Generic (forward) dataflow analysis algorithm

- Given:
  - Abstract domain \((L, \subseteq, \cup, \bot, \top)\)
  - Transfer function
    \(post_L : Basic\ Block \times L \to L\)
  - Control flow graph \(G = (N, E, s)\)
- Compute: least annotation \(IN, OUT\) such that
  1. \(IN[s] = \top\)
  2. For all \(n \in N\), \(post_L(n, IN[n]) \subseteq OUT[n]\)
  3. For all \(p \to n \in E\), \(OUT[p] \subseteq IN(n)\)

\[
\begin{align*}
IN[s] &= \top, \quad OUT[s] = \bot; \\
IN[n] &= OUT[n] = \bot\text{ for all other nodes } n; \\
work &= N; \\
\text{while } work \neq \emptyset \text{ do} \\
&\quad \text{Pick some } n \text{ from work;} \\
&\quad work \leftarrow work \setminus \{n\}; \\
&\quad old \leftarrow OUT[n]; \\
&\quad IN[n] \leftarrow IN[n] \cup \bigcup_{p \in pred(n)} OUT[p]; \\
&\quad OUT[n] \leftarrow post_L(n, IN[n]); \\
&\quad \text{if } old \neq OUT[n] \text{ then} \\
&\quad\quad work \leftarrow work \cup succ(n) \\
\end{align*}
\]

return \(IN, OUT\)
(Partial) Correctness

\[
\begin{align*}
\text{IN}[s] &= \top, \text{OUT}[s] = \bot; \\
\text{IN}[n] &= \text{OUT}[n] = \bot \text{ for all other nodes } n; \\
\text{work} &\leftarrow N; \\
\text{while } \text{work} \neq \emptyset \text{ do} \\
& \quad \text{Pick some } n \text{ from work; } \\
& \quad \text{work} \leftarrow \text{work} \setminus \{n\}; \\
& \quad \text{old} \leftarrow \text{OUT}[n]; \\
& \quad \text{IN}[n] \leftarrow \text{IN}[n] \cup \bigcup_{p \in \text{pred}(n)} \text{OUT}[p]; \\
& \quad \text{OUT}[n] \leftarrow \text{post}_L(n, \text{IN}[n]); \\
& \quad \text{if } \text{old} \neq \text{OUT}[n] \text{ then} \\
& \quad \quad \text{work} \leftarrow \text{work} \cup \text{succ}(n)
\end{align*}
\]

When algorithm terminates, all constraints are satisfied. Invariants:

- \text{IN}[s] = \top
- \text{For any } n \in N, \text{ if } \text{post}_L(n, \text{IN}[n]) \not\subseteq \text{OUT}[n], \text{ we have } n \in \text{work}
- \text{For any } p \to n \in E \text{ with } \text{OUT}[p] \not\subseteq \text{IN}(n), \text{ we have } n \in \text{work}
Optimality

Algorithm computes \textit{least} solution.

- Invariant: $\text{IN} \sqsubseteq^* \overline{\text{IN}}$ and $\text{OUT} \sqsubseteq^* \overline{\text{OUT}}$, where
  - $\overline{\text{IN/OUT}}$ denotes any solution to the constraint system
  - $\sqsubseteq^*$ is pointwise order on function space $N \to \mathcal{L}$
Optimality

Algorithm computes least solution.

- Invariant: \( \text{IN} \sqsubseteq^* \overline{\text{IN}} \) and \( \text{OUT} \sqsubseteq^* \overline{\text{OUT}} \), where
  - \( \overline{\text{IN}}/\overline{\text{OUT}} \) denotes any solution to the constraint system
  - \( \sqsubseteq^* \) is pointwise order on function space \( N \rightarrow \mathcal{L} \)

