Monotone Data Flow Analysis Framework

**Definition.** A relation $R$ on a set $S$ is
(a) reflexive iff $(\forall x \in S)[xRx],$
(b) antisymmetric iff $[xRy \land yRx \rightarrow x=y]$
(c) transitive iff $(\forall x,y,z \in S) [xRy \land yRz \rightarrow xRz]$

**Definition.** A *partial ordering* on a set $S$, which we denote by the symbol $\leq$, is a reflexive, antisymmetric, and transitive relation on $S$. The pair $(S,\leq)$ is called a *partially ordered set*.

We write $x < Y$ iff $x \leq y$ and $x \neq y$.

**Definition.** Let $(S, \leq)$ be a partially ordered set, and let $a$ and $b$ be elements of $S$. A *join* (AKA least upperbound, lub) of $a$ and $b$ (denoted $a \lor b$) is an element $c \in S$ such that $[a \leq c$ and $b \leq c$ and there is no $x \in S$ such that $(a \leq x < c$ and $b \leq x < c$)].

A meet (AKA greatest lower bound, glb) of $a$ and $b$ (denoted $a \land b$) is an element $d \in S$ such that $[d \leq a$ and $d \leq b$ and there is no $x \in S$ such that $(d < x \leq a$ and $d < x \leq b$)].

**Observation.** Both meet and join are idempotent $x \land x = x, x \lor x = x$

**Definition** A lattice is a partially ordered set, any two elements of which have a unique join and meet.

**Definition** A semilattice is a pair $(S, *)$ where $S$ is a nonempty set and $*$ is a binary operation on $S$ which is idempotent, commutative, and associative. If $(S, \land, \lor)$ is a lattice, then both $(S, \land)$ and $(S, \lor)$ are semilattices.
A data flow analysis framework consists of:

1. A semilattice \((L, \wedge)\). Where \(L\) is a nonempty set, and \(\wedge\) is a binary operation on \(L\). \(\wedge\) represents the confluence operator.

2. A set \(F\) of transfer functions \(f \in F, f: L \to L\).

The elements of \(L\) are the values at the top of the nodes of a flow graph. For example, the in\([B]\) sets in the reaching definitions problem would be elements of \(L\). Thus, \(L = 2^D\) where \(D\) is the set of definitions in the program.

The transfer functions define the effect of a node of the flow graph on the elements of \(L\). In the case of reaching definitions, the set \(F\) is the set of functions of the form \(f(X) = A \cup (X - B)\) where \(A, B \in L\). \(A\) is gen and \(B\) is kill. (\(\wedge\) is set union).

For available expressions, \(L = 2^E\) where \(E\) is the set of expressions computed by the program. \(F\) has the same form as the function above, but \(A\) and \(B\) are now sets of expressions. (\(\wedge\) is now set intersection).
For constant propagation each element of L is a function (or table) 
\( \psi: V \rightarrow R \cup \{\text{nonconstant, undefined}\} \).
Where V is the set of variables in the program.

The transfer functions are created from the type of operation in the flowgraph:
(a) if no definitions then f is the identity.
(b) if \( x = c \) then \( f(\mu) = \nu \) with \( \nu(w) = \mu(w) \) \( \forall w \neq x \) and \( \nu(x) = c \).
(c) if \( x = y + z \) then \( f(\mu) = \nu \) with \( \nu(w) = \mu(w) \) \( \forall w \neq x \) \( \nu(x) = \mu(y) + \mu(z) \) [where nonconstant+a=nonconstant,undefined+a=undefined,
nonconstant+undefined=nonconstant] 
(d) if \( \text{read}(x) \) then \( f(\mu) = \nu \) with \( \nu(w) = \mu(w) \) \( \forall w \neq x \). \( \nu(x) = \text{nonconstant} \).

The meet operation of two tables m and n is defined with the following table

<table>
<thead>
<tr>
<th></th>
<th>nc</th>
<th>d</th>
<th>undef</th>
</tr>
</thead>
<tbody>
<tr>
<td>nc</td>
<td>nc</td>
<td>nc</td>
<td>nc</td>
</tr>
<tr>
<td>c</td>
<td>nc</td>
<td>if c=d then c else nc</td>
<td>c</td>
</tr>
<tr>
<td>undef</td>
<td>nc</td>
<td>d</td>
<td>undef</td>
</tr>
</tbody>
</table>
In semilattices: $\mu \leq \nu$ iff $\mu \land \nu = \mu$.

