A Fix-Point Characterization of Herbrand Equivalence of Expressions in Data Flow Frameworks

Jasine Babu[1(✉)], Karunakaran Murali Krishnan[2], and Vineeth Paleri[2]

[1] Indian Institute of Technology Palakkad, Palakkad, India
jasine@iitpkd.ac.in
[2] National Institute of Technology Calicut, Kozhikode, India
{kmurali,vpaleri}@nitc.ac.in

Abstract. Computing Herbrand equivalences of terms in data flow frameworks is well studied in program analysis. While algorithms use iterative fix-point computation on some abstract lattice of expressions relevant to the flow graph, the definition of Herbrand equivalences is based on an equivalence over all program paths formulation, on the (infinite) set of all expressions. The aim of this paper is to develop a lattice theoretic fix-point formulation of Herbrand equivalence on a concrete lattice defined over the set of all terms constructible from variables, constants and operators of a program. This new formulation makes explicit the underlying lattice framework as well as the monotone function involved in computing Herbrand equivalences. We introduce the notion of Herbrand congruence and define an (infinite) concrete lattice of Herbrand congruences. Herbrand equivalence is defined as the maximum fix-point of a composite transfer function defined over an appropriate product lattice of the above concrete lattice. We then reformulate the traditional meet over all paths definition in our lattice theoretic framework. and prove that the maximum fix-point (MFP) and the meet-over-all-paths (MOP) formulations coincide as expected.

Keywords: Herbrand equivalence · Data flow framework · Fix-point

1 Introduction

Data flow frameworks are abstract representations of programs, used in program analysis and compiler optimizations [1,2]. As detection of semantic equivalence of expressions at program points is unsolvable [3], algorithms try to detect a weaker, syntactic notion of equivalence called *Herbrand equivalence* [4–8]. Herbrand equivalence treats operators as uninterpreted functions, and expressions obtained by applying the same operator on equivalent operands are treated equivalent.

Kildall [9] used abstract interpretation [10] to compute Herbrand equivalences at program points using an iterative fix point algorithm over a meet semi-lattice

Md. A. Khan and A. Manuel (Eds.): ICLA 2019, LNCS 11600, pp. 160–172, 2019.
https://doi.org/10.1007/978-3-662-58771-3_15

[2,3,11]. Several algorithms for computation of Herbrand equivalence of program expressions are known [4,5,7,8,12,13]. Most of these algorithms use iterative fix-point computation on an abstract lattice defined over a *working set* of expressions relevant to the program. It is known that the working set needs to include certain non-program expressions as well [9].

The theoretical question of describing the concrete lattice of all expression equivalences and the monotone function whose fix point defines the Herbrand equivalence of expressions at program points seems to be unaddressed in the literature. This is possibly due the fact that the traditional definition of Herbrand equivalence of expressions (see [5, p. 393]) is based on an equivalence over all program paths formulation, rather than a lattice based formulation. In this work, we axiomatize the notion of congruence of all expressions over the variables, constants and operators in a program. We then define a concrete lattice of all congruences. We show that this lattice is complete, which ensures that every monotone function f over the lattice indeed has a maximum fix-point [14], though the lattice has infinite height.

Given a data flow framework, we define transfer functions and indefinite assignments as certain monotone functions on the lattice described above. By a standard product lattice construction, we then define a composite transfer function [15,16], which is monotone over the product lattice. We define the Herbrand equivalence at each program point as component maps (projections) of the maximum fix-point of this composite transfer function. Finally, to validate the new formulation against the existing definition of Herbrand equivalence, we re-formulate the standard definition of Herbrand equivalence in [5] in our lattice framework and show it to be equivalent to the fix-point formulation developed in this paper.

Many of the standard proofs for the equivalence of fix-point and meet over all paths formulations assume that the lattice is either of finite height or that every chain has finite height [1,3,16], where equivalence holds whenever the transfer function under consideration is a meet-morphism. Even though chains are of infinite height, the transfer function being a complete-meet-morphism guarantees that the equivalence still holds true for the formulation presented in this paper. A proof of this equivalence is presented in Sect. 8.

2 Terminology

Let C be a set of constants and X be a set of variables. For simplicity, we assume that the set of operators $Op = \{+\}$. (More operators can be added without any difficulty). The set of all terms over $C \cup X$, $\mathcal{T} = \mathcal{T}(X, C)$ is defined by $t ::= c \mid x \mid (t + t)$, with $c \in C$ and $x \in X$. (Parenthesis is avoided when there is no confusion.) Let $\mathcal{P}$ be a partition of $\mathcal{T}$. Let $[t]_{\mathcal{P}}$ (or simply $[t]$ when there is no confusion) denote the equivalence class from $\mathcal{P}$ containing the term $t \in \mathcal{T}$. If $t' \in [t]_{\mathcal{P}}$, we write $t \cong_{\mathcal{P}} t'$ (or simply $t \cong t'$). For any $A \subseteq \mathcal{T}$, $A(x)$ denotes the set of all terms in A in which the variable x appears and $\overline{A}(x)$ denotes the set of all terms in A in which x does not appear. In particular, for any $x \in X$, $\mathcal{T}(x)$ is the set of all terms containing the variable x and $\overline{\mathcal{T}}(x) = \mathcal{T} \setminus \mathcal{T}(x)$.

