Program Analysis
Lecture 4

Rayna Dimitrova

WS 2016/2017
Recap: Forward versus backward analysis

- **forward analysis** computes information about data that depends on the past of the program execution
- **backward analysis** computes information about data that depends on the future of the program execution

For $b \notin E$

$$X_b = \bigsqcup \{ f_p(X_p) \mid p \in \text{pred}(b) \}$$

$$X_b = \bigsqcup \{ f_s(X_s) \mid s \in \text{succ}(b) \}$$
Recap: May versus must analysis

- **May analyses** detect properties satisfied by at least one execution path to (or from) the entry (or exit) of a block. The computed information is an **overapproximation**: it may possibly be true in an actual execution.

- **Must analyses** detect properties satisfied by all paths of execution reaching (or leaving) the entry (or exit) of a block. All values detected by a must analysis are actually reached. The computed information is an **underapproximation**: it must definitely be true in actual executions.

**may analyses**

- often we define: $D = \mathcal{P}(A)$
- for suitably chosen $A$
- $\leq$ to be $\subseteq$
- $\sqcup$ to be $\cup$
- $\bot$ to be $\emptyset$

**must analyses**

- often we define: $D = \mathcal{P}(A)$
- for suitably chosen $A$
- $\leq$ to be $\supseteq$
- $\sqcup$ to be $\cap$
- $\bot$ to be $A$
Recap: A classification of program analyses

<table>
<thead>
<tr>
<th></th>
<th>may</th>
<th>must</th>
</tr>
</thead>
<tbody>
<tr>
<td>forward</td>
<td>reaching definitions</td>
<td>available expressions</td>
</tr>
<tr>
<td>backward</td>
<td>live variables</td>
<td>very busy expressions</td>
</tr>
</tbody>
</table>
Live variables analysis

A variable is **live** at the end of a block if there exists a path from that block to a use of the variable that does not re-define the variable.

**Goal:**
For each block, determine the variables that may be live at its exit.

**Applications:** **Dead code elimination:** eliminate assignment if the variable is not live after the corresponding block. Also, **register allocation** and **merging of variables** that are never simultaneously live.

**Classification:** We need a **backward analysis** that computes information about the future behavior.

We have to perform a **may analysis** overapproximating the information from the possible executions. The computed information is guaranteed to include the behaviour of each execution.
Live variables analysis: CFG example

$$G = (B, E, F), E = \{7\}$$

1. $[x := 2]$;
2. $[y := 4]$;
3. $[x := 1]$;
4. if $[y > x]$ {
   5. $[z := x]$;
5. } else {
   6. $[z := y \cdot y]$;
6. }
7. $[x := z]$

**Vars:** the set of variables appearing in the program

$$Vars = \{x, y, z\}$$

$$Vars(e)$$ is the set of variables occurring in expression $e$. 
Live variables analysis: lattice

\[(D, \leq) := (\mathcal{P}(Vars), \subseteq), \bigcup X := \bigcup X, \bot := \emptyset\]

An element \(d \in \mathcal{P}(Vars)\) denotes a set of variables.

The lattice satisfies (ACC), since it is finite.
Live variables analysis: transfer functions

initial extremal value \( i := \emptyset \)

transfer functions \( f_b : D \rightarrow D \)

\[
f_b : \mathcal{P}(Vars) \rightarrow \mathcal{P}(Vars)
\]

\[
f_b(X) := (X \setminus \text{kill}(b)) \cup \text{gen}(b)
\]

- **kill(b):** variables whose value is overwritten in block \( b \)

\[
\text{kill}(b) := \begin{cases} 
\{v\} & \text{if } b = [v := e]^l \\
\emptyset & \text{otherwise.}
\end{cases}
\]

- **gen(b):** variables used in block \( b \)

\[
\text{gen}(b) := \begin{cases} 
\text{Vars}(e) & \text{if } b = [v := e]^l, \\
\text{Vars}(e) & \text{if } b = [e]^l \text{ for condition } e, \\
\emptyset & \text{otherwise.}
\end{cases}
\]

