On the computational complexity of Data Flow Analysis

Gaurav Sood
gauravsood0289@gmail.com

K. Murali Krishnan
kmurali@nitc.ac.in

Department of Computer Science and Engineering, National Institute of Technology Calicut, Calicut - 673601, Kerala, India

ABSTRACT

We consider the problem of Data Flow Analysis over monotone data flow frameworks with a finite lattice. The problem of computing the Maximum Fixed Point (MFP) solution is shown to be \( \mathcal{P} \)-complete even when the lattice has just four elements. This shows that the problem is unlikely to be efficiently parallelizable. It is also shown that the problem of computing the Meet Over all Paths (MOP) solution is \( \mathcal{NL} \)-complete (and hence efficiently parallelizable) when the lattice is finite even for non-monotone data flow frameworks. Although the meet over all paths (MOP) solution is desirable, solving the MOP problem is undecidable and hence significantly harder than the MFP problem which is polynomial time computable for lattices of finite height.

Categories and Subject Descriptors

F.1.3 [Computation by Abstract Devices]: Complexity Measures and Classes—Reducibility and completeness; D.3.4 [Programming Languages]: Processors—Optimization; F.3.2 [Logics and Meanings of Programs]: Semantics of Programming Languages—Program analysis

General Terms

Theory

Keywords

Maximum fixed point solution, Meet over all paths solution, \( \mathcal{P} \)-complete, \( \mathcal{NL} \)-complete

1. INTRODUCTION

The problem of data flow analysis over a monotone data framework with a bounded meet semilattice has been well studied in the context of static program analysis and available expressions analysis [1, Sec. 9.2.4–9.2.6]. Iterative fixed point methods as in [7] are commonly used to find the maximum fixed point (MFP) solution as a conservative approximation to the MOP solution [6]. Several important problems like reaching definitions analysis, live variable analysis and available expressions analysis [1, Sec. 9.2.4–9.2.6] are essentially data flow analysis problems over monotone data flow frameworks with a bounded meet semilattice.

In this paper, the computational complexity of MFP and MOP data flow analysis problems over monotone data flow frameworks with a finite bounded meet semilattice is investigated. Since a finite bounded meet semilattice is essentially a finite lattice, we define the problem over finite lattices. We show that computing the MFP solution to data flow analysis problem over a monotone data flow framework with a finite lattice is \( \mathcal{P} \)-complete. We further show that the problem of finding MOP solution is non-deterministic log space complete (\( \mathcal{NL} \)-complete). In fact the proof in Section 3 shows that MOP is \( \mathcal{NL} \)-complete even if the functions associated with the vertices of the control flow graph are non-monotone. These results indicate that the MFP problem is unlikely to be in the complexity class \( \mathcal{NL} \) (and hence fast parallel algorithms are unlikely to exist for the problem [5, Chap. 5]). The \( \mathcal{NL} \) complexity of MOP problem yields \( O(\log^2 n) \) depth, polynomial sized parallel circuit for the problem. This further leads to the observation that although MOP computation is harder than MFP computation in general, computing MOP solution appears significantly easier than computing MFP over finite lattices.

2. BACKGROUND

Let \((L, \leq)\) be a partially ordered set. Let \(\lor\) and \(\land\) respectively denote the join and meet operations in \(L\). A partially ordered set \((L, \leq)\) is a meet semilattice if \(\land\) exists for all \(x, y \in L\). A meet semilattice \((L, \land)\) is a lattice, denoted by \((L, \land, \lor)\), if \(x \lor y\) exists for all \(x, y \in L\). A meet semilattice \((L, \land)\) is a bounded meet semilattice, denoted by \((L, \land, 1)\), if there exists an element \(1 \in L\) such that \(l \land 1 = l\) for all \(l \in L\). A lattice \((L, \lor, \land)\) is a bounded lattice, denoted by \((L, \lor, \land, 0, 1)\), if there exist elements \(0, 1 \in L\) such that \(l \lor 0 = l\) and \(l \land 1 = l\) for all \(l \in L\).

A finite bounded meet semilattice \((L, \land, 1)\) is essentially a finite lattice where join operation is defined as follows: For all \(l, l' \in L\), \(l \lor l' = \land\{l'' \in L \mid l \leq l'' \land l' \leq l''\}\). A lattice is complete lattice if \(\lor S\) and \(\land S\) exist for all \(S \subseteq L\).

