 1, there is a node n in G such that

$$A[n]<\bigwedge_{P \in \mathrm{PATH}(n)} f_P(0).$$

Proof. By condition [M4] in the definition of a monotone function space, we can find acylic graphs G_x and G_y, with input nodes n_x and n_y, and output nodes

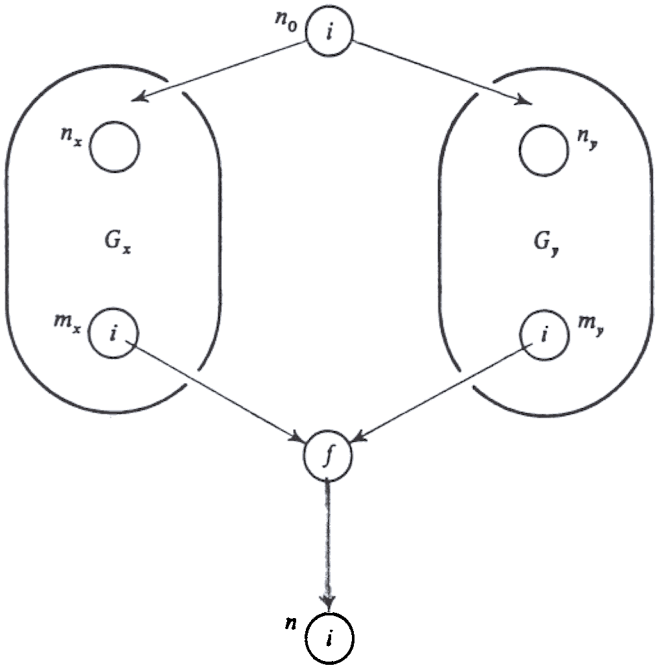

Fig. 2

m_x and m_y, such that after we apply Algorithm 1 to G_x and G_y, we get $A[m_x]=x$ and $A[m_y]=y$. A straightforward induction on the number of meet operations and function applications necessary to construct a lattice element from 0 proves the existence of G_x and G_y.

Consider the graph G of Figure 2. It is easy to check that if we apply Algorithm 1 we have $A[n]=f(x \wedge y)$. By Theorem 2, in G we have $x \leqq \bigwedge_{P \in \mathrm{PATH}(n_x)} f_P(0)$ and $y \leqq \bigwedge_{P \in \mathrm{PATH}(n_y)} f_P(0)$. Thus $\bigwedge_{P \in \mathrm{PATH}(n)} f_P(0) \geqq f(x) \wedge f(y)$ by monotonicity. But we are given $f(x) \wedge f(y) > f(x \wedge y)$, so $A[n] < \bigwedge_{P \in \mathrm{PATH}(n)} f_P(0)$. □

In summary then, Kildall 's algorithm applied to a monotone data flow analysis framework yields a unique solution, independent of the order in which nodes are visited. This solution is the maximum fixed point of the set of equations associated with a flow graph. However, we can only show that the solution is equal to or less than the MOP solution, and when the framework is not distributive there will always be some instance in which the inequality is strict.

5. A Variant of Kildall's Algorithm

We shall now briefly consider an algorithm similar to Kildall's but somewhat more time consuming. This algorithm will obtain the MOP solution in certain situations where Algorithm 1 fails to do so, Figure 2 being a good example of this phenomenon. However, like Algorithm 1, it must fail for some instance of any monotone, nondistributive framework.

We are not proposing this algorithm as an "improvement" on Kildall's algorithm, since the cases in which our algorithm attains the MOP solution and Kildall's doesn't will likely be few and far between in practice. It is interesting, however, to note that the two algorithms differ in their behavior in the general

monotone case, although they are easily shown to produce identical answers for distributive frameworks.

Algorithm 2

Input. As in Algorithm 1

Initialization

$$(\forall n \in N) \qquad B[n] \begin{cases} f_{n_0}(0) & \text{if } n = n_0 \\ I & \text{otherwise} \end{cases}$$

Iteration step. Visit nodes other than n_0 in order $n_1, n_2, \ldots$ (not fixed in advance). We *visit* node n by setting

$$B[n] := \bigwedge_{p \in \mathrm{PRED}(n)} f_n(B[p])$$

The sequence $n_1, n_2, \ldots$ has to satisfy the condition: if there is a node $n \in N - \{n_0\}$ such that $B[n] \neq \bigwedge_{p \in \mathrm{PRED}(n)} f_n(B[p])$ after we have visited node n_s in the sequence, then there exists integer $t > s$ such that $n_t = n$. Also if $B[n] = \bigwedge_{p \in \mathrm{PRED}(n)} f_n(B[p])$ for all $n \neq n_0$, we will eventually halt the iteration.