The transfer functions are monotonic.
## Example

<table>
<thead>
<tr>
<th>block $b$</th>
<th>$\text{kill}(b)$</th>
<th>$\text{gen}(b)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[x := 2]^1$</td>
<td>${x}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$[y := 4]^2$</td>
<td>${y}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$[x := 1]^3$</td>
<td>${x}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$[y &gt; 0]^4$</td>
<td>$\emptyset$</td>
<td>${y}$</td>
</tr>
<tr>
<td>$[z := x]^5$</td>
<td>${z}$</td>
<td>${x}$</td>
</tr>
<tr>
<td>$[z := y \ast y]^6$</td>
<td>${z}$</td>
<td>${y}$</td>
</tr>
<tr>
<td>$[x := z]^7$</td>
<td>${x}$</td>
<td>${z}$</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
X_1 &= f_2(X_2) & = X_2 \setminus \{y\} \\
X_2 &= f_3(X_3) & = X_3 \setminus \{x\} \\
X_3 &= f_4(X_4) & = X_4 \cup \{y\} \\
X_4 &= f_5(X_5) \sqcup f_6(X_6) & = (X_5 \setminus \{z\}) \cup \{x\} \cup (X_6 \setminus \{z\}) \cup \{y\} \\
X_5 &= f_7(X_7) & = (X_7 \setminus \{x\}) \cup \{z\} \\
X_6 &= f_7(X_7) & = (X_7 \setminus \{x\}) \cup \{z\} \\
X_7 &= i & = \emptyset
\end{align*}
\]
Example

\[
\begin{align*}
X_1 &= f_2(X_2) &= X_2 \setminus \{y\} \\
X_2 &= f_3(X_3) &= X_3 \setminus \{x\} \\
X_3 &= f_4(X_4) &= X_4 \cup \{y\} \\
X_4 &= f_5(X_5) \sqcup f_6(X_6) &= (X_5 \setminus \{z\}) \cup \{x\} \cup (X_6 \setminus \{z\}) \cup \{y\} \\
X_5 &= f_7(X_7) &= (X_7 \setminus \{x\}) \cup \{z\} \\
X_6 &= f_7(X_7) &= (X_7 \setminus \{x\}) \cup \{z\} \\
X_7 &= i &= \emptyset
\end{align*}
\]

compute a solution by computing the least fixpoint of the function

\[
g_S : \mathcal{P}(Vars)^7 \rightarrow \mathcal{P}(Vars)^7
\]

in \((\mathcal{P}(Vars)^7, \subseteq^7)\) starting at \((\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset)\)
Example

\[ X_1 = f_2(X_2) = X_2 \setminus \{y\} \]
\[ X_2 = f_3(X_3) = X_3 \setminus \{x\} \]
\[ X_3 = f_4(X_4) = X_4 \cup \{y\} \]
\[ X_4 = f_5(X_5) \sqcup f_6(X_6) = (X_5 \setminus \{z\}) \cup \{x\} \cup (X_6 \setminus \{z\}) \cup \{y\} \]
\[ X_5 = f_7(X_7) = (X_7 \setminus \{x\}) \cup \{z\} \]
\[ X_6 = f_7(X_7) = (X_7 \setminus \{x\}) \cup \{z\} \]
\[ X_7 = i = \emptyset \]

The least solution is

\[ X_1 = \emptyset \]
\[ X_2 = \{y\} \]
\[ X_3 = X_4 = \{x, y\} \]
\[ X_5 = X_6 = \{z\} \]
\[ X_7 = \emptyset \]
Very busy expressions analysis

An expression is **very busy** at the end of a block if for every path starting at that block the expression must be used before the value of any variable occurring in it changes.

**Goal:** For each block in the program, determine which expressions must be very busy at the end of this block.

**Applications:** Evaluate the expression at the block and store its value for later use (**hoisting** the expression).

**Classification:** We need a backward analysis that computes information about the future behavior. We have to perform a **must analysis** underapproximating the information along all possible executions.
Very busy expressions analysis: CFG example

```plaintext
if [a > b]¹{
    [x := b − a]²;
    [y := a − b]³;
}
else {
    [y := b − a]⁴;
    [x := a − b]⁵;
}
[skip]⁶
```

\( G = (B, E, F) \), \( E = \{6\} \)

\( AExp \): set of non-trivial arithmetic expressions appearing in the program
\( AExp = \{b − a, a − b\} \)

\( Vars(e) \) is the set of variables occurring in expression \( e \).
\( AExp(e) \) is the set of sub-expressions of expression \( e \).
Very busy expressions analysis: lattice

$$(D, \leq) := (\mathcal{P}(AExp), \supseteq), \sqcup X := \bigcap X, \bot := AExp$$

An element $d \in \mathcal{P}(AExp)$ denotes a set of expressions.