Also: $\mu \land \nu \leq \mu$ and $\mu \land \nu \leq \nu$

Thus, in reaching definitions, $x \leq y$ is equivalent to $y \subseteq x$.

In available expressions $x \leq y$ is equivalent to $x \subseteq y$. 

Basic assumptions

**Definition.** Given a semilattice \((L, \land)\) of finite length with a top element, a set of operations \(F\) on \(L\) is said to be a *monotone operation space associated with \(L\)* iff the following four conditions are satisfied.

1. Each \(f \in F\) is monotonic. That is
   \[(\forall f \in F)(\forall x, y \in L) [x \leq y \rightarrow f(x) \leq f(y)].\]
   This is equivalent to:
   \[(\forall f \in F)(\forall x, y \in L)[f(x \land y) \leq f(x) \land f(y)]\]

2. There is \(I \in F\) such that \(\forall x \in L, I(x) = x\)

3. \(\forall f, g \in F\) there is \(fg \in F\) such that \(\forall x \in L\) \(fg(x) = f(g(x))\)

4. For each \(x \in L\) there exists \(f \in F\) such that \(x = f(T)\)

**Definition** Given a semilattice \((L, \land)\) with a top element \(T\) in \(L\) (i.e., an element \(T\) such that \(T \land m = m, \forall m \in L\)) , a *monotone data flow analysis framework* is a triple \((L, \land, F)\) where \(F\) is a monotone operation space associated with \(L\).

Recall that \(\land\) is associative, commutative, and idempotent.

**Definition.** Distributivity: \(f(\mu \land \nu) = f(\mu) \land f(\nu)\)
The Meet-Over-All-Paths Solution to Data-Flow Problems

Consider a monotone dataflow framework \((L,\land,F)\). Assume that the functions \(f_B \in F\) represent the effect of a basic block on the sets containing data for a particular dataflow problem.

Let \(f_p(x)=f_{B_{k-1}}(... f_{B_1}(f_{B_0}(x)) ... )\) for a path \(P = (B_0, B_1, ..., B_{k-1})\)

The solution to the dataflow problem is

\[
\text{mop}(B) = \bigwedge (\text{over all paths from } B_0 \text{ to } B) f_p(T), \text{ where } T \geq s \text{ for all elements of the semilattice } L.
\]

When the framework is monotone and distributive, the mop can be computed as follows

\[
\text{foreach node } B \text{ do} \\
\quad \text{out}[B] = f_B[T] \\
\text{od} \\
\text{while changes to any } \text{out} \text{ occur do} \\
\quad \text{foreach block } B \text{ in depth-first order do} \\
\quad \quad \text{in}[B] = \bigwedge (\text{over all predecessors } P \text{ of } B) \text{ out}[P] \\
\quad \quad \text{out}[B]=f_B(\text{in}[B]) \\
\quad \text{od} \\
\text{od}
\]

Available expressions, live analysis, reaching definitions have monotone distributive f’s.

Constant propagation is not distributive.

There is no algorithm to compute the mop for the constant propagation framework.
**Computing Live Variable Using Interval Analysis**

Let \( \text{in}[x] \) be the live definitions at the beginning of block \( x \). This block could be a basic block or represent an interval.

Live variable will be computed in two phases. First, we will present the second phase.

**Algorithm LVIA-2: Live variable using interval analysis - Pass 2.**

*Input*: The derived sequence of control flow graphs \( G_0, G_1, ..., G_m \).
Where \( G_0 \) is the original graph, and \( G_m \) is the trivial graph.

The set \( \text{use}[x] \) of variables with upwardly exposed uses in block \( x \). Block \( x \) could be a basic block (graph \( G_0 \)) or represent an interval (graphs \( G_1 \) thru \( G_m \)).

The set \( \text{notdef}[x,y] \) of variables not necessarily defined when block \( x \) is traversed towards \( y \).