Definition 1 (Substitution). *For $t, \alpha \in \mathcal{T}$, $x \in X$, substitution of x with α in t, denoted by $t[x \leftarrow \alpha]$ is defined by: (1) If $t = x$, then $t[x \leftarrow \alpha] = \alpha$. (2) If $t \notin \mathcal{T}(x)$, $t[x \leftarrow \alpha] = t$. (3) If $t = t_1 + t_2$ then $t[x \leftarrow \alpha] = t_1[x \leftarrow \alpha] + t_2[x \leftarrow \alpha]$.*

For proofs of statements left unproven in the main text and proofs of some elementary results of lattice theory which are used in the main text, the reader may refer to the preprint [17].

3 Congruences of Terms

Definition 2 (Congruence of Terms). *Let $\mathcal{P}$ be a partition of $\mathcal{T}$. $\mathcal{P}$ is a Congruence (of terms) if the following conditions hold:*

1. *For $t, t', s, s' \in \mathcal{T}$, $t' \cong t$ and $s' \cong s$ if and only if $t' + s' \cong t + s$. (Congruences respect operators).*
2. *For any $c \in C$, $t \in \mathcal{T}$, if $t \cong c$ then either $t = c$ or $t \in X$. (The only non-constant terms that are allowed to be congruent to a constant are variables).*

Given a data flow framework, we will associate a congruence with each program point. Each iteration refines the present congruence at each program point till a fix-point is reached. The Herbrand equivalence at each program point will be defined as this fix-point congruence.

Definition 3. *The set of all congruences over $\mathcal{T}$ is denoted by $\mathcal{G}(\mathcal{T})$.*

We define a binary *confluence operation* on the set of congruences, $\mathcal{G}(\mathcal{T})$.

Definition 4 (Confluence). *Let $\mathcal{P}_1 = \{A_i\}_{i \in I}$ and $\mathcal{P}_2 = \{B_j\}_{j \in J}$ be two congruences. For all $i \in I$ and $j \in J$, define $C_{i,j} = A_i \cap B_j$. The confluence of $\mathcal{P}_1$ and $\mathcal{P}_2$ is defined by: $\mathcal{P}_1 \wedge \mathcal{P}_2 = \{C_{i,j} : i \in I, j \in J, C_{i,j} \neq \emptyset\}$.*

Theorem 5. *If $\mathcal{P}_1$ and $\mathcal{P}_2$ are congruences, then $\mathcal{P}_1 \wedge \mathcal{P}_2$ is a congruence.*

We next define an ordering on the set $\mathcal{G}(\mathcal{T})$ and extend it to a complete lattice.

Definition 6 (Refinement of a Congruence). *Let $\mathcal{P}, \mathcal{P}'$ be congruences over $\mathcal{T}$. We say $P \preceq P'$ (read P is a refinement of P') if for each equivalence class $A \in \mathcal{P}$, there exists an equivalence class $A' \in \mathcal{P}'$ such that $A \subseteq A'$.*

Definition 7. *The partition in which each term in $\mathcal{T}$ belongs to a different class is defined as: $\bot = \{\{t\} : t \in \mathcal{T}\}$.*

The following observation is a direct consequence of the definition of $\bot$.

Observation 8. *$\bot$ is a congruence. Moreover for any $\mathcal{P} \in \mathcal{G}(\mathcal{T})$, $\bot \wedge \mathcal{P} = \bot$.*

Lemma 9. *Every non-empty subset of $(\mathcal{G}(\mathcal{T}), \preceq)$ has a greatest lower bound.*

Next, we extend $(\mathcal{G}(\mathcal{T}), \preceq)$ by artificially adding a top element $\top$, so that the greatest lower bound of the empty set is also well defined.

Definition 10. *The lattice* $(\overline{\mathcal{G}(\mathcal{T})}, \preceq, \bot, \top)$ *is defined over the set* $\overline{\mathcal{G}(\mathcal{T})} = \mathcal{G}(\mathcal{T}) \cup \{\top\}$ *with* $\mathcal{P} \preceq \top$ *for each* $\mathcal{P} \in \overline{\mathcal{G}(\mathcal{T})}$. *In particular,* $\top$ *is the greatest lower bound of* $\emptyset$ *and* $\top \wedge \mathcal{P} = \mathcal{P}$ *for every* $\mathcal{P} \in \overline{\mathcal{G}(\mathcal{T})}$.

Hereafter, we will be referring to the element $\top$ as a congruence. Combining Lemma 9, Definition 10 and standard results in lattice theory, we get:

Theorem 11. $(\overline{\mathcal{G}(\mathcal{T})}, \preceq, \bot, \top)$ *is a complete lattice.*

Definition 12 (Infimum). *Let* $S = \{\mathcal{P}_i\}_{i \in I}$ *be an arbitrary collection of congruences in* $\overline{\mathcal{G}(\mathcal{T})}$ *(*S *may be empty or may contain* $\top$*). The infimum of* S*, denoted by* $\bigwedge S$ *or* $\bigwedge_{i \in I} \mathcal{P}_i$*, is defined as the greatest lower bound of the set* $\{\mathcal{P}_i\}_{i \in I}$.