The lattice satisfies (ACC), since it is finite.
Very busy expressions analysis: transfer functions

initial extremal value $i := \emptyset$

transfer functions $f_b : D \rightarrow D$

$$f_b : \mathcal{P}(AExp) \rightarrow \mathcal{P}(AExp)$$

$$f_b(X) := (X \setminus \text{kill}(b)) \cup \text{gen}(b)$$

- $\text{kill}(b)$: expressions whose value is modified by $b$

$$\text{kill}(b) := \begin{cases} 
\{ e' \in AExp \mid v \in \text{Vars}(e') \} & \text{if } b = [v := e]^l \\
\emptyset & \text{if } b = [\text{read}(v)]^l, \\
\emptyset & \text{otherwise.}
\end{cases}$$

- $\text{gen}(b)$: expressions used in block $b$

$$\text{gen}(b) := \begin{cases} 
AExp(e) & \text{if } b = [v := e]^l, \\
AExp(e) & \text{if } b = [e]^l \text{ for condition } e, \\
\emptyset & \text{if } b = [\text{print}(e)]^l, \\
\emptyset & \text{otherwise.}
\end{cases}$$

The transfer functions are monotonic.
Example

<table>
<thead>
<tr>
<th>block $b$</th>
<th>$kill(b)$</th>
<th>$gen(b)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[a &gt; b]^1$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$[x := b - a]^2$</td>
<td>$\emptyset$</td>
<td>${b - a}$</td>
</tr>
<tr>
<td>$[y := a - b]^3$</td>
<td>$\emptyset$</td>
<td>${a - b}$</td>
</tr>
<tr>
<td>$[y := b - a]^4$</td>
<td>$\emptyset$</td>
<td>${b - a}$</td>
</tr>
<tr>
<td>$[x := a - b]^5$</td>
<td>$\emptyset$</td>
<td>${a - b}$</td>
</tr>
<tr>
<td>$[skip]^6$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

$X_1 = f_2(X_2) \sqcup f_4(X_4) = (X_2 \cup \{b - a\}) \cap (X_4 \cup \{b - a\})$

$X_2 = f_3(X_3)$

$X_3 = f_6(X_6)$

$X_4 = f_5(X_5)$

$X_5 = f_6(X_6)$

$X_6 = i$

$X_1 = f_2(X_2) \sqcup f_4(X_4) = (X_2 \cup \{b - a\}) \cap (X_4 \cup \{b - a\})$

$X_2 = f_3(X_3)$

$X_3 = f_6(X_6)$

$X_4 = f_5(X_5)$

$X_5 = f_6(X_6)$

$X_6 = i$

$X_1 = f_2(X_2) \sqcup f_4(X_4) = (X_2 \cup \{b - a\}) \cap (X_4 \cup \{b - a\})$

$X_2 = f_3(X_3)$

$X_3 = f_6(X_6)$

$X_4 = f_5(X_5)$

$X_5 = f_6(X_6)$

$X_6 = i$

$X_1 = f_2(X_2) \sqcup f_4(X_4) = (X_2 \cup \{b - a\}) \cap (X_4 \cup \{b - a\})$

$X_2 = f_3(X_3)$

$X_3 = f_6(X_6)$

$X_4 = f_5(X_5)$

$X_5 = f_6(X_6)$

$X_6 = i$
Example

\[
\begin{align*}
X_1 &= f_2(X_2) \sqcup f_4(X_4) = (X_2 \cup \{b - a\}) \cap (X_4 \cup \{b - a\}) \\
X_2 &= f_3(X_3) = X_3 \cup \{a - b\} \\
X_3 &= f_6(X_6) = X_6 \\
X_4 &= f_5(X_5) = X_5 \cup \{a - b\} \\
X_5 &= f_6(X_6) = X_6 \\
X_6 &= i = \emptyset
\end{align*}
\]

compute a solution by computing the least fixpoint of the function

\[
g_S : \mathcal{P}(AExp)^6 \to \mathcal{P}(AExp)^6
\]

in \((\mathcal{P}(AExp)^6, \supseteq^6)\) starting at \((AExp, AExp, AExp, AExp, AExp, AExp, AExp)\)
Example

\[
\begin{align*}
X_1 &= f_2(X_2) \cup f_4(X_4) = (X_2 \cup \{b - a\}) \cap (X_4 \cup \{b - a\}) \\
X_2 &= f_3(X_3) = X_3 \cup \{a - b\} \\
X_3 &= f_6(X_6) = X_6 \\
X_4 &= f_5(X_5) = X_5 \cup \{a - b\} \\
X_5 &= f_6(X_6) = X_6 \\
X_6 &= \emptyset
\end{align*}
\]

the least solution with respect to \(\supseteq^6\) is

\[
\begin{align*}
X_1 &= \{a - b, b - a\} \\
X_2 &= X_4 = \{a - b\} \\
X_3 &= X_5 = \emptyset \\
X_6 &= \emptyset
\end{align*}
\]

We computed the least fixpoint with respect to \(\supseteq^6\), which is also the greatest fixpoint with respect to the dual lattice \((P(AExp)^6, \subseteq^6)\).
### Summary

