Another example: constant prop

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

\[
\begin{align*}
F_{X \Rightarrow N}(in) &= \{ X \Rightarrow * \} \cup \{ X \Rightarrow N \} \\
F_{X \Rightarrow Y \op Z}(in) &= \{ X \Rightarrow * \} \cup \{ X \Rightarrow N \mid (Y \Rightarrow N_1) \in \text{in} \land (Z \Rightarrow N_2) \in \text{in} \land N = N_1 \op N_2 \}
\end{align*}
\]

Another example: constant prop

\[
\begin{align*}
F_{X \Rightarrow \gamma}(in) &= in - \{ X \Rightarrow * \} \\
F_{\ast X \Rightarrow \gamma}(in) &= in - \{ Z \Rightarrow * \} \cup \{ X \Rightarrow N \mid \forall Z \in \text{may-point-to}(Y) \land (Z \Rightarrow N) \in \text{in} \}
\end{align*}
\]

Another example: constant prop

\[
\begin{align*}
F_{\ast X \Rightarrow \gamma}(in) &= in - \{ X \Rightarrow * \} \\
F_{\ast X \Rightarrow \gamma}(in) &= in - \{ X \Rightarrow * \} \cup \{ X \Rightarrow N \mid \forall Z \in \text{may-point-to}(X) \land (Z \Rightarrow N) \in \text{in} \land Y \Rightarrow N \in \text{in} \land \{ Z \Rightarrow N \mid (Y \Rightarrow N) \in \text{in} \land (Z \Rightarrow N) \in \text{in} \}
\end{align*}
\]

Another example: constant prop

\[
\begin{align*}
F_{\ast X \Rightarrow \gamma}(in) &= in - \{ X \Rightarrow * \} \\
F_{\ast X \Rightarrow \gamma}(in) &= in - \{ X \Rightarrow * \} \cup \{ X \Rightarrow N \mid \forall Z \in \text{may-point-to}(X) \land (Z \Rightarrow N) \in \text{in} \land Y \Rightarrow N \in \text{in} \land \{ Z \Rightarrow N \mid (Y \Rightarrow N) \in \text{in} \land (Z \Rightarrow N) \in \text{in} \}
\end{align*}
\]

Another example: constant prop

\[
\begin{align*}
F_{\ast X \Rightarrow \gamma}(in) &= in - \{ X \Rightarrow * \} \\
F_{\ast X \Rightarrow \gamma}(in) &= in - \{ X \Rightarrow * \} \cup \{ X \Rightarrow N \mid \forall Z \in \text{may-point-to}(X) \land (Z \Rightarrow N) \in \text{in} \land Y \Rightarrow N \in \text{in} \land \{ Z \Rightarrow N \mid (Y \Rightarrow N) \in \text{in} \land (Z \Rightarrow N) \in \text{in} \}
\end{align*}
\]

Another example: constant prop

\[
\begin{align*}
F_{\ast X \Rightarrow \gamma}(in) &= in - \{ X \Rightarrow * \} \\
F_{\ast X \Rightarrow \gamma}(in) &= in - \{ X \Rightarrow * \} \cup \{ X \Rightarrow N \mid \forall Z \in \text{may-point-to}(X) \land (Z \Rightarrow N) \in \text{in} \land Y \Rightarrow N \in \text{in} \land \{ Z \Rightarrow N \mid (Y \Rightarrow N) \in \text{in} \land (Z \Rightarrow N) \in \text{in} \}
\end{align*}
\]

Another example: constant prop

\[
\begin{align*}
F_{\ast X \Rightarrow \gamma}(in) &= in - \{ X \Rightarrow * \} \\
F_{\ast X \Rightarrow \gamma}(in) &= in - \{ X \Rightarrow * \} \cup \{ X \Rightarrow N \mid \forall Z \in \text{may-point-to}(X) \land (Z \Rightarrow N) \in \text{in} \land Y \Rightarrow N \in \text{in} \land \{ Z \Rightarrow N \mid (Y \Rightarrow N) \in \text{in} \land (Z \Rightarrow N) \in \text{in} \}
\end{align*}
\]

Another example: constant prop

\[
\begin{align*}
F_{\ast X \Rightarrow \gamma}(in) &= in - \{ X \Rightarrow * \} \\
F_{\ast X \Rightarrow \gamma}(in) &= in - \{ X \Rightarrow * \} \cup \{ X \Rightarrow N \mid \forall Z \in \text{may-point-to}(X) \land (Z \Rightarrow N) \in \text{in} \land Y \Rightarrow N \in \text{in} \land \{ Z \Rightarrow N \mid (Y \Rightarrow N) \in \text{in} \land (Z \Rightarrow N) \in \text{in} \}
\end{align*}
\]

Another example: constant prop

\[
\begin{align*}
F_{\ast X \Rightarrow \gamma}(in) &= in - \{ X \Rightarrow * \} \\
F_{\ast X \Rightarrow \gamma}(in) &= in - \{ X \Rightarrow * \} \cup \{ X \Rightarrow N \mid \forall Z \in \text{may-point-to}(X) \land (Z \Rightarrow N) \in \text{in} \land Y \Rightarrow N \in \text{in} \land \{ Z \Rightarrow N \mid (Y \Rightarrow N) \in \text{in} \land (Z \Rightarrow N) \in \text{in} \}
\end{align*}
\]