1 It is easy to see that a finite lattice is complete. \[3\]

Let \(L = \{l_1, l_2, \ldots, l_m\}\) be a finite lattice. Let \(L^n = \{(l_1, \ldots, l_n) \mid 1 \leq j \leq n, l_j \in L\}\). The tuple \((l_1, l_2, \ldots, l_m) \in L^n\) will be denoted by \((l_i)_n\) or simply by \((l_i)\) when there is no ambiguity about the index set. Let \(\ell_j\) denote the \(j^{th}\) element of \((l_i)\). For all \((l_i), (l'_i) \in L^n\), \((l_i) \lor (l'_i) = (l_i \lor l'_i)\) and \((l_i) \land (l'_i) = (l_i \land l'_i)\). \[3\]

A function \(f : A \rightarrow B\) is monotone if for all \(x, y \in A\), \(x \leq y\) implies \(f(x) \leq f(y)\).

Let \(G = (V, E)\) be a directed graph. Let \(\deg^-(v)\) and \(\deg^+(v)\) respectively denote the indegree and outdegree of vertex \(v\). The function \(\text{pred} : V \rightarrow 2^V\) is defined as follows:

\(\text{pred}(v) = \{u \mid (u, v) \in E\}\)
Definition 1. Let \( L \) be a lattice and let \( f : L \rightarrow L \). An element \( l \in L \) is called a fixed point of \( f \) if \( f(l) = l \). An element \( l \in L \) is called the maximum fixed point of \( f \) if it is a fixed point of \( f \) and for every \( l' \in L \) whenever \( f(l') = l' \) then \( l' \leq l \). Let \( \text{MFP}(f) \) denote the maximum fixed point of \( f \) whenever it exists.

2.1 Data Flow Analysis (DFA)

Definition 2. A control flow graph is a finite directed graph \( G = (V, E, v_s, v_t) \) where \( V = \{v_1, v_2, \ldots, v_n\} \), \( v_s \in V \), called entry, is a unique vertex satisfying \( \deg^-(v_s) = 0 \) and \( v_t \in V \), called exit, is a unique vertex satisfying \( \deg^+(v_t) = 0 \). Every vertex \( v_i \in V \) is reachable from \( v_s \).

Definition 3. A monotone data flow framework is a triple \( D = (L, \wedge, F) \) where

\[
\begin{align*}
(L, \vee, \wedge, 0, 1) & \text{ is a finite lattice where } L = \{l_1, l_2, \ldots, l_m\} \text{ with } l_1 = 0 \text{ and } l_m = 1; \\
\wedge & \text{ is the confluence operator; and} \\
F & \text{ is a collection of monotone functions from } L \text{ to } L.
\end{align*}
\]

Definition 4. A Data Flow Analysis (DFA) system is a 5-tuple \( \alpha = (G, D, M, v_0, l_0) \) where

\[
\begin{align*}
G & \text{ is a control flow graph;} \\
D & \text{ is a monotone data flow framework;} \\
M : V \rightarrow F & \text{ assigns a function } f_i \in F \text{ to the vertex } v_i \text{ of } G; \text{ and} \\
v_0 & \in V, \ l_0 \in L.
\end{align*}
\]

2.2 Maximum Fixed Point (MFP) problem

Let \((G, D, M, v_0, l_0)\) be a DFA system where \( G = (V, E, v_s, v_t) \) and \( D = (L, \wedge, F) \). Then \( \mathcal{T} : L^\alpha \rightarrow L^\alpha \) is defined as follows

\[
\mathcal{T}(\langle l_i \rangle) = \langle f_i (\bigwedge_{v_j \in \text{pred}(v_i)} l_j) \rangle
\]

Here we assume that \( \bigwedge S = 1 \) when \( S = \emptyset \). With this convention, it is easy to see that \( \mathcal{T} \) is well defined and monotone in \( L^\alpha \).

Theorem 1 (Knaster–Tarski theorem). Let \((L, \vee, \wedge, 0, 1)\) be a complete lattice and let \( f : L \rightarrow L \) be a monotone function. Then the MFP of \( f \) exists and is unique.

Since every finite lattice is complete, it is clear from Theorem 1 that the MFP of \( \mathcal{T} \) exists and is unique. Suppose \( \langle l_i \rangle \) is the MFP of \( \mathcal{T} \), then we use the notation \( \text{MFP}(v_i) \) for the element \( l_j \).

Definition 5. Maximum Fixed Point (MFP) problem: Given a DFA system \((G, D, M, v_0, l_0)\), decide whether \( \text{MFP}(v_0) = l_0 \).

2.3 Meet Over all Paths (MOP) problem

A path \( p \) from vertex \( v_{i_1} \) to vertex \( v_{i_k} \) in a graph \( G \), called a \( v_{i_1} \)-\( v_{i_k} \) path, is a non-empty alternating sequence \( v_{i_1}, v_{i_2}, \ldots, v_{i_{k-1}}, v_{i_k} \) of vertices and edges such that \( v_{i_j} = \{v_{i_j}, v_{i_{j+1}}\} \) for all \( 1 \leq j < k - 1 \). A \( v_{i_1} \)-\( v_{i_k} \) path is written simply as \( v_{i_1} v_{i_2} \cdots v_{i_k} \) when the edges in question are clear. It may be noted that vertices and edges on a path may not be distinct. The length of path \( p \) is denoted by \( \text{len}(p) \). Let \( f_p = f_{i_k} \circ \cdots \circ f_{i_1} \) be called the path function associated with path \( p \). Let \( P_{ij} \) be the set of all paths from vertex \( v_i \) to vertex \( v_j \) in \( G \).