Remark 13. The results in this paper only assumes the meet-completeness of $(\overline{\mathcal{G}(\mathcal{T})}, \preceq, \bot, \top)$ and the existence of a top element. Though the lattice is also join-complete, our proofs do not rely on this property.

4 Transfer Functions

A *transfer function* describes the effect of an assignment on a congruence.

Definition 14 (Transfer Functions). *Let* $y \in X$ *and* $\beta \in \overline{\mathcal{T}}(y)$. *(Note that* y *does not appear in* β*). The transfer function* $f_{y=\beta}$ *transforms a congruence* $\mathcal{P} = \{A_i\}_{i \in I}$ *to* $f_{y=\beta}(\mathcal{P})$*, a collection of subsets of* $\mathcal{T}$*, given by the following:*

- *If* $\mathcal{P} = \{A_i\}_{i \in I}$*, then let* $B_i = \{t \in \mathcal{T} : t[y \leftarrow \beta] \in A_i\}$*, for each* $i \in I$.
- $f_{y=\beta}(\mathcal{P}) = \{B_i : i \in I,\ B_i \neq \emptyset\}$.

See Fig. 1 for an example.

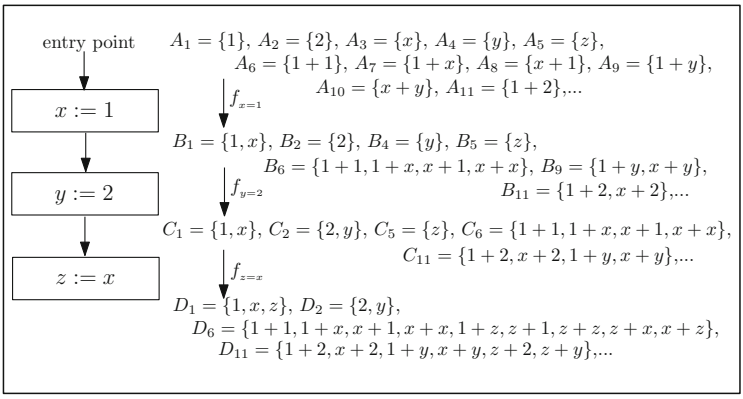

Fig. 1. Application of transfer functions

Note that statements of the form $y := y + c$ can be transformed to the form permitted by Definition 14. We write $f(\mathcal{P})$ instead of $f_{y=\beta}(\mathcal{P})$ to avoid cumbersome notation.

Theorem 15. *If $\mathcal{P}$ is a congruence, then for any $y \in X$, $\beta \in \overline{\mathcal{T}}(y)$, $f_{y=\beta}(\mathcal{P})$ is a congruence.*

Next, we extend the definition of transfer functions to $(\overline{\mathcal{G}(\mathcal{T})}, \preceq, \bot, \top)$.

Definition 16. *Let $y \in X$ and $\beta \in \overline{\mathcal{T}}(y)$. Let $\mathcal{P} \in \overline{\mathcal{G}(\mathcal{T})}$. The extended transfer function $\overline{f}_{y=\beta}(\mathcal{P}) : \overline{\mathcal{G}(\mathcal{T})} \longrightarrow \overline{\mathcal{G}(\mathcal{T})}$ transforms $\mathcal{P}$ to $\overline{f}_{y=\beta}(\mathcal{P}) \in \overline{\mathcal{G}(\mathcal{T})}$ given by $\overline{f}_{y=\beta}(\mathcal{P}) = f_{y=\beta}(\mathcal{P})$ for all $\mathcal{P} \in \mathcal{G}(\mathcal{T})$ and $\overline{f}_{y=\beta}(\top) = \top$.*

We will write $f_{y=\beta}(\mathcal{P})$ (or $f(\mathcal{P})$) instead of $\overline{f}_{y=\beta}(\mathcal{P})$ and call it a transfer function.

We next show that transfer functions are complete-meet-morphisms over the complete lattice $(\overline{\mathcal{G}(\mathcal{T})}, \preceq, \bot, \top)$. Let $f = f_{y=\beta}$, where $y \in X$, $\beta \in \overline{\mathcal{T}}(y)$. For arbitrary collections of congruences $S \subseteq \overline{\mathcal{G}(\mathcal{T})}$, The notation $f(S)$ denotes the set $\{f(s) : s \in S\}$.

Theorem 17. *f is a complete-meet-morphism. That is, for any $\emptyset \neq S \subseteq \overline{\mathcal{G}(\mathcal{T})}$, $f(\bigwedge S) = \bigwedge f(S)$.*

5 Indefinite Assignment

Indefinite (or non-deterministic) assignments model input statements in programs.