In the single node case:
\[
\text{notdef}(x,y) = \text{notdef}(x,z) = U-\text{def}(x)
\]

For a supernode the definition is more involved.

*Output*: \( \text{in}[x] \) the set of live definitions for all blocks \( x \).
Method:

begin
  \( in[N] = use[N] \) /* \( N \) is the single node in \( G_m \) */
  for each \( G = G_{m-1}, \ldots, G_0 \) do
    for each interval \( I \) of \( G \) do
      \( in[head(I)] = in[I] \)
      for each \( J \) in \( SUCC(I) \) do
        \( in[head(J)] = in[J] \)
      od /* \( in[I] \) and \( in[J] \) are available from the previous iteration */
    for each \( x \) in \( I \) - \( head(I) \) in reverse interval order do
      \( in[x] = use[x] \cup \bigcup (notdef[x,y] \cap in[y]) \)
        \( y \in SUCC(x) \)
    od
  od
end
To compute use[x] and notdef[x,y] we apply the following algorithm:

**Algorithm LVIA-1:Live variable using interval analysis-Pass 1.**

*Input:* The derived sequence of control flow graphs \( G_0, G_1, ..., G_m \). Where \( G_0 \) is the original graph, and \( G_m \) is the trivial graph.

The set \( \text{use}[x] \) of variables with upwardly exposed uses in basic block \( x \) of graph \( G_0 \).

The set \( \text{notdef}[x] = \text{U-def}[x] \) of variables not necessarily defined in basic block \( x \) of graph \( G_0 \). Notice that, since \( x \) is a basic block, \( \text{notdef}[x,y] = \text{notdef}[x] \) for all \( y \) in \( \text{SUCC}(x) \).

*Intermediate:* For each \( x \) in interval \( I \), path[x], the set of variables \( V \) for which there is a clear path (not containing a store into \( V \)) from the entry of \( I \) to the entry of \( x \).

*Output:* \( \text{use}[x] \) for all blocks \( x \) in graphs \( G_1, ..., G_m \). \( \text{notdef}[x,y] \) for all pairs of connected blocks \( x \) and \( y \) (i.e. \( y \) is a successor of \( x \) in some graph \( G_i \)).
Method:

begin
  for each $G = G_0, \ldots, G_{m-1}$ do
    for each interval $I$ of $G$ do
      use[$I$] = use[head($I$)]
      path[head($I$)] = U /* U is the set of all variables */
      /*use[head($I$)] is available from the previous iteration*/
      for each $x$ in $I - \text{head}(I)$ in interval order do
        path$_x$ = \bigcup_{y \in \text{PRED}(x)} (path$_y$ \cap \text{notdef}_{y, x})
        use$_I$ = use$_I$ \cup (path$_x$ \cap use$_x$)
      od
    for each $J$ in $\text{SUCC}(I)$ do
      notdef$_{I, J}$ = \bigcup_{y \in \text{PRED}(\text{head}(J)) \cap I} (path$_y$ \cap \text{notdef}_{y, \text{head}(J)})
    od
  od
end
Reaching Definitions Using T1 & T2

T1 and T2 generate regions possibly nested within each other.

To compute the reaching definitions proceed as follows:

For each region and basic block B compute from the inside-out

\[ gen[R,B] \] and \[ kill[R,B] \]

which correspond to the definitions generated and killed, respectively, along paths within the region from the header to the end of block B.

**Start with:** \( gen[B,B] = gen[B] \) and \( kill[B,B] = kill[B] \)

**Every time that T2 is applied** to have region R1 consume region R2 we have:

- If B is in R1, then \( gen[R,B] = gen[R1,B] \) and \( kill[R,B] = kill[R1,B] \)

- If B is in R2 then

\[
\begin{align*}
gen[R,B] &= gen[R2,B] \cup \left( \bigcup_{C \in PRED(head(R2))} \left( \bigcap gen[R1,C] \setminus kill[R2,B] \right) \right) \\
nkill[R,B] &= nkill[R2,B] \cup \left( \bigcup_{C \in PRED(head(R2))} \left( \bigcap nkill[R1,C] \setminus gen[R2,B] \right) \right)
\end{align*}
\]
Every time that T1 is applied to transform region R1 with a self loop into region R we have:

\[ gen[R, B] = gen[R1, B] \cup \left( \bigcup_{C \in PRED(head(R1))} gen[R1, C] \right) - kill[R1, B] \]

\[ kill[R, B] = kill[R1, B] \]

