Another example: constant prop

- Set $D = \mathcal{P}(\{ x \to C \mid x \in \text{Var}, C \in \text{Cond} \})$

\[
\begin{align*}
&\{ x \to 5 \} \\
&\{ x \to 5, y \to 10 \}
\end{align*}
\]

\[
\begin{align*}
X := N & \quad \text{in} \\
\text{out} & \\
\hline
X := Y \text{ op } Z & \quad \text{in} \\
\text{out} & \\
\sqrt{X := 5}
\end{align*}
\]

\[
\begin{align*}
F_X := N(\text{in}) &= \text{in} - \{ x \to x \} \\
& \cup \{ x \to N \}
\end{align*}
\]

\[
\begin{align*}
F_X := Y \text{ op } Z(\text{in}) &= \text{in} - \{ x \to * \} \\
& \cup \{ x \to N, \text{ op } N_2 \} \\
& \cup \{ x \rightarrow N_1 \in \text{in}, \gamma \rightarrow N_2 \in \text{in} \}
\end{align*}
\]
Another example: constant prop

- Set $D = 2 \{ x \to N \mid x \in \text{Vars} \land N \in \mathbb{Z} \}$

\[
\begin{align*}
\text{in} & \quad X := N \\
\text{out} & \\
\text{in} & \quad X := Y \text{ op } Z \\
\text{out} & \\
\end{align*}
\]

\[
F_{X := N}(\text{in}) = \text{in} \setminus \{ X \to * \} \cup \{ X \to N \}
\]

\[
F_{X := Y \text{ op } Z}(\text{in}) = \text{in} \setminus \{ X \to * \} \cup \\
\{ X \to N \mid (Y \to N_1) \in \text{in} \land \\
(Z \to N_2) \in \text{in} \land \\
N = N_1 \text{ op } N_2 \}
\]
Another example: constant prop

\[ F_X := \gamma(\text{in}) = \] \[
\begin{aligned}
&\{ m - \{ x \mapsto * \} \\
&\quad \cup \{ x \mapsto N \mid \forall v \in \text{map}(y) \\
&\quad \quad \quad \quad \quad \quad \quad \quad \quad v \mapsto N \in \text{in} \}\}
\end{aligned}
\]

\[ F_{*X} := \gamma(\text{in}) = \] \[
\begin{aligned}
&\{ m - \{ w \mapsto * \mid w \in \text{map}(x) \} \\
&\quad \cup \{ v \mapsto N \mid v \mapsto N \in \text{in} \} \}
\end{aligned}
\]
Another example: constant prop

\[
\begin{align*}
F_X := \ast_Y (\text{in}) &= \text{in} - \{ X \to \ast \} \\
&\quad \cup \{ X \to N \mid \forall Z \in \text{may-point-to}(Y) \} \\
&\quad \cup (Z \to N) \in \text{in} \\
\end{align*}
\]

\[
\begin{align*}
F_{\ast_X} := Y (\text{in}) &= \text{in} - \{ Z \to \ast \mid Z \in \text{may-point}(X) \} \\
&\quad \cup \{ Z \to N \mid Z \in \text{must-point-to}(X) \land \\
&\quad Y \to N \in \text{in} \} \\
&\quad \cup \{ Z \to N \mid (Y \to N) \in \text{in} \land \\
&\quad (Z \to N) \in \text{in} \}
\end{align*}
\]
Another example: constant prop

\[ \*X := \*Y + \*Z \]
\[ F_X := G(...) \]

\[ \*X := \*Y + \*Z \]
\[ F_{\*X} := \*Y + \*Z_{(in)} = \]

\[ X := G(...) \]
\[ F_X := G(...)_{(in)} = \emptyset \]
Another example: constant prop

\[
\begin{align*}
*X & := *Y + *Z \\
F_{*X := *Y + *Z}(in) & = F_{a := *Y;b := *Z;c := a + b; *X := c}(in)
\end{align*}
\]

\[
\begin{align*}
X & := G(...) \\
F_X := G(...)(in) & = \emptyset
\end{align*}
\]
Another example: constant prop

```
Another example: constant prop

(in \rightarrow out[0], in \rightarrow out[1])
```

```
merge

in[0] \rightarrow in[1] \wedge
```

```
s: if \( x \leq 5 \)
```

```
out[0] \rightarrow out[1]
```

```
in \rightarrow out[0]
```

```
in \rightarrow out[1]
```

```
\{ x \leftarrow x, \rightarrow 2 \}
```
Lattice