Definition 6. Given a DFA system \( \alpha = (G, D, M, v_0, l_0) \), the meet over all paths solution, denoted \( \text{MOP}(\cdot) \), is defined as follows

\[
\text{MOP}(v_i) = \bigwedge_{p \in P_{i\cdot}} f_p(1)
\]

Since \( L^\alpha \) is finite and hence complete, though there could be infinitely many \( v_i \)-\( v_j \) paths, \( \text{MOP}(v_i) \) is well defined by taking the infimum of all path functions.

Definition 7. Meet Over All Paths (MOP) problem: Given a DFA system \((G, D, M, v_0, l_0)\), decide whether \( \text{MOP}(v_0) = l_0 \).

2.4 Monotone Circuit Value (MCV) problem

This problem is used for reduction in Section 4 to prove that \( \text{MFP} \) is \( \mathcal{P} \)-complete.

Definition 8. A monotone Boolean circuit is a 4-tuple \( C = (G, I, v_0, \tau) \) where

\[
\begin{align*}
G & \text{ is a finite directed acyclic graph where } V = \{v_1, v_2, \ldots, v_n\}, \text{ and for all } v_i \in V, \deg^-(v_i) \in \{0, 2\}; \\
I & \text{ is the set of input vertices;} \\
v_0 & \in V, \text{ called output, is the unique vertex in } G \text{ satisfying } \deg^+(v_0) = 0; \text{ and} \\
\tau : V \rightarrow \{\circ, +\} & \text{ assigns either the Boolean AND function (denoted by } \circ \text{) or the Boolean OR function (denoted by } + \text{) to each vertex of } G.
\end{align*}
\]

Let \( u_j \) be the \( j^{th} \) input vertex of a Boolean circuit \( C \) and let \( \langle x_i \rangle_{x_i} \in \{0, 1\}^{|I|} \) be the input to the circuit. The input value assignment is a function \( \nu : I \rightarrow \{0, 1\} \) defined as follows

\[
\nu(u_j) = x_j \quad \forall u_j \in I, x_j \in \{0, 1\}
\]

The function \( \nu : I \rightarrow \{0, 1\} \) can be extended to the function \( \nu : V \rightarrow \{0, 1\} \) called value of a node defined as follows

\[
\nu(v_k) = \begin{cases} 
\nu(v_i) \circ \nu(v_j) & \text{if } \tau(v_k) = \circ \text{ and } \text{pred}(v_k) = \{v_i, v_j\}, \\
\nu(v_i) + \nu(v_j) & \text{if } \tau(v_k) = + \text{ and } \text{pred}(v_k) = \{v_i, v_j\}
\end{cases}
\]

It is easy to see that \( \nu \) is well defined when \( G \) is a directed acyclic graph.

Definition 9. An instance of Monotone Circuit Value (MCV) problem is a pair \((C, \nu)\) with \( C = (G, I, v_0, \tau) \) where

\[
\begin{align*}
C & \text{ is a monotone Boolean circuit; and}
\end{align*}
\]
• \( \nu : I \rightarrow \{0, 1\} \) is an input value assignment.

**Definition 10. Monotone Circuit Value (MCV) problem:**
Given an instance \((C, \nu)\) of MCV, decide whether \( \nu(v_0) = 1 \) [9] p. 122].

2.5 **Graph Meet Reachability (GMR) problem**

This problem will be used as an intermediate problem in Section 4 for showing that MOP is \( \mathcal{NL} \)-complete.

**Definition 11.** Let \( A = \{a_1, a_2, \ldots, a_n\} \) be a finite set and \((L, \vee, \wedge, 0, 1)\) be a finite lattice where \( L = \{l_1, l_2, \ldots, l_m\} \). A directed graph \( G = (V, E) \) is said to be a product graph of \( A \) and \( L \) if

- \( V = \{v_{ij} \mid a_i \in A, l_j \in L\} \) is the set of vertices; and
- \( E \subseteq V \times V \) is the set of directed edges.

**Definition 12.** An instance of Graph Meet Reachability (GMR) problem is a 6-tuple \((G, A, L, v_{0\theta}, a_{\theta'}, l_{\theta'})\) where

- \( G = (V, E) \) is a product graph of \( A \) and \( L \);
- \( v_{0\theta} \in V \) where \( a_{\theta} \in A \) and \( l_{\theta} \in L \);
- \( a_{\theta'} \in A \); and
- \( l_{\theta'} \in L \).