<table>
<thead>
<tr>
<th></th>
<th>reaching definitions</th>
<th>available expressions</th>
<th>live variables</th>
<th>very busy expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D$</td>
<td>$\mathcal{P}(Vars \times B \cup {?})$</td>
<td>$\mathcal{P}(AExp)$</td>
<td>$\mathcal{P}(Vars)$</td>
<td>$\mathcal{P}(AExp)$</td>
</tr>
<tr>
<td>$\leq$</td>
<td>$\subseteq$</td>
<td>$\supseteq$</td>
<td>$\subseteq$</td>
<td>$\supseteq$</td>
</tr>
<tr>
<td>$\sqcup$</td>
<td>$\cup$</td>
<td>$\cap$</td>
<td>$\cup$</td>
<td>$\cap$</td>
</tr>
<tr>
<td>$\perp$</td>
<td>$\emptyset$</td>
<td>$AExp$</td>
<td>$\emptyset$</td>
<td>$AExp$</td>
</tr>
<tr>
<td>approx</td>
<td>may</td>
<td>must</td>
<td>may</td>
<td>must</td>
</tr>
<tr>
<td>flow</td>
<td>forward</td>
<td>forward</td>
<td>backward</td>
<td>backward</td>
</tr>
<tr>
<td>$E$</td>
<td>initial</td>
<td>initial</td>
<td>final</td>
<td>final</td>
</tr>
<tr>
<td>$i$</td>
<td>${(v, ?) \mid v \in Vars}$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$f_b$</td>
<td>$f_b(X) = (X \setminus \text{kill}(b)) \cup \text{gen}(b)$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>
Algorithms for fixpoint computation

Let $S = (G, (D, \leq), i, \{f_b\}_{b \in B})$ be a dataflow system.

We can compute a solution of $S$ by computing the least fixpoint of the monotone function $g_S : D_S \rightarrow D_S$ in the lattice $(D_S, \leq_S)$.

Naive algorithm

$\overline{x} := (\bot, \ldots, \bot)$;

\textbf{do} { \\
\quad $\overline{t} := \overline{x}$; \\
\quad $\overline{x} := g_S(\overline{x})$; \\
\textbf{while} ($\overline{x} \neq \overline{t}$);

The naive algorithm recomputes the value of each variable $X_b$ in each step, even when the values of the variables on which the function $f_b$ depends have not changed in the previous step.

Chaotic iteration exploits the structure of the product lattice.
Algorithms for fixpoint computation

Let $S = (G, (D, \leq), i, \{f_b\}_{b \in B})$ be a dataflow system.

We can compute a solution of $S$ by computing the least fixpoint of the monotone function $g_S : D_S \to D_S$ in the lattice $(D_S, \leq_S)$.

Chaotic iteration

$X_1 := \bot, \ldots, X_{|B|} := \bot$;

\textbf{while } ($\exists b : X_b \neq F_b(X_1, \ldots, X_{|B|})$) \{ 
  $X_b := F_b(X_1, \ldots, X_{|B|})$;
\}

where we define for forward analyses

$$F_b(X_1, \ldots, X_{|B|}) = \begin{cases} 
  i & \text{if } b \in E \\
  \bigsqcup \{f_p(X_p) \mid p \in \text{pred}(b)\} & \text{otherwise.}
\end{cases}$$
Algorithms for fixpoint computation

Let \( S = (G, (D, \leq), i, \{f_b\}_{b \in B}) \) be a dataflow system.

We can compute a solution of \( S \) by computing the least fixpoint of the monotone function \( g_S : D_S \rightarrow D_S \) in the lattice \((D_S, \leq_S)\).

Chaotic iteration

\[
X_1 := \bot, \ldots, X_{|B|} := \bot;
\]
\[
\text{while } (\exists b : X_b \neq F_b(X_1, \ldots, X_{|B|})) \{
  X_b := F_b(X_1, \ldots, X_{|B|});
\}
\]

and for backward analyses

\[
F_b(X_1, \ldots, X_{|B|}) = \begin{cases} 
  i & \text{if } b \in E \\
  \bigsqcup \{f_s(X_s) \mid s \in \text{succ}(b)\} & \text{otherwise.}
\end{cases}
\]
Algorithms for fixpoint computation

Let $S = (G, (D, \leq), i, \{f_b\}_{b \in B})$ be a dataflow system.

We can compute a solution of $S$ by computing the least fixpoint of the monotone function $g_S : D_S \to D_S$ in the lattice $(D_S, \leq_S)$.