Final Step. We set

$$H[n_0] = 0$$

$$(\forall n \in N - \{n_0\})\ H[n] = \bigwedge_{p \in \mathrm{PRED}(n)} B[p].$$

Theorem 5. Given an instance $I = (G, M)$ of a framework $D = (L, \wedge, F)$ as input to Algorithm 2:

(i) Algorithm 2 will eventually halt. The result we get is unique independent of the order in which nodes are visited and $(\forall n \in N)\ H[n] \leqq \bigwedge_{P \in \mathrm{PATH}(n)} f_P(0)$.

(ii) The resulting $B[n]$'s form the maximum fixed point of the set of equations

$$X[n_0] = f_{n_0}(0)$$

$$(\forall n \in N - \{n_0\})\ (X[n] = \bigwedge_{p \in \mathrm{PRED}(n)} f_n(X[p]))$$

(iii) If $A[n]$ is the result of applying Algorithm 1 to $I = (G, M)$, then $A[n] \leqq H[n]$.

Proof. The proofs are similar to the proofs of the results in the previous section and we omit them. □

Theorem 6. Given a monotone framework $D = (L, \wedge, F)$, suppose $(\exists x, y \in L)(\exists f \in F)[f(x \wedge y) < f(x) \wedge f(y)]$, i.e. D is not distributive. Then:

(i) there exists an instance $I = (G, M)$ such that there is a node in G such that

$$H[n] < \bigwedge_{P \in \mathrm{PATH}(n)} f_P(0)$$

after we have applied Algorithm 2 to I.

(ii) there exists an instance $I' = (G', M')$ such that there is a node n in G' such that

$$A[n] < H[n]$$

after we have applied Algorithm 1 and Algorithm 2 to instance I.

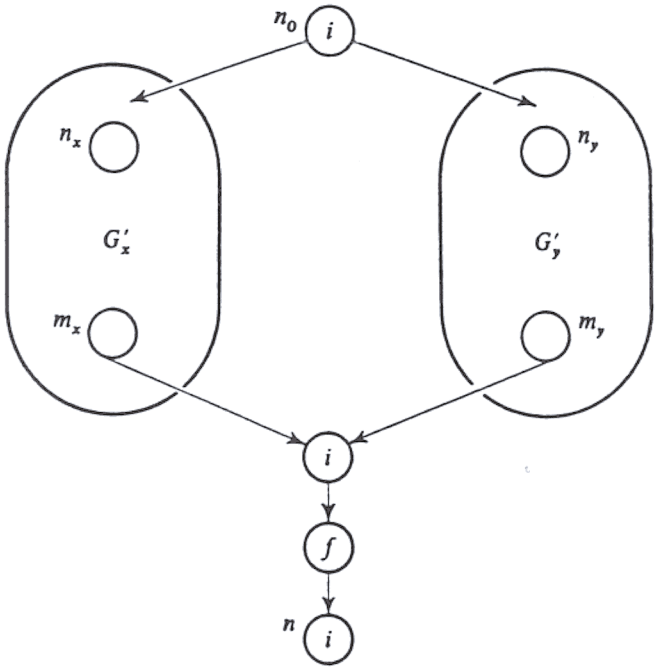

Fig. 3

Proof. We may, as in Theorem 4, invoke condition [M4] to observe that there are graphs G'_x and G'_y such that their output nodes m_x and m_y have $B[m_x]=x$ and $B[m_y]=y$. Then consider the graph of Figure 3. It follows from monotonicity that

$$\bigwedge_{P\in \mathrm{PATH}(n)} f_P(0) \geqq f(x) \wedge f(y) > f(x \wedge y) = H[n].$$

To prove Theorem 6 (ii), we refer to Figure 2. Direct calculation will show that $A[n]=f(x \wedge y)$. By part (iii) of Theorem 5, $H[m_x] \geqq x$ and $H[m_y] \geqq y$. Thus $B[m_x] \geqq x$ and $B[m_y] \geqq y$. It follows that $H[n] \geqq f(x) \wedge f(y)$, so $A[n] < H[n]$. □

