Generic (forward) dataflow analysis algorithm

• Given:
  • Abstract domain \((\mathcal{L}, \sqsubseteq, \sqcup, \bot, \top)\)
  • Transfer function
    \(\text{post}_\mathcal{L} : \text{Basic Block} \times \mathcal{L} \rightarrow \mathcal{L}\)
  • Control flow graph \(G = (N, E, s)\)

• Compute: least function \(f\) such that
  1. \(f(s) = \top\)
  2. For all \(p \rightarrow n \in E, \text{post}_\mathcal{L}(p, f(p)) \sqsubseteq f(n)\)
Generic (forward) dataflow analysis algorithm

- Given:
  - Abstract domain \((\mathcal{L}, \sqsubseteq, \sqcup, \bot, \top)\)
  - Transfer function \(\text{post}_\mathcal{L} : \text{Basic Block} \times \mathcal{L} \rightarrow \mathcal{L}\)
  - Control flow graph \(G = (N, E, s)\)

- Compute: least function \(f\) such that
  1. \(f(s) = \top\)
  2. For all \(p \rightarrow n \in E, \text{post}_\mathcal{L}(p, f(p)) \sqsubseteq f(n)\)

```plaintext
f(s) \leftarrow \top;
f(n) = \bot \text{ for all other nodes};
work \leftarrow N \setminus \{s\};
while work \neq \emptyset do
    \begin{align*}
    \text{Pick some } n \text{ from work; } \\
    \text{work } &\leftarrow \text{work } \setminus \{ n \} ; \\
    \text{work } &\leftarrow \text{work } \cup \text{succ}(n) \\
    \end{align*}
    v \leftarrow \bigcup_{p \in \text{pred}(n)} \text{post}_\mathcal{L}(p, f(p));
    \begin{align*}
    \text{if } v \neq f(n) \text{ then } \\
    f(n) &\leftarrow v; \text{ } \\
    \text{work } &\leftarrow \text{work } \cup \text{succ}(n)
    \end{align*}
```

Invariants:
- work contains all \(n \in N\) that may violate their constraints (\(\text{post}_\mathcal{L}(p, f(p)) \nless \top\) for some \(p \rightarrow n \in E\))
- Use \(f_i\) to denote \(f\) on the \(i\)th iteration and \(f\) to denote least solution to the constraint system. Then for all \(n\), \(f_1(n) \less f_2(n) \less f_3(n) \less \cdots\) must eventually stabilize.

Termination:
- Why does this algorithm terminate?
  - Ascending chain condition for each \(n\), \(f_1(n) \less f_2(n) \less f_3(n) \less \cdots\) must eventually stabilize.
Generic (forward) dataflow analysis algorithm

- Given:
  - Abstract domain \((\mathcal{L}, \sqsubseteq, \sqcup, \bot, \top)\)
  - Transfer function \(\text{post}_{\mathcal{L}}: \text{Basic Block} \times \mathcal{L} \rightarrow \mathcal{L}\)
  - Control flow graph \(G = (N, E, s)\)

- Compute: least function \(f\) such that
  1. \(f(s) = \top\)
  2. For all \(p \rightarrow n \in E, \text{post}_{\mathcal{L}}(p, f(p)) \sqsubseteq f(n)\)

Invariants:

- \(\text{work}\) contains all \(n \in N\) that may violate their constraints \((\text{post}(p, f(p)) \not\sqsubseteq f(n)\) for some \(p \rightarrow n \in E)\)

- Use \(f_i\) to denote \(f\) on the \(i\)th iteration and \(f^*\) to denote least solution to the constraint system. Then for all \(n, f_i(n) \sqsubseteq f^*(n)\).

```
f(s) \leftarrow \top;
f(n) = \bot\) for all other nodes;
\text{work} \leftarrow N \setminus \{s\};
while work \neq \emptyset do
  Pick some \(n\) from work;
  work \leftarrow work \setminus \{n\};
  v \leftarrow \bigsqcup_{p \in \text{pred}(n)} \text{post}_{\mathcal{L}}(p, f(p)) ;
  if v \neq f(n) then
    f(n) \leftarrow v;
    work \leftarrow work \cup \text{succ}(n)
```
Generic (forward) dataflow analysis algorithm

Given:
- Abstract domain \((\mathcal{L}, \sqsubseteq, \sqcup, \bot, \top)\)
- Transfer function \(\text{post}_\mathcal{L} : \text{Basic Block} \times \mathcal{L} \to \mathcal{L}\)
- Control flow graph \(G = (N, E, s)\)

Compute: least function \(f\) such that
1. \(f(s) = \top\)
2. For all \(p \to n \in E\), \(\text{post}_\mathcal{L}(p, f(p)) \sqsubseteq f(n)\)

\[f(s) \leftarrow \top;\]
\[f(n) = \bot\text{ for all other nodes;}\]

\[\text{work} \leftarrow N \setminus \{s\};\]
\[\text{while } \text{work} \neq \emptyset \text{ do}\]
\[\text{Pick some } n \text{ from work;}\]
\[\text{work} \leftarrow \text{work} \setminus \{n\};\]
\[v \leftarrow \bigsqcup_{p \in \text{pred}(n)} \text{post}_\mathcal{L}(p, f(p));\]
\[\text{if } v \neq f(n) \text{ then}\]
\[f(n) \leftarrow v;\]
\[\text{work} \leftarrow \text{work} \cup \text{succ}(n)\]

Invariants:
- \(\text{work}\) contains all \(n \in N\) that may violate their constraints \((\text{post}(p, f(p)) \not\sqsubseteq f(n)\) for some \(p \to n \in E)\)
- Use \(f_i\) to denote \(f\) on the \(i\)th iteration and \(f^*\) to denote least solution to the constraint system. Then for all \(n\), \(f_i(n) \sqsubseteq f^*(n)\).