Chaotic iteration

\[
X_1 := \bot, \ldots, X_{|B|} := \bot;
\]

while $(\exists b : X_b \neq F_b(X_1, \ldots, X_{|B|}))$

\[
X_b := F_b(X_1, \ldots, X_{|B|});
\]

\}

For each block $b \in B$ we denote with $Dep_b \subseteq \{1, \ldots, |B|\}$ the set of blocks that depend on $b$, that is, the blocks in whose dataflow equations the variable $X_b$ occurs in the right-hand side.
Worklist-based algorithm

Input: dataflow system \( S = (G = (B, E, F), (D, \leq), i, \{f_b\}_{b \in B}) \)
Output: value for the variable \( X_b \) for each block \( b \in B \)

worklist \( W := \{1, \ldots, |B|\} \);
\( X_1 := \perp, \ldots, X_{|B|} := \perp \);

while \((W \neq \emptyset)\) {
    \( b := W.removeNext() \);
    \( y := F_b(X_1, \ldots, X_{|B|}) \);
    if \((y \neq X_b)\) {
        for \((b' \in Dep_b)\) \( W.add(b') \);
        \( X_b := y \);
    }
}

The operation \( removeNext() \) removes an element from the set \( W \), and \( add(b') \) adds the element \( b' \) to \( W \) (unless \( b' \) is already in \( W \)).
Example: available expressions analysis

\[ X_1 = i = \emptyset \]
\[ X_2 = f_1(X_1) = X_1 \cup \{a + b\} \]
\[ X_3 = f_2(X_2) \sqcup f_5(X_5) = (X_2 \cup \{a \ast b\}) \cap (X_5 \cup \{a + b\}) \]
\[ X_4 = f_3(X_3) = X_3 \cup \{a + b\} \]
\[ X_5 = f_4(X_4) = X_4 \setminus \{a + b, a \ast b, a + 1\} \]

compute a solution by computing the least fixpoint of the function

\[ g_S : \mathcal{P}(AExp)^5 \rightarrow \mathcal{P}(AExp)^5 \]

in \((\mathcal{P}(AExp)^5, \supseteq^5)\) starting at \((\bot, \bot, \bot, \bot, \bot)\), where \(\bot = AExp\)
Example: available expressions analysis

\[
\begin{align*}
X_1 &= i & = & \emptyset \\
X_2 &= f_1(X_1) & = & X_1 \cup \{a + b\} \\
X_3 &= f_2(X_2) \cup f_5(X_5) & = & (X_2 \cup \{a \ast b\}) \cap (X_5 \cup \{a + b\}) \\
X_4 &= f_3(X_3) & = & X_3 \cup \{a + b\} \\
X_5 &= f_4(X_4) & = & X_4 \setminus \{a + b, a \ast b, a + 1\}
\end{align*}
\]

\[
\begin{array}{c}
\rightarrow \quad [x := 5]^1 \\
\downarrow \\
[y := 1]^2 \\
\downarrow \\
[x > 1]^3 \\
\downarrow \\
[y := x \ast y]^4 \\
\downarrow \\
[x := x - 1]^5 \\
\end{array}
\]

initial

\[
\begin{align*}
Dep_1 &= \{2\} \\
Dep_2 &= \{3\} \\
Dep_3 &= \{4\} \\
Dep_4 &= \{5\} \\
Dep_5 &= \{3\}
\end{align*}
\]
Example: available expressions analysis

<table>
<thead>
<tr>
<th>$W$</th>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$X_3$</th>
<th>$X_4$</th>
<th>$X_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${1, 2, 3, 4, 5}$</td>
<td>$AEExp$</td>
<td>$AEExp$</td>
<td>$AEExp$</td>
<td>$AEExp$</td>
<td>$AEExp$</td>
</tr>
<tr>
<td>${2, 3, 4, 5}$</td>
<td>$\emptyset$</td>
<td>$AEExp$</td>
<td>$AEExp$</td>
<td>$AEExp$</td>
<td>$AEExp$</td>
</tr>
<tr>
<td>${3, 4, 5}$</td>
<td>$\emptyset$</td>
<td>${a + b}$</td>
<td>$AEExp$</td>
<td>$AEExp$</td>
<td>$AEExp$</td>
</tr>
<tr>
<td>${4, 5}$</td>
<td>$\emptyset$</td>
<td>${a + b}$</td>
<td>${a + b, a \ast b}$</td>
<td>$AEExp$</td>
<td>$AEExp$</td>
</tr>
<tr>
<td>${5}$</td>
<td>$\emptyset$</td>
<td>${a + b}$</td>
<td>${a + b, a \ast b}$</td>
<td>${a + b, a \ast b}$</td>
<td>$AEExp$</td>
</tr>
<tr>
<td>${3}$</td>
<td>$\emptyset$</td>
<td>${a + b}$</td>
<td>${a + b, a \ast b}$</td>
<td>${a + b, a \ast b}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>${4}$</td>
<td>$\emptyset$</td>
<td>${a + b}$</td>
<td>${a + b}$</td>
<td>${a + b, a \ast b}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>${5}$</td>
<td>$\emptyset$</td>
<td>${a + b}$</td>
<td>${a + b}$</td>
<td>${a + b}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>${a + b}$</td>
<td>${a + b}$</td>
<td>${a + b}$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>
Correctness of the algorithm

Theorem
The worklist algorithm always terminates and computes \( \text{lfp}(g_S) \).