• \((D, \sqsubseteq, \bot, T, U, \sqcap) = \)
Lattice

- \((D, \subseteq, \bot, T, \lor, \land) = (2^A, \supseteq, A, \emptyset, \cap, \cup)\)

where \(A = \{x \rightarrow N \mid x \in \text{Vars} \land N \in \mathbb{Z}\}\)
Example

\begin{align*}
x & := 5 \\
v & := 2 \\
x & := x + 1 \\
w & := v + 1 \\
w & := 3 \\
y & := x \times 2 \\
z & := y + 5 \\
w & := w \times v
\end{align*}
Another Example

\[
\begin{align*}
x & := 5 \\
a & := x + 10 \\
x & := x + 1 \\
x & := x - 1 \\
b & := x + 10
\end{align*}
\]
Another Example starting at top

\begin{align*}
x &:= 5 \\
a &:= x + 10 \\
x &:= x + 1 \\
x &:= x - 1 \\
b &:= x + 10
\end{align*}
Back to lattice

• \((D, \sqsubseteq, \bot, T, \cup, \cap) = (2^A, \supseteq, A, \emptyset, \cap, \cup)\)
  where \(A = \{ x \rightarrow N \mid x \in \text{Vars} \land N \in \mathbb{Z} \}\)

• What’s the problem with this lattice?
Back to lattice

• \((D, \sqsubseteq, \bot, T, \sqcup, \sqcap) = (2^A, \supseteq, A, \emptyset, \cap, \cup)\)
  where \(A = \{ x \rightarrow N | x \in \text{Vars} \land N \in \mathbb{Z} \}\)

• What’s the problem with this lattice?

• Lattice is infinitely high, which means we can’t guarantee termination
Better lattice

- Suppose we only had one variable
Better lattice

- Suppose we only had one variable

- $D = \{ \bot, \top \} \cup \mathbb{Z}$

- $\forall i \in \mathbb{Z} \cdot \bot \leq i \land i \leq \top$

- height = 3
For all variables

- Two possibilities
- Option 1: Tuple of lattices
- Given lattices \((D_1, \subseteq_1, \bot_1, \top_1, \cup_1, \cap_1) \ldots (D_n, \subseteq_n, \bot_n, \top_n, \cup_n, \cap_n)\) create:

\[
\text{tuple lattice } D^n = D_1 \times D_2 \ldots \times D_n \ni (d_1, d_2, d_3 \ldots d_n)
\]
For all variables

- Two possibilities
- Option 1: Tuple of lattices
- Given lattices \((D_1, \sqsubseteq_1, \perp_1, T_1, \sqcup_1, \sqcap_1)\) ... \((D_n, \sqsubseteq_n, \perp_n, T_n, \sqcup_n, \sqcap_n)\) create:

\[
\text{tuple lattice } D^n = ((D_1 \times ... \times D_n), \sqsubseteq, \perp, T, \sqcup, \sqcap) \text{ where }
\]

\[
\perp = (\perp_1, ..., \perp_n)
\]

\[
T = (T_1, ..., T_n)
\]

\[
(a_1, ..., a_n) \sqcup (b_1, ..., b_n) = (a_1 \sqcup_1 b_1, ..., a_n \sqcup_n b_n)
\]

\[
(a_1, ..., a_n) \sqcap (b_1, ..., b_n) = (a_1 \sqcap_1 b_1, ..., a_n \sqcap_n b_n)
\]

height = height\((D_1) + ... + \text{height}\((D_n)\)
For all variables

- Option 2: Map from variables to single lattice
- Given lattice $(D, \sqsubseteq_1, \bot_1, \top_1, \cup_1, \cap_1)$ and a set $V$, create:

  $$\text{map lattice } V \to D = (V \to D, \sqsubseteq, \bot, \top, \cup, \cap)$$
Back to example

\[ m(x := y \text{ op } z) \]

\[ F_X := y \text{ op } z \text{(in)} = \]

\[ m[x \mapsto m(\neg y) \text{ op } m(2)] \]

\[ \text{op} \]

\[ \begin{array}{ccc}
\neg & c_1 & T \\
T & T & T \\
c_2 & T & c_1 \text{ op } c_2 & T \\
T & T & T & T \\
\end{array} \]
Back to example

\[
\begin{align*}
X := Y \text{ op } Z \\
in (X) = in(Y) \text{ op } in(Z)
\end{align*}
\]

where \( a \text{ op } b = \)

\[
\begin{array}{ccc}
\hat{\text{op}} & \bar{1} & d_1 & T \\
T & \bar{1} & T & T \\
d_2 & T & d_1 \text{ op } d_2 & T \\
T & T & T & T
\end{array}
\]
General approach to domain design

• Simple lattices:
  – boolean logic lattice
  – powerset lattice
  – incomparable set: set of incomparable values, plus top and bottom (eg const prop lattice)
  – two point lattice: just top and bottom

• Use combinators to create more complicated lattices
  – tuple lattice constructor
  – map lattice constructor
May vs Must