Let \( R_i = \{l_j \mid v_{ij} \text{ is reachable from } v_{0\theta}\} \).

**Definition 13.** Graph Meet Reachability (GMR) problem:
Given an instance \((G, A, L, v_{0\theta}, a_{\theta'}, l_{\theta'})\) of GMR, decide whether

\[
\bigwedge_{i \in R_{\theta'}} l_i = l_{\theta'}
\]

2.6 **Graph Reachability (GR) problem**

Graph Reachability problem is a well known \( \mathcal{NL} \)-complete problem which will be used for reduction in this paper.

**Definition 14.** An instance of Graph Reachability (GR) problem is a triple \((G, v_s, v_t)\) where

- \( G = (V, E) \) is a directed graph; and
- \( v_s, v_t \in V \).

**Definition 15.** Graph Reachability (GR) problem:
Given an instance \((G, v_s, v_t)\) of GR, decide whether \( v_t \) is reachable from \( v_s \).

**Fact 1.** GR is \( \mathcal{NL} \)-complete [9] Theorem 16.2 on p. 398].

3. **RELATED WORK**

It is shown in [9] that the problem of finding meet over all valid paths (MVP) solution to the interprocedural data flow analysis over a distributive data flow framework with possibly infinite (resp. finite subset) semilattice is \( \mathcal{P} \)-hard (resp. \( \mathcal{P} \)-complete).

It is shown in [9][10] that the problem of finding MFP and MOP solution to data flow analysis over a distributive data flow framework with a distributive sublattice of the power set lattice of a finite set is reducible to graph reachability problem. Hence the problem is non-deterministic log space computable i.e., belongs to the complexity class \( \mathcal{NL} \) (see [9] p. 142 for definition). Since \( \mathcal{NL} \subseteq \mathcal{NC} \)[8, p. 142] for definition). Since \( \mathcal{NL} \subseteq \mathcal{NC} \), \( \mathcal{NC} \) admits fast parallel solutions, these results show that the above problem admits fast parallel algorithms.

The following is an outline for rest of the paper. In Section 4 we show that MFP is \( \mathcal{P} \)-complete by reduction from MCV. In Section 5 we give an \( \mathcal{NL} \) algorithm for computing GMR. In Section 6 we prove that MOP is log space reducible to GMR thereby showing that MOP is in \( \mathcal{NL} \).

Completeness of MOP w.r.t. the class \( \mathcal{NL} \) follows easily by a log space reduction from GR to MOP.

4. **MFP IS P-COMPLETE**

In this section, we give a log space reduction from MCV to MFP. Since MFP is in \( \mathcal{P} \) [9] and MCV is \( \mathcal{P} \)-complete [9], the reduction implies that MFP is also \( \mathcal{P} \)-complete.

4.1 **A reduction from MCV to MFP**

Given an instance \( \alpha = (C, \nu) \) of MCV with \( C = (G, I, v_0, \tau) \). Construct an instance of MFP \( \alpha' = (G', D, M, v_0', (1, 1)) \) as follows

- \( G' = (V', E', v_0', v_1') \) where
  - \( V' = \{v_0'\} \cup \{v_1' \mid v_i \in V(G) \setminus I\} \cup \{v_1' \mid v_i \in V(G)\} \)
  - \( E' \) is defined as follows
    * For each input vertex \( v_i \in I \) add the edge \((v_0', v_1')\) to \( E' \)
    * For each vertex \( v_i \in V \) add the edge \((v_0', v_1')\) to \( E' \)
    * For each vertex \( v_k \in V \setminus I \) with predecessors \( v_i \) and \( v_j \) with \( i < j \) add the edges \((v_1', v_0'), (v_1', v_1')\) to \( E' \). Note that each \( v_0' \in V \setminus I \) has a unique predecessor in \( G \).
- \( D = (L, \bigvee, F) \) is defined as follows
  - \( L = \{(0, 0), (0, 1), (1, 0), (1, 1)\} \) where
    * \( \bigvee \) is bitwise + operation in \([0, 1] \times [0, 1] \)
    * \( \bigwedge \) is bitwise \( \circ \) operation in \([0, 1] \times [0, 1] \)
  - \( \bigvee \) is the confluence operator
  - \( F = \{g_1, g_0, g_{sw}, g_0, g_1\} \) where
    * \( g_1 : L \rightarrow L \) is the identity function
    * \( g_0 : L \rightarrow L \) is defined as follows
      - \( g_0((a_1, a_2)) = (1, 0) \quad \forall (a_1, a_2) \in L \)
    * \( g_1 : L \rightarrow L \) is defined as follows
      - \( g_1((a_1, a_2)) = (1, 1) \quad \forall (a_1, a_2) \in L \)
    * The swap function \( g_{sw} : L \rightarrow L \) is defined as follows
      - \( g_{sw}((a_1, a_2)) = (a_2, a_1) \quad \forall (a_1, a_2) \in L \)
    * \( g_0 : L \rightarrow L \) is defined as follows
      - \( g_0((a_1, a_2)) = (1, a_1 \circ a_2) \quad \forall (a_1, a_2) \in L \)
Figure 1: An instance of MCV

+ $g_+ : L \rightarrow L$ is defined as follows

$$g_+((a_1,a_2)) = (1, a_1 + a_2) \quad \forall (a_1,a_2) \in L$$

It is easy to see that level function is well defined.