Proof

(Termination)

- The initialization of \( W \) and \( x_b \) clearly terminates, as \( B \) is finite.
- Before the execution of the loop, \( W \) contains \( |B| \) elements. Each iteration deletes an element \( b \) of \( W \) and adds at most \( |B| \) new elements if it is the case that \( F_b(X_1, \ldots, X_{|B|}) \neq X_b \). Since by definition of the fixpoint computation sequence with monotonic functions we have \( X_b \leq F_b(X_1, \ldots, X_{|B|}) \), this means that new elements are added only when \( X_b < F_b(X_1, \ldots, X_{|B|}) \). Since \((D, \leq)\) satisfies (ACC), this can only happen a finite number of times. Thus, \( W \) will eventually become empty.
Correctness of the algorithm

Theorem
The worklist algorithm always terminates and computes \( \text{lfp}(g_S) \).

Proof

(Correctness)
Let \( \text{lfp}(g_S) = (X_1^{LFP}, \ldots, X_{|B|}^{LFP}) \) be the least fixpoint solution of \( S \). We prove correctness by showing that:

- The property \( X_b \leq X_b^{LFP} \) for every \( b \in B \) is a loop invariant.
- Upon termination, for all \( b \in B \) we have

\[
F_b(X_1, \ldots, X_{|B|}) = X_b.
\]

Since \( \text{lfp}(g_S) \) is the least solution of the system of dataflow equations, upon termination we have \( X_b^{LFP} \leq X_b \). Together with the invariant this implies that \( X_b = X_b^{LFP} \).
Proof

The property $X_b \leq X_b^{LFP}$ for every $b \in B$ is a loop invariant.

Before the first iteration of the loop we have that $X_b = \bot$ for all $b \in B$, and thus $X_b \leq X_b^{LFP}$ holds for all $b \in B$.

Suppose that $X_b \leq X_b^{LFP}$ holds for all $b \in B$ at the beginning of a loop iteration. If the loop iteration does not change the value of $X_b$, then the property continues to hold after this iteration. When the value of the considered $X_b$ is modified, let $X'_b$ be the new value for block $b$ at the end of the iteration (for all other blocks the corresponding variables remains unchanged in this iteration).

$$X'_b = F_b(X_1, \ldots, X_{|B|})$$
$$\leq F_b(X_1^{LFP}, \ldots, X_{|B|}^{LFP})$$
$$= X_b^{LFP}$$

The inequality follows from the fact that $F_b$ is monotonic, and the last equality from the fact that $\text{lfp}(g_S)$ is a solution of the dataflow equations. Thus, the property still holds after the loop iteration.
Proof

Upon termination, for all $b \in B$ we have $F_b(X_1, \ldots, X_{|B|}) = X_b$.

If $b \in E$, then $F_b(X_1, \ldots, X_{|B|}) = i$, and $X_b$ updated exactly once during the loop, when it is set to $i$, and never changed afterward. For $b \notin E$, consider the last time when a variable $X_p$ with $p \in \text{pred}(b)$ was updated. If this was before the loop, then $X_b$ was updated exactly once in the loop, and set to $F_b(X_1, \ldots, X_{|B|})$, and the value of $F_b(X_1, \ldots, X_{|B|})$ was not changed after that. If this was in the loop, then $b$ is added to $W$ at that iteration for the last time. Then, when $b$ was processed it was set to $F_b(X_1, \ldots, X_{|B|})$, and the value of $F_b(X_1, \ldots, X_{|B|})$ was not changed after that. Thus, in both cases, after termination we have $F_b(X_1, \ldots, X_{|B|}) = X_b$.

Remark: For backward analysis, $F_b(X_1, \ldots, X_{|B|})$ depends on $\text{succ}(b)$, and thus $\text{succ}(b)$ should be considered instead of $\text{pred}(b)$.
Complexity of the algorithm

We give an upper bound on the basic operations performed by the algorithm on input \( S = (G = (B, E, F), (D, \leq), i, \{f_b\}_{b \in B}) \).

