A fix-point characterization of Herbrand equivalence of expressions in data flow frameworks

K. Murali Krishnan

Department of Computer Science & Engineering
National Institute of Technology Calicut, Kerala, India.

Joint work with: Jasine Babu and Vineeth Paleri

ICLA 2019, IIT Delhi
March 5, 2019

Flow Graph Representation

A Simple Program

- P0:
- P1: $x = 1$
- P2: $y = 2$
- P3: $z = x + y$

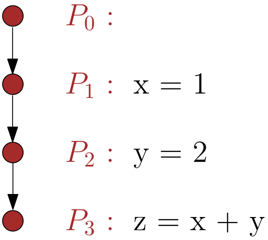
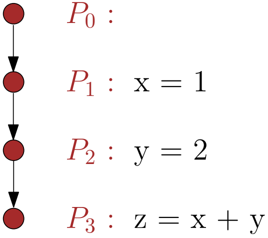


Figure: Flow Graph Representation

We may infer that the expressions $z, 1+2, 1+y, x+y$ and $x+2$ are **equivalent** expressions at $P3$.

Formalizing Equivalence



P_0 :

P_1 : x = 1

P_2 : y = 2

P_3 : z = x + y

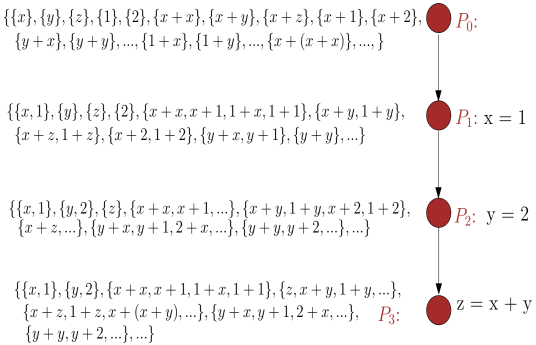
Figure: Flow graph

- **Constants** $C = \{1, 2, \ldots\}$
- **Variables** $V = \{x, y, z, \ldots\}$
- **Operators** $\{+, *, \ldots\}$
- **Terms** $T ::= V|C|T + T|T * T \ldots$
  $T = \{1, 2, x, y, z, x + y, x + z, \ldots\}$
- Only constants and variables *appearing in the program* are included in C and V.
- Thus T is an infinite set, but V and C are finite sets.

What should be the criteria for considering two terms $t_1, t_2 \in T$ to be **equivalent** at a program point P_i?

Properties of Equivalence

- **Congruence Property:** At any program point P_i, $s_1 \equiv t_1$ and $s_2 \equiv t_2$ if and only if $s_1 + s_2 \equiv t_1 + t_2$, $s_1 * s_2 \equiv t_1 * t_2$, . . .
- **Constants:** At any P_i, a constant can be equivalent to *only variables* (and **not** to other constants or non-variable terms).
- **Conservative Assumption:** At P_0 all terms are **inequivalent** to each other.



$\{\{x\}, \{y\}, \{z\}, \{1\}, \{2\}, \{x+x\}, \{x+y\}, \{x+z\}, \{x+1\}, \{x+2\}, \{y+x\}, \{y+y\}, ..., \{1+x\}, \{1+y\}, ..., \{x+(x+x)\}, ..., \}$ P_0:

$\{\{x, 1\}, \{y\}, \{z\}, \{2\}, \{x+x, x+1, 1+x, 1+1\}, \{x+y, 1+y\}, \{x+z, 1+z\}, \{x+2, 1+2\}, \{y+x, y+1\}, \{y+y\}, ...\}$ P_1: x = 1

$\{\{x, 1\}, \{y, 2\}, \{z\}, \{x+x, x+1, ...\}, \{x+y, 1+y, x+2, 1+2\}, \{x+z, ...\}, \{y+x, y+1, 2+x, ...\}, \{y+y, y+2, ...\}, ...\}$ P_2: y = 2

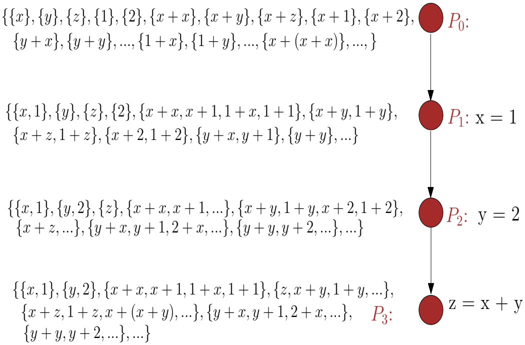
$\{\{x, 1\}, \{y, 2\}, \{x+x, x+1, 1+x, 1+1\}, \{z, x+y, 1+y, ...\}, \{x+z, 1+z, x+(x+y), ...\}, \{y+x, y+1, 2+x, ...\}, \{y+y, y+2, ...\}, ...\}$ P_3: z = x + y

Figure: Congruences

A **partition** of the set of terms T satisfying the **first two properties** above is *defined* as a **congruence**.