Once the process terminates with the trivial graph, \( U \) and

\[ out[B] = gen[U, B] \]

\[ in[B] = \bigcup_{C \in PRED[B]} gen[U, C] \]
Computing Reaching Definitions with Interval Analysis

As in the case of live variables, we proceed in two passes. Again, the result will be in \( \text{in}[x] \) the set of all definitions reaching block \( x \).

**Algorithm RDIA-2: Reaching Definitions with interval analysis-Pass 2.**

*Input:* The derived sequence of control flow graphs \( G_0, G_1, \ldots, G_m \). Where \( G_0 \) is the original graph, and \( G_m \) is the trivial graph.

The set \( \text{notkill}[x,y] \) of definitions not necessarily killed when block \( x \) is traversed towards \( y \).

The set \( \text{gen}[x,y] \) of definitions generated when block \( x \) is traversed towards \( y \).

*Output:* \( \text{in}[x] \) the set of all definitions reaching block \( x \).
begin

\( \text{in}[N] = \Phi \)

for each \( G = G_{m-j}, \ldots, G_0 \) do

for each interval \( I \) of \( G \) do

\( \text{in}[\text{head}(I)] = \text{in}[I] \)

/* \( \text{in}[I] \) is available from the previous iteration */

for each \( x \) in \( I - \text{head}(I) \) in interval order do

\( \text{in}[x] = \bigcup_{y \in \text{PRED}(x)} \text{in}[y] \cap \text{notkill}[y, x] \cup \text{gen}[y, x] \)

od

od

end
To compute gen[x,y] and notkill[x,y] we apply the following algorithm:

**Algorithm RDIA-1: Reaching Definitions with interval analysis-Pass 1.**

*Input*: The derived sequence of control flow graphs $G_0, G_1, ..., G_m$. Where $G_0$ is the original graph, and $G_m$ is the trivial graph.

The set $notkill[x]=U-kill[x]$ of definitions not necessarily killed in basic block $x$ of graph $G_0$. Notice that, since $x$ is a basic block, $notkill[x,y]=notkill[x]$ for all $y$ in $SUCC(x)$.

The set $gen[x]$ of definitions generated in basic block $x$ of graph $G_0$. Notice that, since $x$ is a basic block, $gen[x,y]=gen[x]$ for all $y$ in $SUCC(x)$.

*Intermediate*: For each $x$ in interval $I$, $path[x]$, the set of variables $V$ for which there is a clear path (not containing a store into $V$) from the entry of $I$ to the entry of $x$.

For each block $x$, $rdtop[x]$, the set of definitions that reach the top of $x$ from nodes within the interval.

*Output*: $notkill[x,y]$ and $gen[x,y]$. 
Method:

begin
  for each $G = G_0, \ldots, G_{m-1}$ do
    for each interval $I$ of $G$ do
      $rdtop[\text{head}(I)] = \Phi$
      $path[\text{head}(I)] = U$ /* $U$ is the set of all variables */
      for each $x$ in $I - \text{head}(I)$ in interval order do
        $path[x] = \bigcup_{y \in \text{PRED}(x)} (path[y] \cap \text{notkill}[y, x])$
        $rdtop[x] = \bigcup_{y \in \text{PRED}(x)} rdtop[y] \cap \text{notkill}[y, x] \cup \text{gen}[y, x]$
      od
      for each $J$ in $\text{SUCC}(I)$ do
        $\text{notkill}[I, J] = \bigcup_{y \in \text{PRED}(\text{head}(J)) \cap I} (path[y] \cap \text{notkill}[y, \text{head}(J)])$
        $\text{gen}[I, J] = \bigcup_{y \in \text{PRED}(\text{head}(J)) \cap I} A(y, J) \cup B(I, J)$
      od
  od
end