- Argument: let \( \text{IN}_i/\text{OUT}_i \) be \( \text{IN}/\text{OUT} \) at iteration \( i \); \( n_i \) be workset item
  - Base case \( \text{IN}_0 \sqsubseteq^* \overline{\text{IN}} \) and \( \text{OUT}_0 \sqsubseteq^* \overline{\text{OUT}} \) is easy
  - Inductive step:
    - \( \text{IN}_{i+1}[n_i] = \text{IN}_i[n_i] \sqcup \bigcup_{p\rightarrow n_i \in E} \text{OUT}_i[p] \sqcup \overline{\text{IN}_i[n_i]} \sqcup \bigcup_{p\rightarrow n_i \in E} \overline{\text{OUT}[p]} \sqsubseteq \overline{\text{IN}[n_i]} \)
    - \( \text{OUT}_{i+1}[n_i] = \text{post}_L(n_i, \text{IN}_{i+1}[n_i]) \sqsubseteq \text{post}_L(n_i, \overline{\text{IN}[n_i]}) \sqsubseteq \overline{\text{OUT}[n_i]} \)
    - For any \( n \neq n_i \), \( \text{IN}_{i+1}[n] = \text{IN}_i[n] \sqsubseteq \overline{\text{IN}[n_i]} \)
Termination

• Why does this algorithm terminate?
Termination

- Why does this algorithm terminate?
  - In general, it doesn’t
Termination

- Why does this algorithm terminate?
  - In general, it doesn’t
- **Ascending chain condition** is sufficient.
  - A partial order $\sqsubseteq$ satisfies the ascending chain condition if any infinite ascending sequence
    \[ x_1 \sqsubseteq x_2 \sqsubseteq x_3 \sqsubseteq \ldots \]
    eventually stabilizes: for some $i$, we have $x_j = x_i$ for all $j \geq i$. 
Termination

- Why does this algorithm terminate?
  - In general, it doesn’t
- **Ascending chain condition** is sufficient.
  - A partial order $\sqsubseteq$ satisfies the ascending chain condition if any infinite ascending sequence

$$x_1 \sqsubseteq x_2 \sqsubseteq x_3 \sqsubseteq \ldots$$

  eventually stabilizes: for some $i$, we have $x_j = x_i$ for all $j \geq i$.
- Fact: $X$ is finite $\Rightarrow (2^X, \subseteq)$ and $(2^X, \supseteq)$ satisfy a.c.c. (**available expressions**)
Termination

- Why does this algorithm terminate?
  - In general, it doesn’t
- Ascending chain condition is sufficient.
  - A partial order $\subseteq$ satisfies the ascending chain condition if any infinite ascending sequence

\[
x_1 \subseteq x_2 \subseteq x_3 \subseteq \ldots
\]

eventually stabilizes: for some $i$, we have $x_j = x_i$ for all $j \geq i$.
- Fact: $X$ is finite $\Rightarrow (2^X, \subseteq)$ and $(2^X, \supseteq)$ satisfy a.c.c. (available expressions)
- Fact: $X$ is finite and $(\mathcal{L}, \subseteq)$ satisfies a.c.c. $\Rightarrow (X \rightarrow \mathcal{L}, \subseteq^*)$ satisfies a.c.c. (constant propagation)
Why does this algorithm terminate?
  - In general, it doesn’t

**Ascending chain condition** is sufficient.
  - A partial order $\sqsubseteq$ satisfies the ascending chain condition if any infinite ascending sequence

$$x_1 \sqsubseteq x_2 \sqsubseteq x_3 \sqsubseteq \ldots$$

eventually stabilizes: for some $i$, we have $x_j = x_i$ for all $j \geq i$.

- **Fact:** $X$ is finite $\Rightarrow (2^X, \subseteq)$ and $(2^X, \supseteq)$ satisfy a.c.c. *(available expressions)*
- **Fact:** $X$ is finite and $(\mathcal{L}, \sqsubseteq)$ satisfies a.c.c. $\Rightarrow (X \rightarrow \mathcal{L}, \sqsubseteq^*)$ satisfies a.c.c. *(constant propagation)*

**Termination argument:**
  - If $(\mathcal{L}, \sqsubseteq)$ satisfies a.c.c., so does the space of annotations $(N \rightarrow \mathcal{L}, \sqsubseteq^*)$
  - $\text{OUT}_0 \sqsubseteq^* \text{OUT}_1 \sqsubseteq^* \ldots$, where $\text{OUT}_i$ is the $\text{OUT}$ annotation at iteration $i$
  - This sequence eventually stabilizes $\Rightarrow$ algorithm terminates
Local vs. Global constraints