Suppose that
- \(|B|\) is the number of blocks in \( B \);
- all elements of \( Dep_b \) can be enumerated in time \( O(|Dep_b|) \);
- \((D, \leq)\) has bounded height \( h \geq 1 \).

Then:
- \( W \) and \( X_b \) for \( b \in B \) are initialized in \( O(|B|) \) basic operations.
- Since each variable \( X_b \) is updated at most \( h \) times, each \( b' \) is added to \( W \) at most \( h \cdot |B| \) times. Assuming that computing \( F_b(X_1, \ldots, X_{|B|}) \) takes constant time and not counting the time needed to add values to \( W \), we have that until termination the loop requires \( O(h \cdot |B|^2) \) basic operations.
Precision of the fixpoint solution

Sometimes, the analysis based on systems of dataflow equations is imprecise due to the fact that it does not consider separately the individual paths of execution reaching a program location.

To see this, let us look at another example of dataflow analysis.
Constant propagation analysis

Goal: For each program point, determine whether a variable has a constant value whenever an execution reaches the respective block.

Applications: Useful for the constant folding optimization: a use of the variable can be replaced by the respective constant.

Classification: This is a forward must analysis.
Constant propagation analysis: lattice

\( (D, \leq) := ((\text{Vars} \to (\mathbb{Z} \cup \{?\})) \cup \{\perp\}, \leq) \)

- \( \text{Vars} \) is the set of variables appearing in the program

- an element \( d \in (\text{Vars} \to (\mathbb{Z} \cup \{?\})) \) is a function \( d : \text{Vars} \to (\mathbb{Z} \cup \{?\}) \) such that for a variable \( v \in \text{Vars} \)
  - \( d(v) \in \mathbb{Z} \) indicates that the value of \( v \) is the constant \( d(v) \)
  - \( d(v) = ? \) indicates that the variable \( v \) is non-constant

- the least element \( \perp \) indicates that no information is available

- the partial ordering \( \leq \) is defined by
  - \( \perp \leq d \) for all \( d \in D \)
  - \( d_1 \leq d_2 \) if and only if for every \( v \in \text{Vars} \) we have that either \( d_1(v), d_2(v) \in \mathbb{Z} \) and \( d_1(v) = d_2(v) \), or it holds that \( d_2(v) = ? \)
Constant propagation analysis: lattice

\((D, \leq) := ((\text{Vars} \rightarrow (\mathbb{Z} \cup \{?\})) \cup \{\bot\}, \leq)\)

**Example:** consider \(\text{Vars} = \{v\}\)

\[\begin{array}{cccccc}
\cdots & v \mapsto -2 & v \mapsto -1 & v \mapsto 0 & v \mapsto 1 & v \mapsto 2 & \cdots \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\
\end{array}\]

\(D\) is infinite. Since for a given program the set \(\text{Vars}\) is finite, for each program \((D, \leq)\) has bounded height and thus satisfies \((\text{ACC})\).
Constant propagation analysis: lattice

\[(D, \leq) := ((\text{Vars} \to (\mathbb{Z} \cup \{?\})) \cup \{\bot\}, \leq)\]

Then, the binary join operation \(\sqcup\) is such that

- \(\bot \sqcup d = d \sqcup \bot = d\), for all \(d \in D\)
- for all \(d_1, d_2 \in D \setminus \{\bot\}\) and every \(v \in \text{Vars}\),

\[(d_1 \sqcup d_2)(v) = \begin{cases} d_1(v) & \text{if } d_1(v) = d_2(v), \\ ? & \text{otherwise} \end{cases}\]

It is easy to check that \((D, \leq)\) is a complete lattice.
Constant propagation analysis: transfer functions

To define the transfer functions, we first define abstract evaluation of arithmetic expressions, which is a function $eval$ that maps each arithmetic expression $a$ and $d \in D \setminus \{\bot\}$ to an element of $\mathbb{Z} \cup \{?\}$.

- if $a$ is a variable $v$, then $eval(a, d) := d(v)$
- if $a$ is a constant $k$, then $eval(a, d) := k$
- if $a = a_1 \otimes a_2$, then
  - $eval(a, d) := \top$, if $eval(a_1, d) = \top$ or $eval(a_2, d) = \top$,
  - $eval(a, d) := \top$, if $eval(a_1, d) \otimes eval(a_2, d)$ is undefined,
  - $eval(a, d) := eval(a_1, d) \otimes eval(a_2, d)$, otherwise