6. Undecidability of the MOP Problem for Monotone Data Flow Analysis Frameworks

We have seen that some of the obvious algorithms fail to compute the MOP solution for an arbitrary monotone framework. We shall now show that this situation must hold for any algorithm. In particular, we show that there does not exist an algorithm which, for arbitrary instance $I=(G, M)$ of an arbitrary monotone data flow analysis framework $D=(L, \wedge, F)$, will compute $\bigwedge_{P\in \mathrm{PATH}(n)} f_P(0)$ for all nodes n of G.

Definition. The *Modified Post's Correspondence Problem (MPCP)* is the following: Given arbitrary lists A and B, of k strings each in $\{0, 1\}^+$, say

$$A=w_1, w_2, \ldots, w_k \qquad B=z_1, z_2, \ldots, z_k$$

does there exist a sequence of integers $i_1, i_2, \ldots, i_r$ such that

$$w_1 w_{i_1} w_{i_2} \ldots w_{i_r} = z_1 z_{i_1} z_{i_2} \ldots z_{i_r}$$

It is well known that MPCP is undecidable [7].

Given an instance AB of MPCP with lists $A=w_1, \ldots, w_k$ and $B=z_1, \ldots, z_k$, we can construct a monotone data flow analysis framework $D_{AB}=(L_{AB}, \wedge, F_{AB})$, where the following elements are in L_{AB}:

(1) 0, the lattice zero,

(2) the special element \$, which will in a sense indicate nonsolution to MPCP, and

(3) all strings of integers $1, 2, \ldots, k$ beginning with 1.

The meet on these elements is given by: $x \wedge y = 0$ whenever $x \neq y$. Thus, if $x \leqq y$, then either $x=y$ or $x=0$.

The set of functions F_{AB} includes the following.

(1) the identity function on L_{AB}

(2) functions f_i, for $1 \leqq i \leqq k$ defined by:

(i) if α is a string of integers beginning with 1, then $f_i(\alpha)=\alpha\, i$,

(ii) $f_i(0)=(0)$ and

(iii) $f_i(\$)=\$$,

(3) the function g defined by

(i) for strings $\alpha=1\, i_1\, i_2 \ldots i_m$,

$$g(\alpha)=\begin{cases}0 \text{ if } 1\, i_1\, i_2 \ldots i_m \text{ is a solution to instance } AB \text{ of MPCP}\\ \$ \text{ otherwise,}\end{cases}$$

(ii) $g(0)=0$,

(iii) $g(\$)=\$$,

(4) the function h defined by $h(x)=1$ (that is, the string consisting of 1 alone) for all $x \in L_{AB}$.

(5) All functions constructed from the above by composition.

Lemma 3. $D_{AB}=(L_{AB}, \wedge, F_{AB})$ constructed above is a monotone data flow analysis framework.

Proof. We show only monotonicity; the other properties are easy to check. By Observation 1 and Lemma 1, it suffices to show that if $x \leqq y$ for x and y in L_{AB} then

(1) $f_i(x) \leqq f_i(y)$ for $1 \leqq i \leqq k$,

(2) $g(x) \leqq g(y)$, and

(3) $h(x) \leqq h(y)$.

Since $h(x)=h(y)=1$, (3) is immediate. We have observed that for the meet operation we have defined, $x \leqq y$ implies x is either y or 0. In the former case (1) and (2) are immediate.

In the latter case, $f_i(x)=0$ and $g(x)=0$, so $f_i(x) \leqq f_i(y)$ and $g(x)=g(y)$ follows. □

We can now show that it is impossible to do for monotone frameworks what Kildall did for distributive ones.