Assignments and Transfer functions

- An assignment operation can be modeled by a transfer function that transforms a congruence to another.
- The transfer function $f_{x=1}$ at P_1 transforms the congruence at P_0 to a new congruence at P_1.
- x moves from its present class to the class containing 1.
- Each term $t(x)$ containing x moves to the class containing $t[x \leftarrow 1]$.
- **Simplifying assumption:** In an assignment $x = t$, the term t shall not contain the variable x.



$\{\{x\},\{y\},\{z\},\{1\},\{2\},\{x+x\},\{x+y\},\{x+z\},\{x+1\},\{x+2\},$
$\{y+x\},\{y+y\},...,\{1+x\},\{1+y\},...,\{x+(x+x)\},...,\}$ P_0:

$\{\{x,1\},\{y\},\{z\},\{2\},\{x+x,x+1,1+x,1+1\},\{x+y,1+y\},$
$\{x+z,1+z\},\{x+2,1+2\},\{y+x,y+1\},\{y+y\},...\}$ P_1: x = 1

$\{\{x,1\},\{y,2\},\{z\},\{x+x,x+1,...\},\{x+y,1+y,x+2,1+2\},$
$\{x+z,...\},\{y+x,y+1,2+x,...\},\{y+y,y+2,...\},...\}$ P_2: y = 2

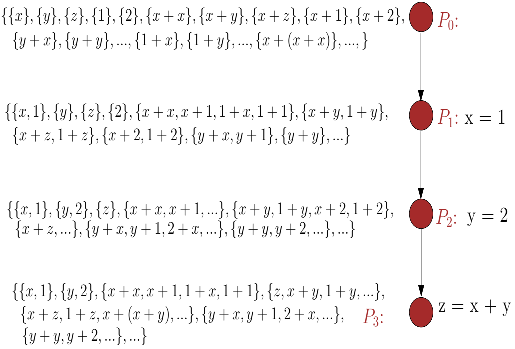
$\{\{x,1\},\{y,2\},\{x+x,x+1,1+x,1+1\},\{z,x+y,1+y,...\},$
$\{x+z,1+z,x+(x+y),...\},\{y+x,y+1,2+x,...\},$ P_3:
$\{y+y,y+2,...\},...\}$ z = x + y

Figure: Transfer functions acting on congruences

Def: $\mathbf{G}(T)$ is *defined* as **the set of all congruences** over T.

Path Analysis

- At P_0, all terms are considered inequivalent. We associate the congruence $\perp = \{\{t\} : t \in T\}$ to P_0.
- Each assignment node along a path $P_0, P_1, \ldots P_k$ transforms its input congruence to a new one.
- The congruence at P_k is obtained by applying to the initial congruence $\perp$, the composition of transfer functions of the assignment statements along the path.
- We call this congruence the path congruence associated with the path $P_0, P_1, \ldots P_k$.



$\{\{x\},\{y\},\{z\},\{1\},\{2\},\{x+x\},\{x+y\},\{x+z\},\{x+1\},\{x+2\},$
$\{y+x\},\{y+y\},...,\{1+x\},\{1+y\},...,\{x+(x+x)\},...,\}$ P_0:

$\{\{x,1\},\{y\},\{z\},\{2\},\{x+x,x+1,1+x,1+1\},\{x+y,1+y\},$
$\{x+z,1+z\},\{x+2,1+2\},\{y+x,y+1\},\{y+y\},...\}$ P_1: x = 1

$\{\{x,1\},\{y,2\},\{z\},\{x+x,x+1,...\},\{x+y,1+y,x+2,1+2\},$
$\{x+z,...\},\{y+x,y+1,2+x,...\},\{y+y,y+2,...\},...\}$ P_2: y = 2

