Static Program Analysis

Lecture 5: Dataflow Analysis IV (Worklist Algorithm & MOP Solution)

Winter Semester 2016/17

Thomas Noll
Software Modeling and Verification Group
RWTH Aachen University

https://moves.rwth-aachen.de/teaching/ws-1617/spa/
Re-Scheduling of First Exam

New date: Tue 21 Feb, 15:00 - 17:00, AH 2/3
Recap: The Fixpoint Approach

Outline of Lecture 5

Recap: The Fixpoint Approach

Efficient Fixpoint Computation

The MOP Solution

Another Analysis: Constant Propagation
Recap: The Fixpoint Approach

The Fixpoint Theorem

Alfred Tarski (1901–1983)

Bronislaw Knaster (1893–1990)

Theorem (Fixpoint Theorem by Tarski and Knaster)

Let \((D, \sqsubseteq)\) be a complete lattice satisfying ACC and \(\Phi : D \rightarrow D\) monotonic. Then

\[
\text{fix}(\Phi) := \bigcup \{ \Phi^k(\bot) \mid k \in \mathbb{N} \}
\]

is the least fixpoint of \(\Phi\) where \(\Phi^0(d) := d\) and \(\Phi^{k+1}(d) := \Phi(\Phi^k(d))\).

Function requirements for dataflow analysis

All transfer functions must be a monotonic
Recap: The Fixpoint Approach

Dataflow Systems

Definition (Dataflow system)

A dataflow system $S = (\text{Lab}, E, F, (D, \sqsubseteq), \iota, \varphi)$ consists of

- a finite set of (program) labels $\text{Lab}$ (here: $\text{Lab}_c$),
- a set of extremal labels $E \subseteq \text{Lab}$ (here: $\{\text{init}(c)\}$ or $\text{final}(c)$),
- a flow relation $F \subseteq \text{Lab} \times \text{Lab}$ (here: $\text{flow}(c)$ or $\text{flow}^R(c)$),
- a complete lattice $(D, \sqsubseteq)$ satisfying ACC (with LUB operator $\sqcup$ and least element $\bot$),
- an extremal value $\iota \in D$ (for the extremal labels), and
- a collection of monotonic transfer functions $\{\varphi_l \mid l \in \text{Lab}\}$ of type $\varphi_l : D \to D$. 
Recap: The Fixpoint Approach

Dataflow Systems and Fixpoints

Definition (Dataflow equation system)

Given: dataflow system \( S = (Lab, E, F, (D, \sqsubseteq), \iota, \varphi) \), \( Lab = \{1, \ldots, n\} \) (w.l.o.g.)

- \( S \) determines the equation system (where \( l \in Lab \))
  \[
  AI_l = \begin{cases} 
  \iota & \text{if } l \in E \\
  \bigcup \{ \varphi_{l'}(AI_{l'}) \mid (l', l) \in F \} & \text{otherwise}
  \end{cases}
  \]

- \((d_1, \ldots, d_n) \in D^n\) is called a solution if
  \[
  d_l = \begin{cases} 
  \iota & \text{if } l \in E \\
  \bigcup \{ \varphi_{l'}(d_{l'}) \mid (l', l) \in F \} & \text{otherwise}
  \end{cases}
  \]

- \( S \) determines the transformation
  \[
  \Phi_S : D^n \rightarrow D^n : (d_1, \ldots, d_n) \mapsto (d'_1, \ldots, d'_n)
  \]
  where
  \[
  d'_l := \begin{cases} 
  \iota & \text{if } l \in E \\
  \bigcup \{ \varphi_{l'}(d_{l'}) \mid (l', l) \in F \} & \text{otherwise}
  \end{cases}
  \]

Corollary

\((d_1, \ldots, d_n) \in D^n\) solves the equation system iff it is a fixpoint of \( \Phi_S \)
Recap: The Fixpoint Approach