- $M : V' \rightarrow F$ is defined as follows

$$M(v_0^1) = f_0^1 = g_I$$

$$M(v_i^1) = f_i^1 = g_0 \quad \forall v_i^0 \in V'$$

$$M(v_j^1) = f_j^1 = \begin{cases} g_o & \text{if } v_i \in I \text{ and } \nu(v_i) = 0 \\ g_+ & \text{if } v_i \in I \text{ and } \nu(v_i) = 1 \end{cases}$$

More compactly $f_i^1 = g_\nu(v_i)$ if $i \in I$.

$$M(v_i^1) = f_i^1 = \begin{cases} g_\circ & \text{if } v_i \in V \setminus I \text{ and } \tau(v_i) = \circ \\ g_+ & \text{if } v_i \in V \setminus I \text{ and } \tau(v_i) = + \end{cases}$$

Figure 1 shows an instance of MCV where $I = \{v_1,v_2,v_3\}$. Figure 2 shows an instance of MFP constructed from the MCV instance of Figure 1.

4.2 Proof of correctness

Let $G = (V,E)$ be a directed acyclic graph. level$(v): V \rightarrow \mathbb{N}$ is defined as follows

$$\text{level}(v_i) = \begin{cases} 0 & \text{if } \deg^+(v_i) = 0 \\ 1 + \max_{v_j \in \text{pred}(v_i)} \text{level}(v_j) & \text{if } \deg^- (v_i) > 0 \end{cases}$$

It is easy to see that level function is well defined.

**Lemma 1.** Let $\alpha = (C,\nu)$ be an instance of MCV with $C = (G,I,v_9,\tau)$ and $G = (V,E)$. Let $\alpha' = (G',D,M,v_0^1,(1,1))$ be the instance of MFP as constructed in Section 4.1. Then $\mathcal{F}$ has a unique fixed point. For all $v_i \in V$, MFP$(v_i^1) = (1,\nu(v_i))$ and for all $v_i \in V \setminus I$, if $v_i^1$ is the predecessor of $v_i^0$ in $G'$, then MFP$(v_i^0) = (\nu(v_i),1)$.

**Proof.** MFP$(\mathcal{F})$ exists by Theorem 1. Therefore, $\mathcal{F}$ has at least one fixed point. Let $\langle l_i \rangle$ be an arbitrary fixed point of $\mathcal{F}$. Let $\ell_i^0$ and $\ell_i^1$ denote the elements of $\langle l_i \rangle$ corresponding to vertices $v_i^0$ and $v_i^1$ respectively. We first prove that $\ell_i^0$ is uniquely defined for all vertices $v_i^0$ of $G'$.

$\ell_i^0 = g_0(\emptyset) = g_I(1) = 1$ is uniquely defined. Let $v_i$ be an arbitrary vertex of $V$. We prove the uniqueness of $\ell_i^k$ by induction on level$(v_i)$.

- **Base case:** level$(v_i) = 0$ i.e. $v_i \in I$. So, $f_i^1 = g_\nu(v_i)$ and $g_\nu(v_i)((a_1,a_2)) = (1,\nu(v_i)) \forall (a_1,a_2) \in L$ by definition. From Equation 2, $\ell_i^k = f_i^1(\ell_i^0) = g_\nu(v_i)(\ell_i^0) = (1,\nu(v_i))$ is uniquely defined.

- **Inductive step:** Let the theorem be true $\forall v_j \in V$ such that level$(v_j) < m$. Let level$(v_k) = m$. Let $v_i, v_j, i < j$ be predecessors of $v_k$ in $G$. By definition of level function, level$(v_k) = m$ and level$(v_j) < level(l_k) = m$. By induction hypothesis, $\ell_i^k = (1,\nu(v_i))$ and $\ell_j^k = (1,\nu(v_j))$.

From Equation 2, $\ell_k^m = g_\nu(v_k) = g_\nu(v_k)(\ell_i^0) = (1,\nu(v_k))$, $\ell_i^k$ is uniquely defined.

$\ell_i^0 = g_0(\emptyset) = g_I(1) = 1$ is uniquely defined. Let $v_i$ be an arbitrary vertex of $V$. We prove the uniqueness of $\ell_i^k$ by induction on level$(v_i)$.