Definition 18. *Let $y \in X$. The transfer function $f_{y=*}$ transforms a congruence $\mathcal{P} \in \mathcal{G}(\mathcal{T})$ to $f(\mathcal{P}) = f_{y=*}(\mathcal{P})$, a collection of subsets of $\mathcal{T}$ given by: for every $t, t' \in \mathcal{T}$, $t \cong_{f(\mathcal{P})} t'$ if and only if both the following conditions are satisfied: (1) $t \cong_{\mathcal{P}} t'$. (2) For every $\beta \in \mathcal{T} \setminus \mathcal{T}(y), t[y \leftarrow \beta] \cong_{\mathcal{P}} t'[y \leftarrow \beta]$.*

Theorem 19. *If $\mathcal{P}$ is a congruence, then for any $y \in X$, $f_{y=*}(\mathcal{P})$ is a congruence.*

We write $\bigwedge_{\beta \in \overline{\mathcal{T}}(y)} f_{y=\beta}(\mathcal{P})$ to denote the set $\bigwedge\{f_{y=\beta}(\mathcal{P}) : \beta \in \overline{\mathcal{T}}(y)\}$. The next theorem shows that each indefinite assignment may be expressed as a confluence of (an infinite collection of) transfer function operations.

Theorem 20. *If $\mathcal{P}$ is a congruence, then for any $y \in X$,*
$f_{y=*}(\mathcal{P}) = \mathcal{P} \wedge \left(\bigwedge_{\beta \in \overline{\mathcal{T}}(y)} f_{y=\beta}(\mathcal{P})\right)$.

We extend indefinite assignments to $(\overline{\mathcal{G}(\mathcal{T})}, \preceq, \bot, \top)$.

Definition 21. *Let $y \in X$ and $\mathcal{P} \in \overline{\mathcal{G}(\mathcal{T})}$. The extended transfer function $\overline{f}_{y=*}(\mathcal{P}) : \overline{\mathcal{G}(\mathcal{T})} \mapsto \overline{\mathcal{G}(\mathcal{T})}$ transforms $\mathcal{P}$ to $\overline{f}_{y=*}(\mathcal{P}) \in \overline{\mathcal{G}(\mathcal{T})}$ given by: $\overline{f}_{y=*}(\mathcal{P}) = f_{y=*}(\mathcal{P})$ for all $\mathcal{P} \in \mathcal{G}(\mathcal{T})$, and $\overline{f}_{y=*}(\top) = \top$.*

We will write $f_{y=*}(\mathcal{P})$ instead of $\overline{f}_{y=*}(\mathcal{P})$ to simplify notation.

Theorem 22. *$f_{y=*}$ is a complete-meet-morphism. That is, for any $\emptyset \neq S \subseteq \overline{\mathcal{G}(\mathcal{T})}$, $f_{y=*}(\bigwedge S) = \bigwedge f_{y=*}(S)$, where $f_{y=*}(S) = \{f_{y=*}(s) : s \in S\}$.*

We show that indefinite assignments can be characterized in terms of just three congruences (instead of dealing with infinitely many as in Theorem 20).

Theorem 23. *Let $\mathcal{P} \in \mathcal{G}(\mathcal{T})$ and let $c_1, c_2 \in C$, $c_1 \neq c_2$. Then, for any $y \in X$, $f_{y=*}(\mathcal{P}) = \mathcal{P} \wedge f_{y \leftarrow c_1}(\mathcal{P}) \wedge f_{y \leftarrow c_2}(\mathcal{P})$.*

6 Data Flow Analysis Frameworks

We next formalize the notion of a data flow framework and apply the formalism developed above to characterize Herbrand equivalence at each point in a program.

Definition 24. *A control flow graph $G(V, E)$ is a directed graph over the vertex set $V = \{1, 2, \ldots, n\}$ for some $n \geq 1$ satisfying the following properties:*

- *$1 \in V$ is called the entry point and has no predecessors.*
- *Every vertex $i \in V$, $i \neq 1$ is reachable from 1 and has at least one predecessor and at most two predecessors.*
- *Vertices with two predecessors are called confluence points.*
- *Vertices with a single predecessor are called (transfer) function points.*

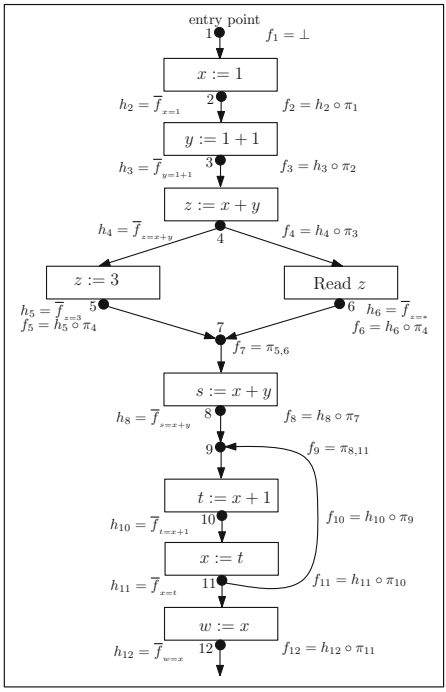

Fig. 2. Component maps of the composite transfer function

Definition 25. *A data flow framework over $\mathcal{T}$ is a pair $D = (G, F)$, where $G(V, E)$ is a control flow graph on the vertex set $V = \{1, 2, \ldots, n\}$ and F is a mapping from the set of function points in V to the set of transfer functions over $\mathcal{G}(\mathcal{T})$. The transfer function associated with function point i will be denoted as h_i.*

Data flow frameworks can be used to represent programs. An example is given in Fig. 2. In the sections that follow, we will use h_i to denote the extended transfer function $\overline{h}_i$ (see Definitions 16 and 21) without further explanation.