For $d \in (Vars \rightarrow (\mathbb{Z} \cup \{\top\}))$, $x, v \in Vars$, and $n \in \mathbb{Z} \cup \{?\}$, define the function $d[x \mapsto n] : Vars \rightarrow (\mathbb{Z} \cup \{?\})$, such that

$$d[x \mapsto n](v) = \begin{cases} n & \text{if } v = x, \\ d(v) & \text{otherwise.} \end{cases}$$
Constant propagation analysis: transfer functions

- for \([x := a]^b\) let

\[
f_b(d) := \begin{cases} 
\bot & \text{if } d = \bot, \\
\ d[x \mapsto \text{eval}(a, d)] & \text{otherwise}
\end{cases}
\]

- for \([\text{read}(x)]^b\) let

\[
f_b(d) := \begin{cases} 
\bot & \text{if } d = \bot, \\
\ d[x \mapsto?] & \text{otherwise}
\end{cases}
\]

- for \([\text{print}(a)]^b\), \([\text{skip}]^b\), and for condition \([e]^b\) let

\[
f_b(d) = d \text{ for all } d \in D
\]
Constant propagation analysis: example

if \([z > 0] \) \{\n\[x := 2\];
\[y := 3\];
\}\nelse \{\n\[x := 3\];
\[y := 2\];
\}\n\[z := x + y\];
\[\text{skip}\]

\[G = (B, E, F), E = \{1\}\]

\[\xymatrix{\rightarrow & [z > 0] \ar@{->}[d] \ar@{->}[r] & \ar@{->}[d] \ar@{->}[r] & \text{initial} \\
[y := 3] & [y := 2] \ar@{->}[d] & [z := x + y] \ar@{->}[d] & \ar@{->}[d] \\
& & [\text{skip}] & [\text{skip}]}
\]
Constant propagation analysis: example

\[ z > 0 \]

\[ x := 2 \]

\[ x := 3 \]

\[ y := 3 \]

\[ y := 2 \]

\[ z := x + y \]

\[ \text{skip} \]

\[ f_b(\bot) = \bot \text{ for } b \in \{1, \ldots, 7\} \]

For \( d \in D \setminus \{\bot\} \) we have

\[
\begin{align*}
  f_1(d) &= d \\
  f_2(d) &= d[x \mapsto 2] \\
  f_3(d) &= d[y \mapsto 3] \\
  f_4(d) &= d[x \mapsto 3] \\
  f_5(d) &= d[y \mapsto 2] \\
  f_6(d) &= d[z \mapsto \text{eval}(x + y, d)] \\
  f_7(d) &= d
\end{align*}
\]
Constant propagation analysis: example

\((n_1, n_2, n_3)\) denotes the function \(d(x) = n_1, d(y) = n_2, d(z) = n_3\)

\[
\begin{align*}
X_1 &= (?, ?, ?) \\
X_2 &= f_1(X_1) \\
X_3 &= f_2(X_2) \\
X_4 &= f_1(X_1) \\
X_5 &= f_4(X_4) \\
X_6 &= f_3(X_3) \sqcup f_5(X_5) \\
X_7 &= f_6(X_6)
\end{align*}
\]

fixpoint computation

\[
\begin{align*}
(\bot, \bot, \bot, \bot, \bot, \bot, \bot)
((?, ?, ?), \bot, \bot, \bot, \bot, \bot, \bot)
((?, ?, ?), (?, ?, ?), \bot, (?, ?, ?), \bot, \bot, \bot)
((?, ?, ?), (?, ?, ?), (2, ?, ?), (?, ?, ?), (3, ?, ?), \bot, \bot)
((?, ?, ?), (?, ?, ?), (2, ?, ?), (?, ?, ?), (3, ?, ?), (?, ?, ?), \bot)
((?, ?, ?), (?, ?, ?), (2, ?, ?), (?, ?, ?), (3, ?, ?), (?, ?, ?), (?, ?, ?))
((?, ?, ?), (?, ?, ?), (2, ?, ?), (?, ?, ?), (3, ?, ?), (?, ?, ?), (?, ?, ?))
\end{align*}
\]
Constant propagation analysis: example

$(n_1, n_2, n_3)$ denotes the function $d(x) = n_1, d(y) = n_2, d(z) = n_3$

$X_1 = (?, ?, ?)$
$X_2 = f_1(X_1)$
$X_3 = f_2(X_2)$
$X_4 = f_1(X_1)$
$X_5 = f_4(X_4)$
$X_6 = f_3(X_3) \sqcup f_5(X_5)$
$X_7 = f_6(X_6)$

least fixpoint

$((?, ?, ?), (?, ?, ?), (2, ?, ?), (?, ?, ?), (3, ?, ?), (?, ?, ?), (?, ?, ?))$

From this solution we cannot infer the information that at the beginning of block 7 the value of $z$ is 5 regardless of the execution path.