- **Base case:** level$(v_i) = 0$ i.e. $v_i \in I$. So, $f_i^1 = g_\nu(v_i)$ and $g_\nu(v_i)((a_1,a_2)) = (1,\nu(v_i)) \forall (a_1,a_2) \in L$ by definition. From Equation 2, $\ell_i^k = f_i^1(\ell_i^0) = g_\nu(v_i)(\ell_i^0) = (1,\nu(v_i))$ is uniquely defined.

- **Inductive step:** Let the theorem be true $\forall v_j \in V$ such that level$(v_j) < m$. Let level$(v_k) = m$. Let $v_i, v_j, i < j$ be predecessors of $v_k$ in $G$. By definition of level function, level$(v_k) = m$ and level$(v_j) < level(l_k) = m$. By induction hypothesis, $\ell_i^k = (1,\nu(v_i))$ and $\ell_j^k = (1,\nu(v_j))$.

From Equation 2, $\ell_k^m = g_\nu(v_k) = g_\nu(v_k)(\ell_i^0) = (1,\nu(v_k))$, $\ell_i^k$ is uniquely defined.

The case $\tau(v_k) = +$ is proved similarly.
Since $\ell^i_0$ is uniquely defined for all vertices $v_i^0$ of $G'$, $(l_i)$ is unique and hence $(l_i)$ is the maximum fixed point of $F$. So, for all $v_i \in V$, $\text{MFP}(v_i^0) = \ell^i_0 = (1, \nu(v_i))$ and for all $v_i \in V \setminus I$, if $v_i^1$ is the predecessor of $v_i^0$ in $G'$, then $\text{MFP}(v_i^0) = \ell^i_0 = (\nu(v_i^1), 1)$.

**Corollary 1.** Let $\alpha = (C, \nu)$ be an instance of MCV with $C = (G, I, v_0, \tau)$ and $G = (V, E)$. Let $\alpha' = (G', D, M, v_1^0, (1, 1))$ be the instance of MFP as constructed in Section 4.1.

\[ \nu(v_0) = 1 \iff \text{MFP}(v_0^0) = (1, 1). \]

**Theorem 2.** MFP is $\mathcal{P}$-complete.

**Proof.** A polynomial time algorithm for MFP is given in [7]. MCV is shown $\mathcal{P}$-complete in [4]. It is easy to see that the above reduction is computable in log space. Hence MFP is $\mathcal{P}$-complete. □

5. AN ALGORITHM FOR GMR

Algorithm [1] is an algorithm for deciding GMR.

**Algorithm 1** Algorithm for GMR

1: procedure GMRA($G, A, L, v_0^\phi, a_{\phi}, l_0^\phi$)
2: $\text{temp} \leftarrow 1$
3: for $i \leftarrow 1, n$ do
4: if $v_i^\phi$ is reachable from $v_0^\phi$ then
5: $\text{temp} \leftarrow \text{temp} \land l_i$
6: end if
7: end for
8: if $\text{temp} = l_0^\phi$ then
9: return True
10: else
11: return False
12: end if
13: end procedure

The observation below is a direct consequence of the above algorithm.

**Lemma 2.** Let $\alpha = (G, A, L, v_0^\phi, a_{\phi}, l_0^\phi)$ be an instance of GMR. Let $R_\alpha = \{ l_i \mid v_i^1 \text{ is reachable from } v_0^\phi \}$. Then Algorithm 1 returns true $\iff \bigwedge_{l_i \in R_\alpha} l_i = l_0^\phi$.

**Lemma 3.** GMR is computable in non-deterministic log space.

**Proof.** Variables $\text{temp}$ and $i$ take $O(\log |L|)$ space. Since graph reachability takes up only non-deterministic log space [8 Example 2.10 on p. 48]), Line 4 takes non-deterministic $O(\log |V|) = O(\log (|A| \cdot |L|))$ space. So, GMR is computable in non-deterministic log space. □

6. MOP IS NL-COMPLETE

In this section, we give a log space reduction from MOP to GMR. Since GMR is non-deterministic log space computable, this implies that MOP can also be computed in non-deterministic log space.

6.1 A reduction from MOP to GMR

Given an instance $\alpha = (G, D, M, v_0, l_0)$ of MOP with $G = (V, E, v_s, v_t)$, $D = (L, \setminus A, F)$ and $L = \{ l_1, l_2, \ldots, l_m \}$. Construct an instance of GMR $\alpha' = (G', A, L, v_0^0, v_1^0, l_0)$ as follows

- $A = \{ v_i^0 \mid v_i \in V(G) \} \cup \{ v_i^j \mid v_i \in V(G) \}$
- $G' = (V', E')$ where
  - $V' = \{ v_i^0 \mid v_i \in A, l_j \in L \}$
  - $E'$ is defined as follows

  * For each vertex $v_i \in V, l_j \in L$, if $f_i(l_j) = l_k$, add the edge $(v_i^0, v_i^1)$ to $E'$.
  * For each edge $(v_i^0, v_j^0) \in E, l_k \in L$, add the edge $(v_i^0, v_j^0)$ to $E'$.