7 Herbrand Equivalence

Let $D = (G, F)$ be a data flow framework over $\mathcal{T}$. In the following, we will define the Herbrand Congruence function $H_D : V(G) \mapsto \overline{\mathcal{G}(\mathcal{T})}$. For each vertex $i \in V(G)$, the congruence $H_D(i)$ will be called the *Herbrand Congruence* associated with the vertex i of the data flow framework D. The function H_D will be defined as the maximum fix-point of a complete-meet-morphism $f_{_D} : \overline{\mathcal{G}(\mathcal{T})}^n \mapsto \overline{\mathcal{G}(\mathcal{T})}^n$. The function $f_{_D}$ will be called the *composite transfer function* associated with the data flow framework D.

Definition 26 (Product Lattice). *Let n be a positive integer. The product lattice, $(\overline{\mathcal{G}(\mathcal{T})}^n, \preceq_n, \bot_n, \top_n)$ is defined as follows: for $\overline{\mathcal{P}} = (\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n)$, $\overline{\mathcal{Q}} = (\mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_n) \in \overline{\mathcal{G}(\mathcal{T})}^n$, $\overline{\mathcal{P}} \preceq_n \overline{\mathcal{Q}}$ if $\mathcal{P}_i \preceq \mathcal{Q}_i$ for each $1 \leq i \leq n$, $\bot_n = (\bot, \bot, \ldots, \bot)$ and $\top_n = (\top, \top, \ldots, \top)$.*

For $S \subset \overline{\mathcal{G}(\mathcal{T})}^n$, the notation $\bigwedge S$ will be used to denote the greatest lower bound of S in the product lattice.

By Theorem 11, and standard results in lattice theory, we have:

Theorem 27. *The product lattice satisfies the following properties:*

1. *$(\overline{\mathcal{G}(\mathcal{T})}^n, \preceq_n, \bot_n, \top_n)$ is a complete lattice.*
2. *If $\tilde{S} \subseteq \overline{\mathcal{G}(\mathcal{T})}^n$ is non-empty, with $\tilde{S} = S_1 \times S_2 \times \cdots \times S_n$, where $S_i \subseteq \overline{\mathcal{G}(\mathcal{T})}$ for $1 \leq i \leq n$. Then $\bigwedge \tilde{S} = (\bigwedge S_1, \bigwedge S_2, \ldots, \bigwedge S_n)$.*

As preparation for defining the composite transfer function, we introduce the following functions:

Definition 28 (Projection Maps). *Let n be a positive integer. For each $i \in \{1, 2, \ldots, n\}$,*

- *The projection map to the i^{th} co-ordinate, $\pi_i : \overline{\mathcal{G}(\mathcal{T})}^n \mapsto \overline{\mathcal{G}(\mathcal{T})}$ is defined by $\pi_i(\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n) = \mathcal{P}_i$ for any $(\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n) \in \overline{\mathcal{G}(\mathcal{T})}^n$.*
- *The confluence map $\pi_{i,j} : \overline{\mathcal{G}(\mathcal{T})}^n \mapsto \overline{\mathcal{G}(\mathcal{T})}$ is defined by $\pi_{i,j}(\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n) = \mathcal{P}_i \wedge \mathcal{P}_j$ for any $(\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n) \in \overline{\mathcal{G}(\mathcal{T})}^n$.*

In addition to the above functions, we will also use the constant map which maps each element in $\overline{\mathcal{G}(\mathcal{T})}^n$ to $\bot$. The following observation is a consequence of standard results in lattice theory.

Observation 29. *Constant maps, projection maps and confluence maps are complete-meet-morphisms.*

For each $k \in V(G)$, $\text{Pred}(k)$ denotes the set of predecessors of the vertex k in the control flow graph G.

Definition 30 (Composite Transfer Function). *Let $D = (G, F)$ be a data flow framework over $\mathcal{T}$. For each $k \in V(G)$, define the component map $f_k : \overline{\mathcal{G}(\mathcal{T})}^n \mapsto \overline{\mathcal{G}(\mathcal{T})}$ as follows:*

1. *If $k = 1$, the entry point, then $f_k = \bot$. (f_1 is the constant function that always returns the value $\bot$).*
2. *If k is a function point with $\text{Pred}(k) = \{j\}$, then $f_k = h_k \circ \pi_j$, where h_k is the (extended) transfer function corresponding to the function point k, and π_j the projection map to the j^{th} coordinate as defined in Definition 28.*
3. *If k is a confluence point with $\text{Pred}(k) = \{i, j\}$, then $f_k = \pi_{i,j}$, where $\pi_{i,j}$ is the confluence map as defined in Definition 28.*

The composite transfer function of D is defined to be the unique function f_D satisfying $\pi_k \circ f_D = f_k$ for each $k \in V(G)$.