Solving Dataflow Problems by Fixpoint Iteration

Remarks:
- \((D, \sqsubseteq)\) being a complete lattice ensures that \(\Phi_S\) is well defined
- Since \((D, \sqsubseteq)\) is a complete lattice satisfying ACC, so is \((D^n, \sqsubseteq^n)\) (where \((d_1, \ldots, d_n) \sqsubseteq^n (d'_1, \ldots, d'_n)\) iff \(d_i \sqsubseteq d'_i\) for every \(1 \leq i \leq n\))
- Monotonicity of transfer functions \(\varphi_I\) in \((D, \sqsubseteq)\) implies monotonicity of \(\Phi_S\) in \((D^n, \sqsubseteq^n)\) (since \(\sqcup\) also monotonic)
- Thus the (least) fixpoint is effectively computable by iteration:

\[
\text{fix}(\Phi_S) = \bigsqcup \{\Phi_S^k(\perp_D^n) \mid k \in \mathbb{N}\}
\]

where \(\perp_D^n = (\perp_D, \ldots, \perp_D)\) \(n\) times
- If height of \((D, \sqsubseteq)\) is \(m\)
  \[\Rightarrow\] height of \((D^n, \sqsubseteq^n)\) is \(m \cdot n\)
  \[\Rightarrow\] fixpoint iteration requires at most \(m \cdot n\) steps
Efficient Fixpoint Computation

Outline of Lecture 5

Recap: The Fixpoint Approach

Efficient Fixpoint Computation

The MOP Solution

Another Analysis: Constant Propagation
A Worklist Algorithm I

Observation: fixpoint iteration re-computes every $A_i$ in every step
Efficient Fixpoint Computation

A Worklist Algorithm I

Observation: fixpoint iteration re-computes every $A_l$ in every step

$\implies$ redundant if $A_{l'}$ at no $F$-predecessor $l'$ changed

Algorithm 5.1 (Worklist algorithm)

Input: dataflow system $S = (\mathcal{L}, \mathcal{E}, \mathcal{F}, (\mathcal{D}, \sqsubseteq), \iota, \phi)$

Variables: $W \in (\mathcal{L} \times \mathcal{L})^*$, $\{A_l | l \in \mathcal{L}\}$

Procedure:

$W := \varepsilon$; for $(l, l')$ $\in \mathcal{F}$ do $W := W \cdot (l, l')$; % Initialise $W$ for $l \in \mathcal{L}$ do if $l \in \mathcal{E}$ then $A_l := \iota$ else $A_l := \bot$; % Initialise $A_l$

while $W \neq \varepsilon$ do $(l, l') := \text{head}(W)$; $W := \text{tail}(W)$; % Next control-flow edge

if $\phi_l(A_l) \not\sqsubseteq A_{l'}$ then % Fixpoint not yet reached

$A_{l'} := A_{l'} \sqcup \phi_l(A_l)$; % Update analysis information

for $(l', l'')$ $\in \mathcal{F}$ do if $(l', l'')$ not in $W$ then $W := (l', l'') \cdot W$; % Propagate modification

Output: $\{A_l | l \in \mathcal{L}\}$
Efficient Fixpoint Computation

A Worklist Algorithm I

Observation: fixpoint iteration re-computes every $AI_i$ in every step

$\Rightarrow$ redundant if $AI_i'$ at no $F$-predecessor $l'$ changed

$\Rightarrow$ optimisation by worklist over control-flow edges

Algorithm 5.1 (Worklist algorithm)

Input: dataflow system $S$ = $(Lab, E, F, (D, \sqsubseteq), \iota, \phi)$

Variables: $W \in (Lab \times Lab)^*$, $\{AI_l \in D | l \in Lab\}$

Procedure:

1. $W := \varepsilon$; for $(l, l') \in F$ do $W := W \cdot (l, l')$; % Initialise $W$ for $l \in Lab$

2. if $l \in E$ then $AI_l := \iota$ else $AI_l := \bot D$; % Initialise $AI$ for $l \in Lab$

3. while $W \neq \varepsilon$ do $(l, l') := \text{head}(W)$; $W := \text{tail}(W)$; % Next control-flow edge