**Example 1.** Figure 3 shows a Data Flow Graph and a lattice. Let the set $F$ of monotone functions be defined as follows: $F = \{ f_1, f_2, f_3, f_4, f_5, f_6 \}$ where

- $f_1 = \{ (l_1, l_1), (l_2, l_4), (l_3, l_4), (l_4, l_5), (l_5, l_5) \}$
- $f_2 = \{ (l_1, l_1), (l_2, l_5), (l_3, l_5), (l_4, l_5), (l_5, l_5) \}$
- $f_3 = \{ (l_1, l_2), (l_2, l_5), (l_3, l_5), (l_4, l_5), (l_5, l_5) \}$
- $f_4 = \{ (l_1, l_1), (l_2, l_5), (l_3, l_5), (l_4, l_5), (l_5, l_5) \}$
- $f_5 = \{ (l_1, l_2), (l_2, l_5), (l_3, l_5), (l_4, l_5), (l_5, l_5) \}$
- $f_6 = \{ (l_1, l_1), (l_2, l_4), (l_3, l_4), (l_4, l_4), (l_5, l_5) \}$

**Figure 2** shows the corresponding product graph.

6.2 Proof of correctness

**Lemma 4.** Let $\alpha = (G, D, M, v_0, l_0)$ be an instance of MOP with $G = (V, E, v_s, v_t)$ and $D = (L, \setminus A, F)$. Let $\alpha' = (G', A, L, v_0^0, v_1^0, l_0)$ be an instance of GMR as constructed in Section 6.1. Then for all $v_i \in V$, there exists a $v_s$-$v_i$ path $p$ in $G$ such that $f_p(1) = l_j \iff v_i^1$ is reachable from $v_i^0$ in $G'$.

**Proof.**

- **If part:** Let $v_i$ be an arbitrary vertex in $V$. Let $p$ be a $v_s$-$v_i$ path in $G$ and let $f_p(1) = l_j$ for some $l_j \in L$. We prove the if part by induction on $\text{len}(p)$.
  
  **Base case:** Let $\text{len}(p) = 0$. i.e. $v_i = v_s$. Then $l_j = f_p(1) = f_s(1)$. Then $(v_i^0, v_i^1) \in E'$. So, $v_i^1$ is reachable from $v_i^0$ in $G'$.

  **Inductive step:** Let $\text{len}(p) = k$ and let the if part be true for all paths from $v_s$ in $G$ with length less than $k$. Let $p = v_s, \ldots, v_i, v_t$ for some $v_i \in V$. Let $p'$ be the path $p$ with $v_i$ excluded i.e. $p = p' \cdot v_i$ where $i$ is the path concatenation operator. Therefore, $p'$ is a path from $v_s$ with length $k - 1$. Let $f_{p'}(1) = l_j'$ and $f_{l_j'}(l_j) = l_j$ for some $l_j \in L$. By induction hypothesis, $v_i^1$ is reachable from $v_i^0$ in $G'$. By construction of $E'$, $(v_i^0, v_i^1, (v_i^1, v_i^0)) \in E'$. So, $v_i^1$ is reachable from $v_i^0$ in $G'$.

  **So, the if part is true for all $v_s$-$v_i$ paths in $G$.**

- **Only if part:** Let $v_i^1$ is reachable from $v_i^0$ in $G'$. Let $p$ be a path in $G'$ from $v_i^0$ to $v_i^1$. It is easy to show that length of $p$ is odd. So, we prove the only if part by induction on $\text{len}(p)$ where $\text{len}(p)$ is odd.
  
  **Base case:** Let $\text{len}(p) = 1$. i.e. $(v_0^0, v_1^0) \in E'$. By construction of $E'$, $v_i = v_s$ and $f_1(1) = l_j$. So, there exists a trivial $v_s$-$v_i$ path $q$ in $G$, the path having only one node $v_s$, such that $f_q(1) = l_j$. 
It is easy to see that the above proofs do not use the monotonicity of the data flow framework. So, the $N\mathcal{L}$-completeness result holds even if the data flow framework is not monotone.
A lattice representation of the lattice.