The purpose of defining f_D is the following. Suppose we have associated a congruence with each program point in a data flow framework. Then f_D specifies how a simultaneous and synchronous application of all the transfer functions/confluence operations at the respective program points modifies the congruences at each program point. The definition of f_D conservatively sets the confluence at the entry point to $\bot$, treating terms in $\mathcal{G}(\mathcal{T})$ to be inequivalent to each other at the entry point. See Fig. 2 for an example. The following observation is a direct consequence of the above definition.

Observation 31. *The composite transfer function f_D (Definition 30) satisfies the following properties:*

1. *If $k = 1$, the entry point, then $\pi_k \circ f_D = \bot$.*
2. *If k is a function point with $\text{Pred}(k) = \{j\}$, then $f_k = \pi_k \circ f_D = h_k \circ \pi_j$, where h_k is the (extended) transfer function corresponding to the function point k.*
3. *If k is a confluence point with $\text{Pred}(k) = \{i, j\}$, then $f_k = \pi_k \circ f_D = \pi_{i,j}$.*

The following lemma is a consequence of Observation 31.

Lemma 32. *Let $D = (G, F)$ be a data flow framework over $\mathcal{T}$ and $k \in V(G)$. Let $S = \{f_D(\top_n), f_D^2(\top_n), \ldots\}$, where f_D is the composite transfer function of D.*

1. *If $k = 1$, the entry point, then $\pi_k \circ f_D^l(\top_n) = \bot$ for all $l \geq 1$, hence $\pi_k(\bigwedge S) = \bot$.*

2. If k is a function point with $\text{Pred}(k) = \{j\}$*, then for all* $l \geq 1$*,*

$$\begin{aligned}(\pi_k \circ f_{D}^{l})(\top_n) &= (\pi_k \circ f_{D})(f_{D}^{l-1}(\top_n)) \\ &= (h_k \circ \pi_j \circ f_{D}^{l-1})(\top_n)\end{aligned}$$

3. If k is a confluence point with $\text{Pred}(k) = \{i, j\}$*, then for all* $l \geq 1$*,*

$$\begin{aligned}(\pi_k \circ f_{D}^{l})(\top_n) &= (\pi_k \circ f_{D})(f_{D}^{l-1}(\top_n)) \\ &= (\pi_{i,j})(f_{D}^{l-1}(\top_n)) \\ &= \big((\pi_i \circ f_{D}^{l-1})(\top_n)\big) \wedge \big((\pi_j \circ f_{D}^{l-1})(\top_n)\big)\end{aligned}$$

By standard facts in lattice theory, we have:

Theorem 33. *The following properties hold for the composite transfer function* f_{D} *(Definition 30):*

1. f_{D} *is monotone, and is a complete-meet-morphism.*
2. *The component maps* $f_k = \pi_k \circ f_{D}$ *are complete-meet-morphisms for all* $k \in \{1, 2, \ldots, n\}$*.*
3. f_{D} *has a maximum fix-point.*
4. *If* $S = \{\top, f_{D}(\top_n), f_{D}^{2}(\top_n), \ldots\}$*, then* $\bigwedge S$ *is the maximum fix-point of* f_{D}*.*

The objective of defining Herbrand Congruence as the maximum fix-point of the composite transfer function is possible now.

Definition 34 (Herbrand Congruence). *Given a data flow framework* $D = (G, F)$ *over* $\mathcal{T}$*, the Herbrand Congruence function* $H_D : V(G) \mapsto \overline{\mathcal{G}(\mathcal{T})}$ *is defined as the maximum fix-point of the composite transfer function* f_{D}*. For each* $k \in V(G)$*, the value* $H_D(k) \in \overline{\mathcal{G}(\mathcal{T})}$ *is referred to as the Herbrand Congruence at program point* k*.*

The following is a consequence of Theorem 33 and the definition of Herbrand Congruence.

Observation 35. *For each* $k \in V(G)$*,* $H_D(k) = \bigwedge\{(\pi_k \circ {f_{D}}^{l})(\top_n) : l \geq 0\}$*.*

Proof

$$\begin{aligned}H_D(k) &= \pi_k(\textstyle\bigwedge_n\{{f_{D}}^{l}(\top_n) : l \geq 0\}) \text{ (by Theorem 33)} \\ &= \textstyle\bigwedge\{\pi_k({f_{D}}^{l}(\top_n)) : l \geq 0\} \text{ (because } \pi_k \text{ is a complete-meet-morphism)} \\ &= \textstyle\bigwedge\{(\pi_k \circ {f_{D}}^{l})(\top_n) : l \geq 0\}\end{aligned}$$

□

The definition of Herbrand congruence must be shown to be consistent with the traditional meet-over-all-paths description of Herbrand equivalence of terms in a data flow framework. The next section addresses this issue.