4. if $\phi(l)(AI_l) \not\sqsubseteq AI_l'$ then % Fixpoint not yet reached

5. $AI_l' := AI_l' \sqcup \phi(l)(AI_l)$; % Update analysis information

6. for $(l', l'' \in F$ do if $(l', l'')$ not in $W$ then $W := (l', l'') \cdot W$; % Propagate modification

Output: $\{AI_l | l \in Lab\}$
Efficient Fixpoint Computation

A Worklist Algorithm I

Observation: fixpoint iteration re-computes every $AI_l$ in every step

$\implies$ redundant if $AI_{l'}$ at no $F$-predecessor $l'$ changed

$\implies$ optimisation by worklist over control-flow edges

Algorithm 5.1 (Worklist algorithm)

Input: dataflow system $S = (\text{Lab}, E, F, (D, \sqsubseteq), \iota, \varphi)$
Efficient Fixpoint Computation

A Worklist Algorithm I

Observation: fixpoint iteration re-computes every $AI_l$ in every step

$\implies$ redundant if $AI_{l'}$ at no $F$-predecessor $l'$ changed

$\implies$ optimisation by worklist over control-flow edges

Algorithm 5.1 (Worklist algorithm)

Input: dataflow system $S = (Lab, E, F, (D, \sqsubseteq), \iota, \varphi)$

Variables: $W \in (Lab \times Lab)^*$, $\{AI_l \in D \mid l \in Lab\}$
Efficient Fixpoint Computation

A Worklist Algorithm I

Observation: fixpoint iteration re-computes every $A_{l_i}$ in every step

$\implies$ redundant if $A_{l'_i}$ at no $F$-predecessor $l'$ changed

$\implies$ optimisation by worklist over control-flow edges

Algorithm 5.1 (Worklist algorithm)

Input: dataflow system $S = (\text{Lab, } E, F, (D, \sqsubseteq), \iota, \varphi)$

Variables: $W \in (\text{Lab} \times \text{Lab})^*$, $\{A_{l_i} \in D \mid l_i \in \text{Lab}\}$

Procedure:

$W := \varepsilon$; for $(l, l') \in F$ do $W := W \cdot (l, l')$; % Initialise $W$

for $l \in \text{Lab}$ do

if $l \in E$ then $A_{l_i} := \iota$ else $A_{l_i} := \perp_D$; % Initialise $A_l$
Efficient Fixpoint Computation

A Worklist Algorithm I

**Observation:** fixpoint iteration re-computes every $A_l$ in every step

$\implies$ redundant if $A_{l'}$ at no $F$-predecessor $l'$ changed

$\implies$ optimisation by worklist over control-flow edges

**Algorithm 5.1 (Worklist algorithm)**

Input: *dataflow system* $S = (Lab, E, F, (D, \sqsubseteq), \iota, \varphi)$

Variables: $W \in (Lab \times Lab)^*$, \{ $A_l \in D \mid l \in Lab$ \}

Procedure:

\[
W := \varepsilon; \text{ for } (l, l') \in F \text{ do } W := W \cdot (l, l'); \\
\text{ for } l \in Lab \text{ do} \\
\text{ if } l \in E \text{ then } A_l := \iota \text{ else } A_l := \bot_D; \\
\text{ while } W \neq \varepsilon\text{ do} \\
(\text{head}(W); W := \text{tail}(W); \\
\text{ if } \varphi_l(A_l) \not\sqsubseteq A_{l'} \text{ then} \\
A_{l'} := A_{l'} \sqcup \varphi_l(A_l); \\
\text{ for } (l', l'') \in F \text{ do} \\
\text{ if } (l', l'') \text{ not in } W \text{ then } W := (l', l'') \cdot W; \\
\]
Efficient Fixpoint Computation