$\{\{x,1\},\{y,2\},\{x+x,x+1,1+x,1+1\},\{z,x+y,1+y,...\},$
$\{x+z,1+z,x+(x+y),...\},\{y+x,y+1,2+x,...\},$
$\{y+y,y+2,...\},...\}$ P_3: z = x + y

Figure: Congruence at a program point

Confluence Points

- P0:
- P1: $z = 1$
- P2: $read(x)$
- P3: $If\ (x < 1)\ then\ y = x + 1$
- P4: $else\ y = x + 2$
- P5:

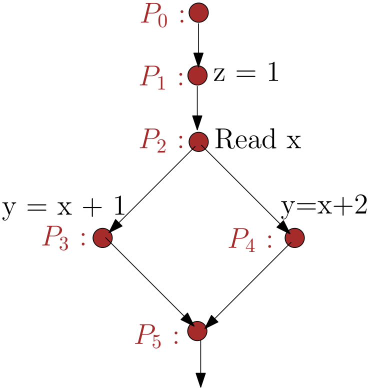
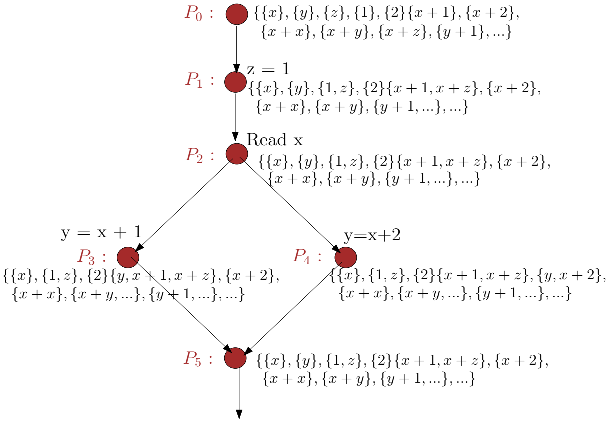


Figure: P_5 is a Confluence Point

Def: A point in the flow graph where **two branches meet** is called a **confluence point**. We assume that at most two paths merge at a confluence point.

Multi-path Analysis

- P0:
- P1: $z = 1$
- P2: $read(x)$
- P3: $If\ (x < z)\ then\ y = 1$
- P4: $else\ y = 2$
- P5:



$P_0 :$ $\{\{x\}, \{y\}, \{z\}, \{1\}, \{2\}\{x+1\}, \{x+2\}, \{x+x\}, \{x+y\}, \{x+z\}, \{y+1\}, ...\}$

z = 1

$P_1 :$ $\{\{x\}, \{y\}, \{1, z\}, \{2\}\{x+1, x+z\}, \{x+2\}, \{x+x\}, \{x+y\}, \{y+1, ...\}, ...\}$

Read x

$P_2 :$ $\{\{x\}, \{y\}, \{1, z\}, \{2\}\{x+1, x+z\}, \{x+2\}, \{x+x\}, \{x+y\}, \{y+1, ...\}, ...\}$

y = x + 1

$P_3 :$ $\{\{x\}, \{1, z\}, \{2\}\{y, x+1, x+z\}, \{x+2\}, \{x+x\}, \{x+y, ...\}, \{y+1, ...\}, ...\}$

y=x+2

$P_4 :$ $\{\{x\}, \{1, z\}, \{2\}\{x+1, x+z\}, \{y, x+2\}, \{x+x\}, \{x+y, ...\}, \{y+1, ...\}, ...\}$

$P_5 :$ $\{\{x\}, \{y\}, \{1, z\}, \{2\}\{x+1, x+z\}, \{x+2\}, \{x+x\}, \{x+y\}, \{y+1, ...\}, ...\}$

Figure: Congruence at confluence points

Conservative analysis assumes that terms t_1 and t_2 are equivalent at P_i if and only if $t_1 \equiv t_2$ in the path congruence of every P_0-P_i path.

Algebra of Congruences

Theorem

$\mathbf{G}(T)$ is a complete lattice.