Termination:
- Why does this algorithm terminate?
Generic (forward) dataflow analysis algorithm

• Given:
  • Abstract domain \((\mathcal{L}, \sqsubseteq, \sqcup, \bot, \top)\)
  • Transfer function \(post_\mathcal{L} : \text{Basic Block} \times \mathcal{L} \to \mathcal{L}\)
  • Control flow graph \(G = (N, E, s)\)

• Compute: least function \(f\) such that
  1. \(f(s) = \top\)
  2. For all \(p \to n \in E, post_\mathcal{L}(p, f(p)) \sqsubseteq f(n)\)

\[
f(s) \leftarrow \top; \\
f(n) = \bot \text{ for all other nodes}; \\
work \leftarrow N \setminus \{s\}; \\
\text{while } work \neq \emptyset \text{ do} \\
\quad \text{Pick some } n \text{ from work; } \\
\quad work \leftarrow work \setminus \{n\}; \\
\quad v \leftarrow \bigsqcup_{p \in \text{pred}(n)} post_\mathcal{L}(p, f(p)); \\
\quad \text{if } v \neq f(n) \text{ then} \\
\quad \quad f(n) \leftarrow v; \\
\quad \quad work \leftarrow work \cup \text{succ}(n)
\]

Invariants:
• \(work\) contains all \(n \in N\) that may violate their constraints \((post(p, f(p)) \not\sqsubseteq f(n)\) for some \(p \to n \in E)\)
• Use \(f_i\) to denote \(f\) on the \(i\)th iteration and \(f^*\) to denote least solution to the constraint system. Then for all \(n, f_i(n) \sqsubseteq f^*(n)\).

Termination:
• Why does this algorithm terminate?
• Ascending chain condition \(\Rightarrow\) for each \(n, f_1(n) \sqsubseteq f_2(n) \sqsubseteq f_3(n) \sqsubseteq \ldots\) must eventually stabilize
Coincidence

• We had two specifications for available expressions
  • **Global**: \( e \in ae(n) \) 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**: \( ae \) is the *smallest* function such that
    • \( ae(s) = \emptyset \)
    • For each \( p \rightarrow n \in E \), \( post_{AE}(p, ae(p)) \supseteq ae(n) \)

• Why are these specifications the same?
Coincidence

- We had two specifications for available expressions
  - **Global**: $e \in ae(n)$ 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**: $ae$ is the *smallest* function such that
    - $ae(s) = \emptyset$
    - For each $p \rightarrow n \in E$, $post_{ae}(p, ae(p)) \supseteq ae(n)$

- **Why are these specifications the same?**

- **Coincidence theorem** (Kildall, Kam & Ullman): for any abstract domain $(\mathcal{L}, \sqsubseteq, \sqcup, \bot, \top)$ and distributive transfer function $post_{\mathcal{L}}$, the least solution $f$ to the constraint system
  1. $f(s) \sqsubseteq \top$
  2. For each $p \rightarrow n \in E$, $post_{\mathcal{L}}(p, f(p)) \sqsubseteq f(n)$

  coincides with the function $g(n) = \bigcup_{\pi \in \text{Path}(s,n)} post_{\mathcal{L}}(\pi, \top)$, where $post_{\mathcal{L}}$ is extended to paths by taking

  $$post_{\mathcal{L}}(n_1n_2...n_k, \top) = post_{\mathcal{L}}(n_{k-1}, ..., post_{\mathcal{L}}(n_1, \top))$$
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 $\text{post}_L(n, F) = (F \setminus \text{kill}(n)) \cup \text{gen}(n)$. 
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 $n$.
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 \( \text{Var} \), \( \bot \) is \( \emptyset \)
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 $\text{Var}$, $\bot$ is $\emptyset$
    - $\text{kill}(x := e) = \{x\}$
    - $\text{gen}(x := e) = \emptyset$
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 definitoin \((n, x)\) 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: \(2^{\mathbb{N} \times \text{Var}}\)
    - *Existential* ⇒ order is \(\subseteq\), join is \(\cup\), \(\top\) is \(\mathbb{N} \times \text{Var}\), \(\bot\) is \(\emptyset\)
    - \(\text{kill}(n) = \{(m, x) : m \in \mathbb{N}, (x := e) \text{ in } n\}\)
    - \(\text{gen}(n) = \{(n, x) : (x := e) \text{ in } n\}\)