Algorithm 2 Converts a lattice given as a covering relation to a lattice as a poset

1: procedure CovRel-to-Poset\((L, \prec)\)
2: \hspace{1em} for \(i \leftarrow 1, n\) do
3: \hspace{2em} for \(j \leftarrow 1, n\) do
4: \hspace{3em} Set \(l_i \leq l_j\) to value False
5: \hspace{2em} end for
6: \hspace{1em} end for
7: \hspace{1em} for \(i \leftarrow 1, n\) do
8: \hspace{2em} for \(j \leftarrow 1, n\) do
9: \hspace{3em} if \(l_j\) is reachable from \(l_i\) then
10: \hspace{4em} Set \(l_i \leq l_j\) to value True
11: \hspace{3em} end if
12: \hspace{2em} end for
13: \hspace{1em} end for
14: end procedure

A lattice can be represented as a poset \((L, \leq)\) as a covering relation for a poset \((L, \prec)\) or an algebraic structure \((L, \lor, \land)\). In this section, we give a non-deterministic log space algorithm to convert a lattice given as a poset or a covering relation to a lattice as an algebraic structure. This makes the completeness result of MOP w.r.t. the class \(\mathcal{N} \mathcal{L}\) independent of the particular representation of the lattice.

A covering relation \((L, \prec)\) of a poset can be viewed as a graph where \(L\) is the set of vertices and \(\prec\) is the set of edges. So, reachability is defined as it is done for a graph.

Algorithm 3 Converts a lattice given as a poset to a lattice as an algebraic structure

1: procedure Poset-to-AlgStr\((L, \leq)\)
2: \hspace{1em} for \(i \leftarrow 1, n\) do
3: \hspace{2em} for \(j \leftarrow 1, n\) do
4: \hspace{3em} Set \(l_i \lor l_j\) to value 1
5: \hspace{3em} Set \(l_i \land l_j\) to value 0
6: \hspace{2em} end for
7: \hspace{1em} end for
8: \hspace{1em} for \(i \leftarrow 1, n\) do
9: \hspace{2em} for \(j \leftarrow 1, n\) do
10: \hspace{3em} for \(k \leftarrow 1, n\) do
11: \hspace{4em} if \(l_i \leq l_k\) and \(l_j \leq l_k\) then
12: \hspace{5em} if \(l_k \leq l_i \lor l_j\) then
13: \hspace{6em} Set \(l_i \lor l_j\) to value \(l_k\)
14: \hspace{5em} end if
15: \hspace{4em} else if \(l_k \leq l_i\) and \(l_k \leq l_j\) then
16: \hspace{5em} if \(l_i \land l_j\) \(\leq l_k\) then
17: \hspace{6em} Set \(l_i \land l_j\) to value \(l_k\)
18: \hspace{5em} end if
19: \hspace{4em} end if
20: end if
21: end for
22: end for
23: end procedure

A lattice can be represented as a poset \((L, \leq)\) as a covering relation for a poset \((L, \prec)\) or an algebraic structure \((L, \lor, \land)\). In this section, we give a non-deterministic log space algorithm to convert a lattice given as a poset or a covering relation to a lattice as an algebraic structure. This makes the completeness result of MOP w.r.t. the class \(\mathcal{N} \mathcal{L}\) independent of the particular representation of the lattice.

Algorithm 2 (resp. Algorithm 3) converts a lattice given as a covering relation of a poset (resp. a poset) to the lattice as a poset (resp. an algebraic structure). The composition of the two algorithms converts a lattice given as a covering relation of a poset to the lattice as an algebraic structure.

**Lemma 5.** Given a poset \((L, \leq)\) or a covering relation \((L, \prec)\) representation of a lattice, its algebraic structure representation \((L, \lor, \land)\) can be computed in non-deterministic log space.

**Proof.** Line 9 of Algorithm 2 takes non-deterministic log space since \(GR\) takes non-deterministic log space [8] Example 2.10 on p. 48. All other lines of the two algorithms take at most log space. So, Algorithm 2 takes non-deterministic log space while Algorithm 3 takes log space. The composition of the two algorithms takes non-deterministic log space [11] Theorem 8.23 on p. 324. So, the conversions can be done in non-deterministic log space.

It may be noted that the lattice in Section 4 is of constant size. So, it can be converted to a poset or a covering relation in constant time. So, the completeness result of MOP w.r.t. the class \(\mathcal{N} \mathcal{L}\) is also independent of the particular representation of the lattice.

SIGACT-SIGPLAN symposium on Principles of Programming Languages, POPL ’73, pages 194–206, New York, NY, USA, 1973. ACM.

[8] C. H. Papadimitriou. Computational Complexity. Addison Wesley, 1993.

[9] T. Reps. On the sequential nature of interprocedural program-analysis problems. Acta Informatica, 33:739–757, 1996. 10.1007/BF03036473.

[10] T. Reps, S. Horwitz, and M. Sagiv. Precise interprocedural dataflow analysis via graph reachability. In Proceedings of the 22nd ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages, POPL ’95, pages 49–61, New York, NY, USA, 1995. ACM.

[11] M. Sipser. Introduction to the Theory of Computation. Course Technology, 2 edition, Feb. 2005.

[12] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. Pacific Journal of Mathematics, 5(2):285–309, 1955.