- The Meet of congruences $\mathbf{C_1}$ and $\mathbf{C_2}$ is *defined* as: $\mathbf{C_1} \wedge \mathbf{C_2} = \{A_i \cap B_j : A_i \in \mathbf{C_1}, B_j \in \mathbf{C_2}, A_i \cap B_j \neq \emptyset\}$.
- $t_1 \equiv t_2$ in $\mathbf{C_1} \wedge \mathbf{C_2} \iff t_1 \equiv t_2$ in $\mathbf{C_1}$ and $t_1 \equiv t_2$ in $\mathbf{C_2}$.
- The definition extends to infinite collections $\{C_i\}_{i \in I}$ of congruences to yield the infimum, $\bigwedge_{i \in I} C_i$.
- The bottom element, $\bot = \{\{t\} : t \in T\}$ in $\mathbf{G}(T)$ satisfies $\bot \wedge C = \bot$ for each $C \in \mathbf{G}(T)$. Thus, $\mathbf{G}(T)$ is a complete meet semi-lattice.
- By artificially adding a top element $\top$, we can make $\mathbf{G}(T)$ a complete lattice.
- Ordering in $\mathbf{G}(T)$ is given by: $C_1 \preceq C_2$ if C_1 is a *finer partitioning* of T.
- $\mathbf{G}(T)$ is an infinite lattice.

Properties of transfer functions

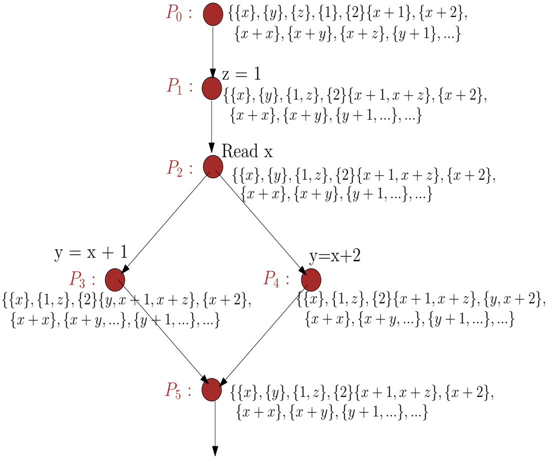
Theorem

The transfer function $f_{y=t}$ is a complete meet-morphism on $\mathbf{G}(T)$.

- $C_1, C_2 \in \mathbf{G}(T)$ be congruences.
- If $C_1 \preceq C_2$, then $f_{y=t}(C_1) \preceq f_{y=t}(C_2)$ (monotonicity).
- $f_{y=t}(C_1 \wedge C_2) = f_{y=t}(C_1) \wedge f_{y=t}(C_2)$. (meet-morphism).
- If $\{C_i\}_{i \in I}$ is an arbitrary non-empty collection of congruences then $f_{y=t}(\bigwedge_{i \in I} C_i) = \bigwedge_{i \in I} f_{y=t}(C_i)$. (complete meet-morphism).
- We define $f_{y=t}(\top) = \top$.

Meet Over all Paths (MOP)

- Formulate congruence at P_i as: $\mathcal{H}(i)$ = meet of all $P_0 - P_i$ path congruences.
- Note: The notion of path in the context of flow graphs includes paths with cycles.
- The equivalence classes at each P_i are traditionally called the **Herbrand Equivalence** classes.
- Hence we call $\mathcal{H}(i)$ the Herbrand congruence at P_i.
- When the program contain loops, we need to consider the meet of a countably infinite number of path congruences.



P_0 : $\{\{x\},\{y\},\{z\},\{1\},\{2\}\{x+1\},\{x+2\},$
$\{x+x\},\{x+y\},\{x+z\},\{y+1\},...\}$

z = 1

P_1 : $\{\{x\},\{y\},\{1,z\},\{2\}\{x+1,x+z\},\{x+2\},$
$\{x+x\},\{x+y\},\{y+1,...\},...\}$

Read x

P_2 : $\{\{x\},\{y\},\{1,z\},\{2\}\{x+1,x+z\},\{x+2\},$
$\{x+x\},\{x+y\},\{y+1,...\},...\}$

y = x + 1

P_3 : $\{\{x\},\{1,z\},\{2\}\{y,x+1,x+z\},\{x+2\},$
$\{x+x\},\{x+y,...\},\{y+1,...\},...\}$

y=x+2

P_4 : $\{\{x\},\{1,z\},\{2\}\{x+1,x+z\},\{y,x+2\},$
$\{x+x\},\{x+y,...\},\{y+1,...\},...\}$

P_5 : $\{\{x\},\{y\},\{1,z\},\{2\}\{x+1,x+z\},\{x+2\},$
$\{x+x\},\{x+y\},\{y+1,...\},...\}$

Figure: Meet over all paths computation.