A Worklist Algorithm I

Observation: fixpoint iteration re-computes every $A_l$ in every step

$\Rightarrow$ redundant if $A_{l'}$ at no $F$-predecessor $l'$ changed

$\Rightarrow$ optimisation by worklist over control-flow edges

Algorithm 5.1 (Worklist algorithm)

Input: dataflow system $S = (\text{Lab}, E, F, (D, \sqsubseteq), \iota, \varphi)$

Variables: $W \in (\text{Lab} \times \text{Lab})^*, \{A_l \in D \mid l \in \text{Lab}\}$

Procedure: $W := \varepsilon$; for $(l, l') \in F$ do $W := W \cdot (l, l')$; % Initialise $W$

for $l \in \text{Lab}$ do

if $l \in E$ then $A_l := \iota$ else $A_l := \bot_D$; % Initialise $A_l$

while $W \neq \varepsilon$ do

$(l, l') := \text{head}(W)$; $W := \text{tail}(W)$; % Next control-flow edge

if $\varphi_l(A_l) \nsubseteq A_{l'}$ then

$A_{l'} := A_{l'} \sqcup \varphi_l(A_l)$; % Update analysis information

for $(l', l'') \in F$ do

if $(l', l'')$ not in $W$ then $W := (l', l'') \cdot W$; % Propagate modification

Output: $\{A_l \mid l \in \text{Lab}\}$
### A Worklist Algorithm II

**Example 5.2 (Worklist algorithm)**

Available Expression analysis for

\[
\begin{align*}
c &= [x := a+b]^1; \\
   &\quad [y := a*b]^2; \\
   &\quad \text{while } [y > a+b]^3 \text{ do} \\
   &\quad \quad [a := a+1]^4; \\
   &\quad \quad [x := a+b]^5 \\
   &\quad \text{end}
\end{align*}
\]

(cf. Examples 2.9 and 4.11)

Transfer functions:

\[
\begin{align*}
\varphi_1(A) &= A \cup \{a+b\} \\
\varphi_2(A) &= A \cup \{a*b\} \\
\varphi_3(A) &= A \cup \{a+b\} \\
\varphi_4(A) &= A \setminus \{a+b, a*b, a+1\} \\
\varphi_5(A) &= A \cup \{a+b\}
\end{align*}
\]

Computation protocol: on the board
Efficient Fixpoint Computation

An “Optimisation”

**Conjecture:** it suffices to initialise worklist with *edges leaving extremal labels* (such that analysis information will propagate through CFG)
Efficient Fixpoint Computation

An “Optimisation”

**Conjecture:** it suffices to initialise worklist with edges leaving extremal labels (such that analysis information will propagate through CFG)

But ...

---

**Example 5.3 (Counterexample)**

Live Variables analysis for:

\[ c = [x := 0] \]
\[ [x := x + 1] \]
\[ [x := 2] \]

Solution: \( LV_1 = \{x\} \), \( LV_2 = \emptyset \), \( LV_3 = \{x\} \)
Efficient Fixpoint Computation

An “Optimisation”

**Conjecture:** it suffices to initialise worklist with **edges leaving extremal labels** (such that analysis information will propagate through CFG)

But ...

---

**Example 5.3 (Counterexample)**

Live Variables analysis for \( c = [x := 0]^{1}; [x := x + 1]^{2}; [x := 2]^{3} \)

Solution: \( LV_1 = \{x\}, LV_2 = \emptyset, LV_3 = \{x\} \)

“Optimised” worklist algorithm: 

\[
\begin{array}{cc|ccc}
W & LV_1 & LV_2 & LV_3 \\
(3, 2) & \emptyset & \emptyset & \{x\} \\
\end{array}
\]

⇒ wrong result!
Efficient Fixpoint Computation

An “Optimisation”

Conjecture: it suffices to initialise worklist with edges leaving extremal labels (such that analysis information will propagate through CFG)

But ...