8 MOP Characterization

In this section, we give a meet over all paths characterization for the Herbrand Congruence at each program point. This is essentially a lattice theoretic reformulation of the characterization by Steffen et al. [5, p. 393]. Consider a data flow framework $D = (G, F)$ over $\mathcal{T}$, with $V(G) = \{1, 2, \ldots, n\}$.

Definition 36 (Path). *For any non-negative integer l, a program path (or simply a path) of length l to a vertex $k \in V(G)$ is a sequence $\alpha = (v_0, v_2, \ldots v_l)$ satisfying $v_0 = 1$, $v_l = k$ and $(v_{i-1}, v_i) \in E(G)$ for each $i \in \{1, 2, \ldots l\}$. For each $i \in \{0, 1, \ldots, l\}$, α_i denotes the initial segment of α up to the i^{th} vertex, given by $(v_0, v_1, \ldots, v_i)$.*

Next, we associate a congruence in $\overline{\mathcal{G}(\mathcal{T})}$ with each path in D. The path function captures the effect of application of transfer functions along the path on the initial congruence $\perp$, in the order in which the transfer functions appear along the path.

Definition 37 (Path Congruence). *Let $\alpha = (v_0, v_1, \ldots, v_l)$ be a path of length l to vertex $k \in V(G)$ for some $l \geq 0$. We define:*

1. *When $i = 0$, $m_{\alpha_i} = \perp$.*
2. *If $i > 0$ and $v_i = j$, where $j \in V(G)$ is a function point, then $m_{\alpha_i} = h_j(m_{\alpha_{i-1}})$, where $h_j \in F$ is the extended transfer function associated with the function point j.*
3. *If $i > 0$ and v_i is a confluence point, then $m_{\alpha_i} = m_{\alpha_{i-1}}$.*
4. *The path congruence associated with α, $m_\alpha = m_{\alpha_l}$.*

For $k \in V(G)$ and $l \geq 0$, let $\Phi_l(k)$ denote the set of all paths of length *less than* l from the entry point 1 to the vertex k. In particular, $\Phi_0(k) = \emptyset$, for all $k \in V(G)$. The following observation is a consequence of the definition of $\Phi_l(k)$.

Observation 38. *If $k \in V(G)$ and $l \geq 1$,*

1. *If k is the entry point, then $\Phi_l(k) = \{(1)\}$, the set containing only the path of length zero, starting and ending at vertex 1.*
2. *If k is a function point with $\text{Pred}(k) = \{j\}$, then $\{\alpha_{l-1} : \alpha \in \Phi_l(k)\} = \{\alpha' : \alpha' \in \Phi_{l-1}(j)\} = \Phi_{l-1}(j)$.*
3. *If k is a confluence point with $\text{Pred}(k) = \{i, j\}$, then $\{\alpha_{l-1} : \alpha \in \Phi_l(k)\} = \{\alpha' : \alpha' \in \Phi_{l-1}(i)\} \cup \{\alpha' : \alpha' \in \Phi_{l-1}(j)\} = \Phi_{l-1}(i) \cup \Phi_{l-1}(j)$.*

For $l \geq 0$, we define the congruence $M_l(k)$ to be the meet of all path congruences associated with paths of length less than l from the entry point to vertex k in G. Stated formally,

$$M_l(k) = \bigwedge \{m_\alpha : \alpha \in \Phi_l(k)\}, \text{for } l \geq 0$$

Observation 39. *If $l = 0$, $\Phi_l(k) = \emptyset$ and hence $M_0(k) = \top$, for all $k \in V(G)$. Further, $M_1(1) = \bot$ and $M_1(k) = \top$, for $k \neq 1$. In general, $M_l(k) = \top$ if there are no paths of length less than l from 1 to k in G.*

We define Φ_k to be the set of all paths from vertex 1 to vertex k in G, i.e., $\Phi_k = \bigcup_{l \geq 1} \Phi_l(k)$ and $MOP(k) = \bigwedge\{m_\alpha : \alpha \in \Phi(k)\} = \bigwedge\{M_l(k) : l \geq 0\}$. (The second equality follows from standard results in lattice theory and Observation 39.) The congruence $MOP(k)$ is the meet of all path congruences associated with paths in Φ_k.

Our objective is to prove $MOP(k) = H_D(k)$ for each $k \in \{1, 2, \ldots, n\}$ so that H_D captures the meet over all paths information about equivalence of expressions in $\mathcal{T}$. As noted in the introduction, the proof does not immediately follow from the transfer function being a meet-morphism, as in [1,3,16] since the lattice is neither of finite height nor the chains in the lattice stabilize at finite height.

We begin with the following observations.

Lemma 40. *For each $k \in V(G)$ and $l \geq 1$*

1. *If $k = 1$, the entry point, then $M_l(k) = \bot$.*
2. *If k is a function point with $\text{Pred}(k) = \{j\}$, then $M_l(k) = h_k(M_{l-1}(j))$, where h_k is the (extended) transfer function corresponding to the function point k.*
3. *If k is a confluence point with $\text{Pred}(k) = \{i, j\}$, then $M_l(k) = M_{l-1}(i) \wedge M_{l-1}(j)$.*

Lemma 41. *For each $k \in V(G)$ and $l \geq 0$, $M_l(k) = (\pi_k \circ {f_D}^l)(\top_n)$.*

Finally, we show that the iterative fix-point characterization of Herbrand equivalence and the meet over all paths characterization coincide.