Flow graphs with Loops

- P0:
- P1: *read*(x)
- P2:
- P3: *if* (condition) $y = x + 1$
- P4: *else* $y = x + 2$
- P5:
- P6: $x = y$
- ::::: if (condition) *goto* P2
- P7: $z = x + y$

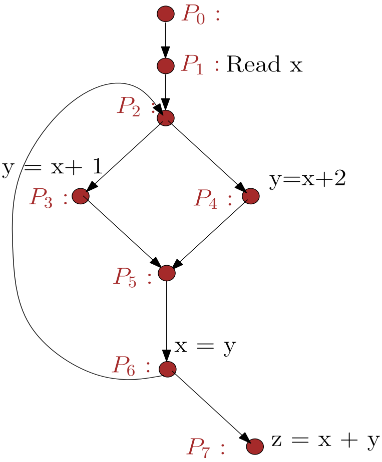
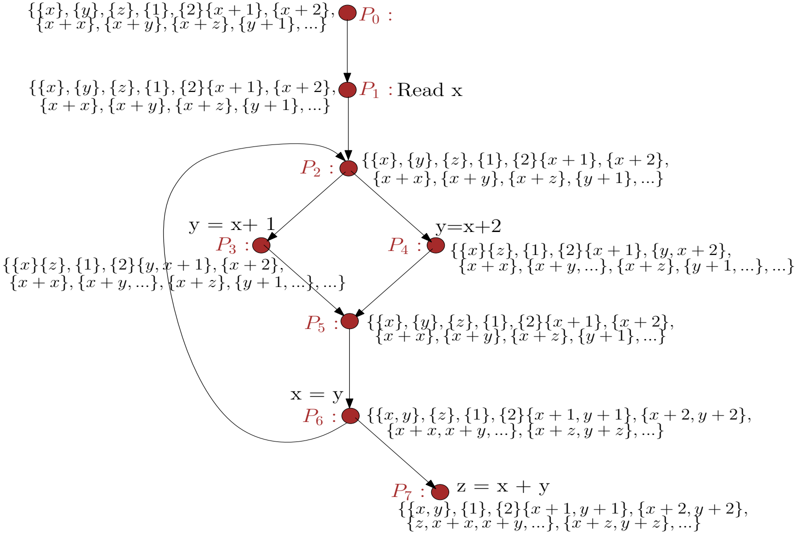


Figure: Flow graph with loops.

Congruences in flow graphs with loops



P_0 :
$\{\{x\},\{y\},\{z\},\{1\},\{2\}\{x+1\},\{x+2\},\{x+x\},\{x+y\},\{x+z\},\{y+1\},\ldots\}$

P_1 : Read x
$\{\{x\},\{y\},\{z\},\{1\},\{2\}\{x+1\},\{x+2\},\{x+x\},\{x+y\},\{x+z\},\{y+1\},\ldots\}$

P_2 :
$\{\{x\},\{y\},\{z\},\{1\},\{2\}\{x+1\},\{x+2\},\{x+x\},\{x+y\},\{x+z\},\{y+1\},\ldots\}$

y = x+ 1
P_3 :
$\{\{x\}\{z\},\{1\},\{2\}\{y,x+1\},\{x+2\},\{x+x\},\{x+y,\ldots\},\{x+z\},\{y+1,\ldots\},\ldots\}$

y=x+2
P_4 :
$\{\{x\}\{z\},\{1\},\{2\}\{x+1\},\{y,x+2\},\{x+x\},\{x+y,\ldots\},\{x+z\},\{y+1,\ldots\},\ldots\}$

P_5 :
$\{\{x\},\{y\},\{z\},\{1\},\{2\}\{x+1\},\{x+2\},\{x+x\},\{x+y\},\{x+z\},\{y+1\},\ldots\}$

x = y
P_6 :
$\{\{x,y\},\{z\},\{1\},\{2\}\{x+1,y+1\},\{x+2,y+2\},\{x+x,x+y,\ldots\},\{x+z,y+z\},\ldots\}$

P_7 : z = x + y
$\{\{x,y\},\{1\},\{2\}\{x+1,y+1\},\{x+2,y+2\},\{z,x+x,x+y,\ldots\},\{x+z,y+z\},\ldots\}$