Another example: constant prop

\[
\begin{align*}
F_{\ast X \Rightarrow \gamma}(in) &= in - \{ X \Rightarrow * \} \\
F_{\ast X \Rightarrow \gamma}(in) &= in - \{ X \Rightarrow * \} \cup \{ X \Rightarrow N \mid \forall Z \in \text{may-point-to}(X) \land (Z \Rightarrow N) \in \text{in} \land Y \Rightarrow N \in \text{in} \land \{ Z \Rightarrow N \mid (Y \Rightarrow N) \in \text{in} \land (Z \Rightarrow N) \in \text{in} \}
\end{align*}
\]

Another example: constant prop

\[
\begin{align*}
F_{\ast X \Rightarrow \gamma}(in) &= in - \{ X \Rightarrow * \} \\
F_{\ast X \Rightarrow \gamma}(in) &= in - \{ X \Rightarrow * \} \cup \{ X \Rightarrow N \mid \forall Z \in \text{may-point-to}(X) \land (Z \Rightarrow N) \in \text{in} \land Y \Rightarrow N \in \text{in} \land \{ Z \Rightarrow N \mid (Y \Rightarrow N) \in \text{in} \land (Z \Rightarrow N) \in \text{in} \}
\end{align*}
\]

Another example: constant prop

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

\[
\text{if (\ldots)}
\]

Lattice

\[ (D, \sqsubseteq, \bot, T, U, \Pi) = \]

Lattice

\[ (D, \sqsubseteq, \bot, T, U, \Pi) = \]

Example

Another Example

Another Example starting at top
Back to lattice

- \((D, \sqsubseteq, \sqcap, \sqcup, \sqcap, \sqcup) = (\mathbb{2}^\mathbb{A}, \sqsupseteq, \mathbb{A}, \emptyset, \sqcap, \sqcup)\)
  where \(\mathbb{A} = \{ x \to \mathbb{N} | x \in \text{Vars} \land N \in \mathbb{Z} \}\)

- What's the problem with this lattice?

Better lattice

- Suppose we only had one variable

For all variables

- Two possibilities
  - Option 1: Tuple of lattices
    - Given lattices \((D_1, \sqsubseteq, \sqcap, \sqcup, \sqcap_1, \sqcup_1) \ldots (D_n, \sqsubseteq, \sqcap, \sqcup, \sqcap_n, \sqcup_n)\)
      create:
      \[ \text{tuple lattice } D^n = \ldots \]

Back to lattice

- \((D, \sqsubseteq, \sqcap, \sqcup, \sqcap, \sqcup) = (\mathbb{2}^\mathbb{A}, \sqsupseteq, \mathbb{A}, \emptyset, \sqcap, \sqcup)\)
  where \(\mathbb{A} = \{ x \to \mathbb{N} | x \in \text{Vars} \land N \in \mathbb{Z} \}\)

- 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

For all variables

- Two possibilities
  - Option 1: Tuple of lattices
    - Given lattices \((D_1, \sqsubseteq, \sqcap, \sqcup, \sqcap_1, \sqcup_1) \ldots (D_m, \sqsubseteq, \sqcap, \sqcup, \sqcap_m, \sqcup_m)\)
      create:
      \[ \text{tuple lattice } D^n = \ldots \]

  - \(D = \{\bot, \top\} \cup \mathbb{Z}\)
  - \(\forall i \in \mathbb{Z} : \bot \sqsubseteq i \land i \sqsubseteq \top\)
  - height = 3

- For all variables
For all variables

- Option 2: Map from variables to single lattice
- Given lattice \((D, \sqsubseteq, \bot, \top, \sqcup, \sqcap)\) and a set \(V\), create:

  \[ \text{map lattice } V \rightarrow D = (V \rightarrow D, \sqsubseteq, \bot, \top, \sqcup, \sqcap) \]

  \[ \bot = \lambda V \mapsto \bot, \]

General approach to domain design

- Simple lattices:
  - boolean logic lattice
  - powerset lattice
  - incomparable set: set of incomparable values, plus top and bottom (e.g., 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>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:

\[ S \subseteq \{ X = \xi \mid X \in \mathcal{U}, \xi \in \mathcal{E}_{\text{perm}} \} \]

Flow functions

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

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

Example

- $x := \text{read()}$
- $v := a + b$
- $x := x + 1$
- $w := x + 1$
- $a := w$
- $v := a + b$
- $t := x + 1$
**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 \( D = 2 \text{Vars} \)
- Lattice: \( (D, \subseteq, \bot, T, U, \cap) = \)

**Example: live variables**

- Set \( D = 2 \text{Vars} \)
- Lattice: \( (D, \subseteq, \bot, T, U, \cap) = (2\text{Vars}, \subseteq, \emptyset, \text{Vars}, U, \cap) \)

**Example: live variables**

- Set \( D = 2 \text{Vars} \)
- Lattice: \( (D, \subseteq, \bot, T, U, \cap) = (2\text{Vars}, \subseteq, \emptyset, \text{Vars}, U, \cap) \)

**Example: live variables**

- Set \( D = 2 \text{Vars} \)
- Lattice: \( (D, \subseteq, \bot, T, U, \cap) = (2\text{Vars}, \subseteq, \emptyset, \text{Vars}, U, \cap) \)
Example: live variables

Revisiting assignment

How can we remove the $x := x + 1$ stmt?

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?

Precision

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

```
if (...) {
  x := -1;
} else {
  x := 1;
}
y := x * x;
... y ...
```

**MOP**

- For a path \( p \), which is a sequence of statements \([s_1, \ldots, s_n]\), define: \( F_p(in) = F_{s_n} \ldots F_{s_1}(in) \ldots \)
- In other words: \( F_p = F_{s_n} \circ \ldots \circ F_{s_1} \)
- 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) = \( \bigwedge_{p \in \text{paths-to}(e)} F_p(\bot) \)
- 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 \( \sqsubseteq \) 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**

- 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