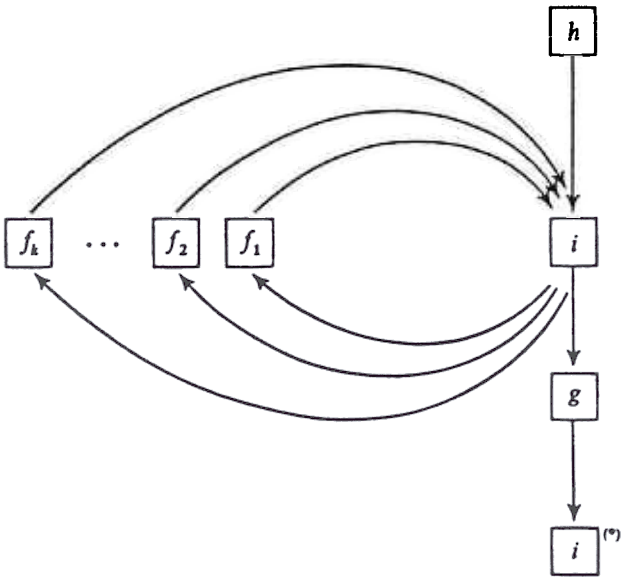

Fig. 4

Theorem 7. There does not exist an algorithm A with the following properties.

(1) the input to A is

(i) Algorithms to perform meet, equality testing and application of functions to lattice elements for some monotone framework and,

(ii) An instance I of the framework.

(2) The output of A is the MOP solution for I.

Proof. If A exists, then we can apply A to any monotone framework D_{AB} constructed above with the instance I chosen as shown in Figure 4. The MOP solution to that problem at the point (*) is easily seen to be \$ if the instance AB of the MPCP has no solution, and *0* if it does. Thus, if A existed, we could solve the MPCP. □

It should be noted that Theorem 7 does not rule out finding algorithms for the MOP solution for particular monotone framework or for large subclasses of them. However, by a proof similar to that of Theorem 7, we can exhibit a particular monotone framework for which no algorithm to compute MOP solutions of its instances exists.

Finally, we shall remark that Theorem 7 is a strengthening of a result by Dana Angluin to the effect that computing the MOP solution for a monotone framework is NP-hard.

References

1. Aho, A. V., Ullman, J. D.: The theory of parsing, translation, and compiling; Vol II—Compiling. Englewood Cliffs (N. J.): Prentice-Hall 1973
2. Cocke, J.: Global common subexpression elimination. SIGPLAN Notices 5, 20–24 (1970)
3. Cocke, J., Schwartz, J. T.: Programming languages and their compilers. Courant Institute, New York University, New York (1970)

4. Fong, A. C., Kam, J. B., Ullman, J. D.: Applications of lattice algebra to loop optimization. Proc. 2nd ACM Conference on Principles of Programming Languages, Palo Alto (Cal.), January 1975, pp. 1–9
5. Graham, S. L., Wegman, M.: A fast and usually linear algorithm for global flow analysis. J. ACM **23**, 172–202 (1976)
6. Hecht, M. S., Ullman, J. D.: A simple algorithm for global flow analysis problems. SIAM J. Computing **4**, 519–532 (1975)
7. Hopcroft, J. E., Ullman, J. D.: Formal languages and their relation to automata. Reading (Mass.): Addison-Wesley 1969
8. Kennedy, K.: A global flow analysis algorithm. International J. Computer Math. **3**, 5–15 (1971)
9. Kildall, G. A.: Global expression optimization during compilation. Proc. ACM Conference on Principles of Programming Languages, Boston (Mass.), October 1973, pp. 194–206
10. Kam, J. B., Ullman, J. D.: Global data flow analysis and iterative algorithms. J. ACM **23**, 158–171 (1976)
11. Schwartz, J. T.: Copy optimization in SETL. Department of Computer Science, Courant Institute, New York University, New York, SETL Newsletters 130,131, 1974
12. Schaefer, M.: A mathematical theory of global program optimization. Englewood Cliffs (N.J.): Prentice-Hall 1973
13. Tennenbaum, T.: Type determination for very high level languages. Department of computer Science, Courant Institute, New York University, New York, TSO-3, 1974
14. Ullman, J. D.: Fast algorithms for the elimination of common subexpressions. Acta Informatica **2**, 191–213 (1973)
15. Ullman, J. D.: Data Flow Analysis. Proc. 2nd USA-Japan Computer Conference. Montvale (N.J.): AFIPS Press 1975
16. Vyssotsky, V. A.: Private Communication to M. S. Hecht, June 1973
17. Wegbreit, B.: Property extraction in well-founded property sets. IEEE Trans. on Software Engineering SE-1, 270–285 (1975)

Dr. John B. Kam
Dr. Jeffrey D. Ullman
Dept. of Electrical Engineering and Computer Science
Princeton University
Princeton, N. J. 08540, USA