Indefinite (Non deterministic) Assignments

- A statement of the form Read(y) is modeled by a special transfer function $f_{y=*}$.
- Let $C \in \mathbf{G}(T)$ and $C' = f_{y=*}(C)$. We require that
  $t_1 \equiv t_2$ in $C' \iff$ for each $t \in T$, $t_1 \equiv t_2$ in $f_{y=t}(C)$.
- Informally, t_1 and t_2 must be equivalent in the post assignment congruence if and only if irrespective of the term t assigned to y, they are equivalent.
- Hence, we define $f_{y=*}(C) = \bigwedge_{t \in T} f_{y=t}(C)$.
- It can be shown that if c_1, c_2 are constants such that $c_1 \neq c_2$ then
  $f_{y=*} = f_{y=c_1} \wedge f_{y=c_2}$.
- The characterization uses the defining property of a congruence that a constant can be equivalent to only variables, and **not** to other constants or non-variable terms.
- Thus, an input statement can be modeled with two transfer function operations and a confluence operation.

Problem Formulation

- Can we formulate the Herbrand equivalence classes at each P_i as the maximum fix-point (MFP) of some monotone function defined over $\mathbf{G}(T)$?
- Not quite - since the equivalence classes at P_i depends on the equivalence classes at other program points as well.
- To get the global picture, we need to consider the product lattice $[\mathbf{G}(T)]^n$, where $n =$ number of program points.
- $[\mathbf{G}(T)]^n$ is a complete lattice.
- We will define a composite transfer function $\mathcal{F} : [\mathbf{G}(T)]^n \mapsto [\mathbf{G}(T)]^n$.
- It turns out that $\mathcal{F}$ has a maximum fix-point.
- Moreover, $\mathrm{MFP}(\mathcal{F})[i] = \mathcal{H}(i)$, the Herbrand congruence at the program point P_i for each i.

The Composite Transfer Function

$\mathcal{F} : [\mathbf{G}(T)]^n \mapsto [\mathbf{G}(T)]^n$

- Let $\mathbf{C} = (C_1, C_1, \ldots C_n) \in [\mathbf{G}(T)]^n$.
- To define $\mathcal{F}$, it suffices to define $\mathcal{F}[i]$ for each program point P_i.
- Case 1: If P_i node associated with a transfer function $f_{y=t}$ and if P_j be the predecessor of P_i then: $\mathcal{F}(\mathbf{C})[i] = f_{y=t}(C_j)$.
- Case 2: If P_i is a confluence point with predecessors P_j and P_k then: $\mathcal{F}(\mathbf{C})[i] = C_j \wedge C_k$.
- Case 3: If $P_i = P_0$, the start node, define $\mathcal{F}(\mathbf{C})[i] = \perp$.
- $\mathcal{F}$ is a **monotone, distributive, complete meet-morphism**.
- Denote by $\mathrm{MFP}(\mathcal{F})$, the maximum fix-point of $\mathcal{F}$ in $[\mathbf{G}(T)]^n$ (must exist by Tarski's fix-point theorem).
- Lemma 1: $\mathrm{MFP}(\mathcal{F}) = \bigwedge\{\top, \mathcal{F}(\top), \mathcal{F}^2(\top), \ldots\} = \bigwedge_{k \geq 0} \mathcal{F}^k(\top)$.
  (The fact that $\mathcal{F}$ is a complete meet-morphism is used in the proof of the lemma).

Coincidence Theorem

- Let $\mathcal{H}^k(i)$ denote the meet of all path congruences $P_0 - P_i$ of length at most k.
- It follows that the **meet of all path congruences** at P_i, $\mathcal{H}(i) = \bigwedge_{k \geq 0} \mathcal{H}^k(i)$.
- Lemma 2: $\mathcal{H}^k(i) = \mathcal{F}^k(\top)[i]$, where $\top$ is the top element in $[\mathbf{G}(T)]^n$. (Straight-forward induction argument.)
- Thus $\mathcal{H}(i) = \bigwedge_{k \geq 0} \mathcal{H}^k(i) = (\bigwedge_{k \geq 0} \mathcal{F}^k(\top))[i]$ at each program point P_i.
- By Lemma 1 we have MFP$(\mathcal{F}) = \bigwedge_{k \geq 0} \mathcal{F}^k(\top))$.
- Hence we get:

Coincidence Theorem

Theorem: $\mathcal{H}(i) =$MFP$(\mathcal{F})[i]$, at each program point P_i.