- We had two specifications for available expressions
  - **Global:** $e$ available at entry of $n$ iff for every path from $s$ to $n$ in $G$:
    1. the expression $e$ is evaluated along the path
    2. after the last evaluation of $e$ along the path, no variables in $e$ are overwritten
  - **Local:** $\text{IN}$, $\text{OUT}$ is least annotation such that
    1. $\text{IN}[s] = \top$
    2. For all $n \in N$, $\text{post}_L(n, \text{IN}[n]) \sqsubseteq \text{OUT}[n]$
    3. For all $p \rightarrow n \in E$, $\text{OUT}[p] \sqsubseteq \text{IN}(n)$

- *Why are these specifications the same?*
Coincidence

- Let \((\mathcal{L}, \sqsubseteq, \sqcup, \bot, \top)\) be an abstract domain and let \(\text{post}_\mathcal{L}\) be a transfer function.
- “Global specification” is formulated as join over paths:

\[
\text{JOP}[n] = \bigsqcup_{\pi \in \text{Path}(s, n)} \text{post}_\mathcal{L}(\pi, \top)
\]

where \(\text{Path}(s, n)\) denotes set of paths from \(s\) to \(n\), and \(\text{post}_\mathcal{L}\) is extended to paths by taking

\[
\text{post}_\mathcal{L}(n_1 n_2 \ldots n_k, \top) = \text{post}_\mathcal{L}(n_k, \ldots, \text{post}_\mathcal{L}(n_1, \top))
\]
Coincidence

- Let \((\mathcal{L}, \sqsubseteq, \sqcup, \bot, \top)\) be an abstract domain and let \(\text{post}_\mathcal{L}\) be a transfer function.
  - “Global specification” is formulated as join over paths:
    \[
    \text{JOP}[n] = \bigcup_{\pi \in \text{Path}(s, n)} \text{post}_\mathcal{L}(\pi, \top)
    \]
    where \(\text{Path}(s, n)\) denotes set of paths from \(s\) to \(n\), and \(\text{post}_\mathcal{L}\) is extended to paths by taking
    \[
    \text{post}_\mathcal{L}(n_1 n_2 \ldots n_k, \top) = \text{post}_\mathcal{L}(n_k, \ldots, \text{post}_\mathcal{L}(n_1, \top))
    \]

- Coincidence theorem (Kildall, Kam & Ullman): let \((\mathcal{L}, \sqsubseteq, \sqcup, \bot, \top)\) be an abstract domain satisfying the a.c.c., \(\text{post}_\mathcal{L}\) be a \textit{distributive} transfer function, and IN/OUT be least solution to
  1. \(\text{IN}[s] = \top\)
  2. For all \(n \in N\), \(\text{post}_\mathcal{L}(n, \text{IN}[n]) \sqsubseteq \text{OUT}[n]\)
  3. For all \(p \rightarrow n \in E\), \(\text{OUT}[p] \sqsubseteq \text{IN}(n)\)

Then for all \(n\), \(\text{JOP}[n] = \text{IN}[n]\).
Coincidence

Let \((\mathcal{L}, \sqsubseteq, \sqcup, \bot, \top)\) be an abstract domain and let \(\text{post}_\mathcal{L}\) be a transfer function.

- “Global specification” is formulated as join over paths:

\[
\text{JOP}[n] = \bigcup_{\pi \in \text{Path}(s, n)} \text{post}_\mathcal{L}(\pi, \top)
\]

where \(\text{Path}(s, n)\) denotes set of paths from \(s\) to \(n\), and \(\text{post}_\mathcal{L}\) is extended to paths by taking

\[
\text{post}_\mathcal{L}(n_1n_2\ldots n_k, \top) = \text{post}_\mathcal{L}(n_k, \ldots, \text{post}_\mathcal{L}(n_1, \top))
\]

- Coincidence theorem (Kildall, Kam & Ullman): let \((\mathcal{L}, \sqsubseteq, \sqcup, \bot, \top)\) be an abstract domain satisfying the a.c.c., \(\text{post}_\mathcal{L}\) be a distributive transfer function, and \(\text{IN/OUT}\) be least solution to

1. \(\text{IN}[s] = \top\)
2. For all \(n \in N, \text{post}_\mathcal{L}(n, \text{IN}[n]) \sqsubseteq \text{OUT}[n]\)
3. For all \(p \to n \in E, \text{OUT}[p] \sqsubseteq \text{IN}(n)\)

Then for all \(n\), \(\text{JOP}[n] = \text{IN}[n]\).