Theorem 42. *Let $D = (G, F)$ be a data flow framework. Then, for each $k \in V(G)$, $MOP(k) = H_D(k)$.*

Proof

$$\begin{aligned} MOP(k) &= \bigwedge\{m_\alpha : \alpha \in \Phi(k)\} \\ &= \bigwedge\{M_l(k) : l \geq 0\} \text{ (by Observation 39)} \\ &= \bigwedge\{(\pi_k \circ {f_D}^l)(\top_n) : l \geq 0\} \text{ (by Lemma 41)} \\ &= H_D(k) \text{ (by Observation 35)} \end{aligned}$$

□

Note that the proof of Observation 35 requires the composite transfer function to be a complete-meet-morphism.

9 Conclusion

We have shown that Herbrand equivalences of terms at program points in a data flow framework can be formulated in terms of the maximum fix-point of a composite transfer function defined over an infinite complete lattice of congruences. The standard definition of Herbrand equivalence [5] is reformulated as a meet over all paths definition in this new lattice framework and is shown to be equivalent to the fix-point formulation. The equivalence of the MFP characterization with the standard formulation provides a theoretical justification for the use of fix-point algorithms used in practice for computing Herbrand equivalences. The new fix-point formulation permits us to view the existing working set based fix-point algorithms as instances of abstract interpretation from the ideal concrete lattice into appropriately defined abstract lattices. We hope that this view can help to make correctness proofs of working set algorithms more transparent.

References

1. Nielson, F., Nielson, H.R., Hankin, C.: Principles of Program Analysis, vol. 1. Springer, Heidelberg (1999). https://doi.org/10.1007/978-3-662-03811-6
2. Aho, A.V., Lam, M.S., Sethi, R., Ullman, J.D.: Compilers: Principles, Techniques, and Tools, vol. 2. Addison-Wesley Reading, Boston (2006)
3. Kam, J.B., Ullman, J.D.: Monotone data flow analysis frameworks. Acta Informatica **7**(3), 305–317 (1977)
4. Rüthing, O., Knoop, J., Steffen, B.: Detecting equalities of variables: combining efficiency with precision. In: Cortesi, A., Filé, G. (eds.) SAS 1999. LNCS, vol. 1694, pp. 232–247. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48294-6_15
5. Steffen, B., Knoop, J., Rüthing, O.: The value flow graph: a program representation for optimal program transformations. In: Jones, N. (ed.) ESOP 1990. LNCS, vol. 432, pp. 389–405. Springer, Heidelberg (1990). https://doi.org/10.1007/3-540-52592-0_76
6. Müller-Olm, M., Rüthing, O., Seidl, H.: Checking herbrand equalities and beyond. In: Cousot, R. (ed.) VMCAI 2005. LNCS, vol. 3385, pp. 79–96. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30579-8_6
7. Gulwani, S., Necula, G.C.: A polynomial-time algorithm for global value numbering. Sci. Comput. Program. **64**(1), 97–114 (2007)
8. Pai, R.R.: Detection of redundant expressions: a precise, efficient, and pragmatic algorithm in SSA. Comput. Lang. Syst. Struct. **46**, 167–181 (2016)
9. Kildall, G.A.: A unified approach to global program optimization. In: Proceedings of the 1st Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, POPL 1973, pp. 194–206. ACM (1973)
10. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, POPL 1977, pp. 238–252. ACM (1977)
11. Kam, J.B., Ullman, J.D.: Global data flow analysis and iterative algorithms. J. ACM **23**(1), 158–171 (1976)

12. Alpern, B., Wegman, M.N., Zadeck, F.K.: Detecting equality of variables in programs. In: Proceedings of the 15th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 1988, pp. 1–11. ACM (1988)
13. Rosen, B.K., Wegman, M.N., Zadeck, F.K.: Global value numbers and redundant computations. In: Proceedings of the 15th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 1988, pp. 12–27. ACM (1988)
14. Tarski, A.: A lattice-theoretical fixpoint theorem and its applications. Pacific J. Math. **5**(2), 285–309 (1955)
15. Cousot, P.: Asynchronous iterative methods for solving a fixed point system of monotone equations in a complete lattice. In: Research Report R.R. 88, Laboratoire IMAG, Université scientifique et médicale de Grenoble, Grenoble, France, 15 p., September 1977
16. Geser, A., Knoop, J., Lüttgen, G., Rüthing, O., Steffen, B.: Non-monotone fixpoint iterations to resolve second order effects. In: Gyimóthy, T. (ed.) CC 1996. LNCS, vol. 1060, pp. 106–118. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-61053-7_56
17. Babu, J., Krishnan, K.M., Paleri, V.: A fix-point characterization of herbrand equivalence of expressions in data flow frameworks. arXiv:1708.04976 (2017)