Abstract Lattices

- Algorithms that computes Herbrand equivalence by fix-point iteration works with a limited finite working set $W \subset T$ of expressions and attempts to compute the equivalence classes of $\mathcal{H}(i)$ for each P_i, restricted to terms in W.
- The working set generally is a superset of the program expressions - i.e., expressions appearing in the program.
- We can define the abstract lattice $\mathbf{G}(W)$ of congruences of terms in W.
- Formally, we can define the abstraction of a congruence $\mathcal{C}$ in $\mathbf{G}(T)$ to $\mathbf{G}(W)$ by: $\Phi(\mathcal{C}) = \{A \cap W : A \in \mathcal{C}, A \cap W \neq \emptyset\}$.
- Then, $\mathbf{G}(W) = \{\Phi(\mathcal{C}) : \mathcal{C} \in \mathbf{G}(T), \Phi(\mathcal{C}) \neq \emptyset\}$.
- $\mathbf{G}(W)$ will be a finite complete lattice called the abstract lattice.
- The computational problem is to find $\Phi(\mathcal{H}(i))$ for each program point P_i in the flow graph.

Abstract Transfer Functions

- Suppose an abstract lattice $\mathbf{G}(W)$ over a finite working set W is given.
- For each transfer function $f_{y=t}$ in $\mathbf{G}(T)$, the corresponding abstract transfer function $\tilde{f}_{y=t}$ is defined on $\mathbf{G}(W)$ by: $\tilde{f}_{y=t}(\Phi(\mathcal{C})) = \Phi(f_{y=t}(\mathcal{C}))$.
- Similarly, the abstract composite transfer function $\tilde{\mathcal{F}}$ can be defined on $[\mathbf{G}(W)]^n$.
- It is not hard to see that $\mathrm{MFP}(\tilde{\mathcal{F}}) = \Phi(\mathrm{MFP}(\mathcal{F})) = \Phi(\mathcal{H}(i))$.

$$\phi\left(f_{y=t}\left(\mathcal{C}\right)\right) = \tilde{f}_{y=t}\left(\phi\left(\mathcal{C}\right)\right)$$

$$\begin{tikzcd}
G(T) & G(W) \\
\mathcal{C} = \{A_1, A_2, \ldots, A_r\} \arrow[r, "\phi"] \arrow[d, "f_{y=t}"] & \mathcal{C}^W = \{A_i^W : A_i^W \neq \emptyset, 1 \leq i \leq r\} \text{ where } A_i^W = A_i \cap W \arrow[d, "\tilde{f}_{y=t}"] \\
\mathcal{B} = \{B_1, B_2, \ldots, B_t\} \arrow[r, "\phi"] & \mathcal{B}^W = \{B_i^W : B_i^W \neq \emptyset, 1 \leq i \leq t\} \text{ where } B_i^W = B_i \cap W
\end{tikzcd}$$

Figure: Computation in an abstract lattice.

Computation of Herbrand Equivalence

- Several algorithms are known to compute Herbrand equivalence using fix-point iteration.
- Conceptually, each algorithm is distinguished by:
  1. The choice of the working set W.
  2. How the computation of the $\wedge$ operation in $\mathbf{G}(W)$ is performed.
  3. How the computation of the abstract transfer functions $\tilde{f}_{y=t}$ is performed.
- Each algorithm tries to compute $\Phi(\mathcal{H}(i))$ at each program point P_i either exactly or approximately.
- Approximate (or incomplete) methods compute a *conservative approximation* $\mathcal{G}(i) \in \mathbf{G}(W)$ at each program point P_i, such that $\mathcal{G}(i) \preceq \Phi(\mathcal{H}(i))$.

The journey so far

- Kildall (1973) proposed the first complete algorithm, but required exponential running time.
- Subsequently several polynomial time incomplete algorithms were discovered (Rüthing et. al. [1999], Alpern et. al. [1988], Rosen et. al.[1988], ...).
- Finally complete polynomial time algorithms were proposed (Gulvani and Necula. [2007], Pai. [2016]).
- A variant of the meet over all paths formulation for $\mathcal{H}(i)$ at each P_i appear in Steffen et. al. [1990].
- We hope that the fix-point formulation proposed here will provide a more natural view of the problem from the perspective of computation.

Thank you!