- Has to do with definition of computed info

- Set of $x \rightarrow y$ must-point-to pairs
  - if we compute $x \rightarrow y$, then, then during program execution, $x$ must point to $y$

- Set of $x \rightarrow y$ may-point-to pairs
  - if during program execution, it is possible for $x$ to point to $y$, then we must compute $x \rightarrow y$
## May vs must

<table>
<thead>
<tr>
<th></th>
<th>May</th>
<th>Must</th>
</tr>
</thead>
<tbody>
<tr>
<td>most optimistic (bottom)</td>
<td>Ø</td>
<td>FS</td>
</tr>
<tr>
<td>most conservative (top)</td>
<td>FS</td>
<td>Ø</td>
</tr>
<tr>
<td>safe</td>
<td>Add</td>
<td>Remove</td>
</tr>
<tr>
<td>merge</td>
<td>U</td>
<td>∨</td>
</tr>
<tr>
<td></td>
<td>May</td>
<td>Must</td>
</tr>
<tr>
<td>--------------------------</td>
<td>--------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>most optimistic (bottom)</td>
<td>empty set</td>
<td>full set</td>
</tr>
<tr>
<td>most conservative (top)</td>
<td>full set</td>
<td>empty set</td>
</tr>
<tr>
<td>safe</td>
<td>overly big</td>
<td>overly small</td>
</tr>
<tr>
<td>merge</td>
<td>$\cup$</td>
<td>$\cap$</td>
</tr>
</tbody>
</table>
Common Sub-expression Elim

- Want to compute when an expression is available in a var
- Domain:
Common Sub-expression Elim

• Want to compute when an expression is available in a var

• Domain:

\[ S = \{ X \rightarrow E \mid X \in \text{Var}, E \in \text{Exp} \} \]

\[ \emptyset = 2^S \]

\[ \mathbf{F} = S \]

\[ \mathbf{T} = \emptyset \]

\[ u = \land \]
Flow functions

\[ X := Y \text{ op } Z \]

\[ F_X := Y \text{ op } Z \text{ (in)} = \]

\[ a := a + b \]
\[ a \to a + b \]

\[ F_X := Y \text{ op } Z \text{ (in)} = \]

\[ \text{in} - \{ x \to \ast, \ast \to \cdots \ast \} \]
\[ \cup \{ x \to Y \text{ op } Z \} \]
\[ \land x \neq y \land x \neq z \]

\[ F_X := Y \text{ (in)} = \]

\[ \text{in} - \{ x \to \ast, \ast \to \cdots \ast \} \]
\[ \cup \{ x \to E \mid y \to E \in \text{in} \} \]
Flow functions

\[ F_{X := Y \text{ op } Z}(\text{in}) = \text{in} - \{ X \rightarrow * \} \]
\[ - \{ * \rightarrow \ldots X \ldots \} \cup \]
\[ \{ X \rightarrow Y \text{ op } Z \mid X \neq Y \land X \neq Z \} \]

\[ F_{X := Y}(\text{in}) = \text{in} - \{ X \rightarrow * \} \]
\[ - \{ * \rightarrow \ldots X \ldots \} \cup \]
\[ \{ X \rightarrow E \mid Y \rightarrow E \in \text{in} \} \]
Example

\[ x := \text{read()} \]
\[ v := a + b \]
\[ x := x + 1 \]
\[ w := x + 1 \]
\[ w := x + 1 \]
\[ a := w \]
\[ v := a + b \]
\[ z := x + 1 \]
\[ t := a + b \]
Direction of analysis

• Although constraints are not directional, flow functions are.

• All flow functions we have seen so far are in the forward direction.

• In some cases, the constraints are of the form $\text{in} = F(\text{out})$.

• These are called backward problems.

• Example: live variables
  – compute the set of variables that may be live.
Live Variables

• A variable is live at a program point if it will be used before being redefined.

• A variable is dead at a program point if it is redefined before being used.
Example: live variables

• Set $D = \mathcal{P}(\nu)$

• Lattice: $(D, \subseteq, \bot, T, \cup, \cap) =$

  $\leq \emptyset \lor \lor \land$
Example: live variables

- Set $D = 2^{\text{Vars}}$
- Lattice: $(D, \subseteq, \bot, \top, \cup, \cap) = (2^{\text{Vars}}, \subseteq, \emptyset, \text{Vars}, \cup, \cap)$

\[
\text{in}
\]

\[
\text{out}
\]

\[
X := Y \text{ op } Z
\]

\[
F_{X := Y \text{ op } Z}(\text{out}) = \text{out}
\]

\[
\{ h \times \} \cup \{ y, z \}
\]

\[
a := a + a
\]
Example: live variables

- Set $D = 2^\text{Vars}$
- Lattice: $(D, \sqsubseteq, \bot, T, \cup, \cap) = (2^\text{Vars}, \sqsubseteq, \emptyset, \text{Vars}, \cup, \cap)$

\[
\begin{array}{c}
\text{in} \\
\text{X := Y op Z} \\
\text{out}
\end{array}
\]

\[
F_X := Y \text{ op } Z(\text{out}) = \text{out} - \{ X \} \cup \{ Y, Z \}
\]
Example: live variables

\begin{align*}
x &:= 5 \\
y &:= x + 2
\end{align*}

\[
x := x + 1
\]

\[
y := x + 10
\]