- \(\text{post}_\mathcal{L}\) is distributive if for all \(x, y \in \mathcal{L}\), \(\text{post}_\mathcal{L}(n, x \sqcup y) = \text{post}_\mathcal{L}(n, x) \sqcup \text{post}_\mathcal{L}(n, y)\)
Available expressions

Recall transfer function $post_{AE}$ for available expressions:

$$post_{AE}(x = e, E) = \{ e' \in (E \cup \{ e \}) : x \text{ not in } e' \}$$

$post_{AE}$ is distributive:

$$post_{AE}(x = e, E_1 \cap E_2) = \{ e' \in ((E_1 \cap E_2) \cup \{ e \}) : x \text{ not in } e' \}$$
$$= \{ e' \in E_1 \cup \{ e \} : x \text{ not in } e' \} \cap \{ e' \in (E_2 \cup \{ e \}) : x \text{ not in } e' \}$$
$$= post_{AE}(x = e, E_1) \cap post_{AE}(x = e, E_2)$$
Constant propagation

Is $\text{post}_{cp}$ distributive?
Is $post_{CP}$ distributive?

$$post_{CP}(x := x + y, \{x \mapsto 0, y \mapsto 1\} \sqcup \{x \mapsto 1, y \mapsto 0\}) = post_{CP}(x := x + y, \{x \mapsto T, y \mapsto T\})$$

$$= \{x \mapsto T, y \mapsto T\}$$
Constant propagation

Is $\text{post}_{\text{CP}}$ distributive?

$$\text{post}_{\text{CP}}(x := x + y, \{x \mapsto 0, y \mapsto 1\} \sqcup \{x \mapsto 1, y \mapsto 0\}) = \text{post}_{\text{CP}}(x := x + y, \{x \mapsto T, y \mapsto T\})$$

$$= \{x \mapsto T, y \mapsto T\}$$

$$\text{post}_{\text{CP}}(x := x + y, \{x \mapsto 0, y \mapsto 1\}) = \{x \mapsto 1, y \mapsto 1\}$$

$$\text{post}_{\text{CP}}(x := x + y, \{x \mapsto 1, y \mapsto 0\}) = \{x \mapsto 1, y \mapsto 0\}$$

$$\{x \mapsto 1, y \mapsto 1\} \sqcup \{x \mapsto 1, y \mapsto 0\} = \{x \mapsto 1, y \mapsto T\}$$
Gen/kill analyses

• Suppose we have a finite set of data flow “facts”
• Elements of the abstract domain are sets of facts
• For each basic block \( n \), associate a set of generated facts \( \text{gen}(n) \) and killed facts \( \text{kill}(n) \)
• Define \( \text{post}_L(n, F) = (F \setminus \text{kill}(n)) \cup \text{gen}(n) \).
Suppose we have a finite set of data flow “facts”

Elements of the abstract domain are sets of facts

For each basic block \( n \), associate a set of generated facts \( \text{gen}(n) \) and killed facts \( \text{kill}(n) \)

Define \( \text{post}_L(n,F) = (F \setminus \text{kill}(n)) \cup \text{gen}(n) \).

The order on sets of facts may be \( \subseteq \) or \( \supseteq \)

- \( \subseteq \) used for existential analyses: a fact holds at \( n \) if it holds along some path to \( n \)
  - E.g., a variable is possibly-uninitialized at \( n \) if it is possibly-uninitialized along some path to \( n \).
- \( \supseteq \) used for universal analyses: a fact holds at \( n \) if it holds along all paths to \( n \)
  - E.g., an expression is available at \( n \) if it is available along all paths to \( n \).
Gen/kill analyses