... y ...
Example: live variables

How can we remove the \( x := x + 1 \) stmt?
Revisiting assignment

\[ X := Y \text{ op } Z \]

in

out

\[ F_X := Y \text{ op } Z(\text{out}) = \text{out} - \{ X \} \cup \{ Y, Z \} \]

\[ x = 15 \]

\[ \rightarrow x \text{ is dead} \]
Revisiting assignment

\[
F_X := Y \circ Z(\text{out}) = \text{out} - \{ X \} \cup \{ Y, Z \}
\]

\[
\text{out} - \{ x \} \cup \{ y \} \in \text{out}
\]

\[
x \notin \text{out} \iff \emptyset : \{ y, z \}
\]
Theory of backward analyses

• Can formalize backward analyses in two ways
  
  • Option 1: reverse flow graph, and then run forward problem
  
  • Option 2: re-develop the theory, but in the backward direction
Precision

• Going back to constant prop, in what cases would we lose precision?

```plaintext
if (p)
  x = 5
else
  x = 7
if (p)
  x = x + 7
else
  x = x + 5
```
Going back to constant prop, in what cases would we lose precision?

\[
x := 5 \\
\text{if } (<\text{expr}>) \{ \\
    x := 6 \\
\} \text{ else } \\
\]

\[
... \ x \ ...
\]

where \(<\text{expr}>\) is equiv to false

\[
\text{if } (p) \{ \\
x := 5; \\
\} \text{ else } \\
x := 4; \\
\}
\]

\[
... \\
\]

\[
\text{if } (...) \{ \\
x := -1; \\
\} \text{ else } \\
x := 1; \\
\}
\]

\[
y := x * x; \\
... \ y \ ...
\]
Precision

• The first problem: Unreachable code
  – solution: run unreachable code removal before
  – the unreachable code removal analysis will do its best, but may not remove all unreachable code

• The other two problems are path-sensitivity issues
  – Branch correlations: some paths are infeasible
  – Path merging: can lead to loss of precision
MOP: meet over all paths

- Information computed at a given point is the meet of the information computed by each path to the program point

```plaintext
if (...) {
    x := -1;
} else {
    x := 1;
}
y := x * x;
... y ...
```
For a path $p$, which is a sequence of statements $[s_1, ..., s_n]$ , define: $F_p(in) = F_{s_n}( ... F_{s_1}(in) ... )$

In other words: $F_p = \overline{F_{s_1} \circ ... \circ F_{s_n}}$

Given an edge $e$, let paths-to($e$) be the (possibly infinite) set of paths that lead to $e$

Given an edge $e$, $MOP(e) = \bigvee_{p \in \text{paths-to}(e)} F_p(\bot)$

For us, should be called JOP (ie: join, not meet)
MOP vs. dataflow

- MOP is the “best” possible answer, given a fixed set of flow functions
  - This means that MOP $\subseteq$ dataflow at edge in the CFG

- In general, MOP is not computable (because there can be infinitely many paths)
  - vs dataflow which is generally computable (if flow fns are monotonic and height of lattice is finite)

- And we saw in our example, in general, MOP $\neq$ dataflow
MOP vs. dataflow

\[ F(\alpha \cup \beta) = F(\alpha) \cup F(\beta) \]

- However, it would be great if by imposing some restrictions on the flow functions, we could guarantee that dataflow is the same as MOP. What would this restriction be?
MOP vs. dataflow

• However, it would be great if by imposing some restrictions on the flow functions, we could guarantee that dataflow is the same as MOP. What would this restriction be?

• Distributive problems. A problem is distributive if:

\[ \forall a, b . F(a \sqcup b) = F(a) \sqcup F(b) \]

• If flow function is distributive, then MOP = dataflow
Summary of precision

• Dataflow is the basic algorithm

• To basic dataflow, we can add path-separation
  – Get MOP, which is same as dataflow for distributive problems
  – Variety of research efforts to get closer to MOP for non-distributive problems

• To basic dataflow, we can add path-pruning
  – Get branch correlation

• To basic dataflow, can add both:
  – meet over all feasible paths