- Suppose we have a finite set of data flow “facts”
- Elements of the abstract domain are sets of facts
- For each basic block \( n \), associate a set of generated facts \( \text{gen}(n) \) and killed facts \( \text{kill}(n) \)
- Define \( \text{post}_L(n, F) = (F \setminus \text{kill}(n)) \cup \text{gen}(n) \).
- The order on sets of facts may be \( \subseteq \) or \( \supseteq \)
  - \( \subseteq \) used for existential analyses: a fact holds at \( n \) if it holds along some path to \( n \)
    - E.g., a variable is possibly-uninitialized at \( n \) if it is possibly-uninitialized along some path to \( n \).
  - \( \supseteq \) used for universal analyses: a fact holds at \( n \) if it holds along all paths to \( n \)
    - E.g., an expression is available at \( n \) if it is available along all paths to \( n \)
- In either case, \( \text{post}_L \) is monotone and distributive
  \[
  \text{post}_L(n, F \cup G) = ((F \cup G) \setminus \text{kill}(n)) \cup \text{gen}(n)
  = ((F \setminus \text{kill}(n)) \cup (G \setminus \text{kill}(n))) \cup \text{gen}(n)
  = ((F \setminus \text{kill}(n)) \cup \text{gen}(n)) \cup (((G \setminus \text{kill}(n))) \cup \text{gen}(n))
  = \text{post}_L(n, F) \cup \text{post}_L(n, G)
  \]
A variable $x$ is **possibly-uninitialized** at a location $n$ if there is some path from start to $n$ along which $x$ is never written to.

If $n$ *uses* an uninitialized variable, that could indicate undefined behavior

- Can catch these errors at compile time using possibly-uninitialized variable analysis
- E.g. javac does this by default

Possibly-uninitialized variables as a dataflow analysis problem:
Possibly-uninitialized variables analysis

- A variable $x$ is **possibly-uninitialized** at a location $n$ if there is some path from start to $n$ along which $x$ is never written to.
- If $n$ uses an uninitialized variable, that could indicate undefined behavior
  - Can catch these errors at compile time using possibly-uninitialized variable analysis
  - E.g. javac does this by default
- Possibly-uninitialized variables as a dataflow analysis problem:
  - Abstract domain: $2^{\text{Var}}$ (each $V \in 2^{\text{Var}}$ represents a set of possibly-uninitialized vars)
    - *Existential* $\Rightarrow$ order is $\subseteq$, join is $\cup$, $\top$ is Var, $\bot$ is $\emptyset$
A variable $x$ is **possibly-uninitialized** at a location $n$ if there is some path from start to $n$ along which $x$ is never written to.

If $n$ uses an uninitialized variable, that could indicate undefined behavior
- Can catch these errors at compile time using possibly-uninitialized variable analysis
- E.g. javac does this by default

Possibly-unintialized variables as a dataflow analysis problem:
- Abstract domain: $2^{\mathit{Var}}$ (each $V \in 2^{\mathit{Var}}$ represents a set of possibly-uninitialized vars)
  - **Existential** $\Rightarrow$ order is $\subseteq$, join is $\cup$, $\top$ is $\mathit{Var}$, $\bot$ is $\emptyset$
  - $\mathbf{kill}(x := e) = \{x\}$
  - $\mathbf{gen}(x := e) = \emptyset$
Reaching definitions analysis

- A definition is a pair \((n, x)\) consisting of a basic block \(n\), and a variable \(x\) such that \(n\) contains an assignment to \(x\).
- We say that a definition \((n, x)\) reaches a node \(m\) if there is a path from start to \(m\) such that the latest definition of \(x\) along the path is at \(n\).
- Reaching definitions as a data flow analysis:
Reaching definitions analysis

- A definition is a pair \((n, x)\) consisting of a basic block \(n\), and a variable \(x\) such that \(n\) contains an assignment to \(x\).
- We say that a definition \((n, x)\) reaches a node \(m\) if there is a path from start to \(m\) such that the latest definition of \(x\) along the path is at \(n\).
- Reaching definitions as a data flow analysis:
  - Abstract domain: \(2^{N \times \text{Var}}\)
    - \text{Existential} \Rightarrow \text{order is } \subseteq, \text{join is } \cup, \top \text{ is } N \times \text{Var}, \bot \text{ is } \emptyset
  - \text{kill}(n) = \{(m, x) : m \in N, (x := e) \text{ in } n\}
  - \text{gen}(n) = \{(n, x) : (x := e) \text{ in } n\}
Wrap-up

- In a compiler, program analysis is used to inform optimization
  - Outside of compilers: verification, testing, software understanding...
- Dataflow analysis is a particular *family* of program analyses, which operates by solving a constraint system over an ordered set
  - Gen/kill analysis are a sub-family with nice properties
  - The basic idea of solving constraints systems over ordered sets appears in lots of different places!
    - Parsing – computation of first, follow, nullable
    - Networking – computing shortest paths
    - Automated planning – distance-to-goal estimation
    - ...