Example 5.3 (Counterexample)

Live Variables analysis for $c = [x := 0]^1; [x := x + 1]^2; [x := 2]^3$

Solution: $LV_1 = \{x\}, LV_2 = \emptyset, LV_3 = \{x\}$

“Optimised” worklist algorithm:

<table>
<thead>
<tr>
<th>$W$</th>
<th>$LV_1$</th>
<th>$LV_2$</th>
<th>$LV_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(3, 2)$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>${x}$</td>
</tr>
<tr>
<td>$\varepsilon$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>${x}$</td>
</tr>
</tbody>
</table>

⇒ wrong result!
Correctness of Worklist Algorithm

Theorem 5.4 (Correctness of worklist algorithm)

Given a dataflow system $S = (\text{Lab}, E, F, (D, \sqsubseteq), \iota, \varphi)$, Algorithm 5.1 always terminates and computes $\text{fix}(\Phi_S)$. 

Proof. see [Nielson/Nielson/Hankin 2005, p. 75 ff]
Correctness of Worklist Algorithm

Theorem 5.4 (Correctness of worklist algorithm)

Given a dataflow system $S = (\text{Lab}, E, F, (D, \sqsubseteq), \iota, \phi)$, Algorithm 5.1 always terminates and computes $\text{fix}(\Phi_S)$.

Proof.

see [Nielson/Nielson/Hankin 2005, p. 75 ff]
The MOP Solution

Outline of Lecture 5

Recap: The Fixpoint Approach

Efficient Fixpoint Computation

The MOP Solution

Another Analysis: Constant Propagation
The MOP Solution

The MOP Solution I

- Other solution method for dataflow systems
The MOP Solution

The MOP Solution I

- Other solution method for dataflow systems
- MOP = Meet Over all Paths
The MOP Solution

The MOP Solution I

- Other solution method for dataflow systems
- MOP = Meet Over all Paths
- Analysis information for block $B^l$
  - least upper bound over all paths leading to $l$
  - most precise information for $l$ (“reference solution”)
The MOP Solution

The MOP Solution I

- Other solution method for dataflow systems
- MOP = Meet Over all Paths
- Analysis information for block $B'$
  - least upper bound over all paths leading to $l$
  - most precise information for $l$ (“reference solution”)

Definition 5.5 (Paths)

Let $S = (\text{Lab}, E, F, (D, \sqsubseteq), \iota, \varphi)$ be a dataflow system. For every $l \in \text{Lab}$, the set of paths up to $l$ is given by

$$\text{Path}(l) := \{[l_1, \ldots, l_{k-1}] \mid k \geq 1, l_1 \in E, (l_i, l_{i+1}) \in F \text{ for every } 1 \leq i < k, l_k = l\}.$$
The MOP Solution

The MOP Solution I

- Other solution method for dataflow systems
- MOP = Meet Over all Paths
- Analysis information for block \( B^l \)
  - least upper bound over all paths leading to \( l \)
  - most precise information for \( l \) ("reference solution")

**Definition 5.5 (Paths)**

Let \( S = (\text{Lab}, E, F, (D, \sqsubseteq), \iota, \varphi) \) be a dataflow system. For every \( l \in \text{Lab} \), the set of paths up to \( l \) is given by

\[
\text{Path}(l) := \{[l_1, \ldots, l_{k-1}] \mid k \geq 1, l_1 \in E, (l_i, l_{i+1}) \in F \text{ for every } 1 \leq i < k, l_k = l\}.
\]

For a path \( \pi = [l_1, \ldots, l_{k-1}] \in \text{Path}(l) \), we define the transfer function \( \varphi_{\pi} : D \to D \) by

\[
\varphi_{\pi} := \varphi_{l_{k-1}} \circ \ldots \circ \varphi_{l_1} \circ \text{id}_D
\]

(so that \( \varphi_{[]} = \text{id}_D \)).
The MOP Solution

Definition 5.6 (MOP solution)

Let \( S = (\text{Lab}, E, F, (D, \sqsubseteq), \iota, \varphi) \) be a dataflow system where \( \text{Lab} = \{l_1, \ldots, l_n\} \).

The \textbf{MOP solution} for \( S \) is determined by

\[
mop(S) := (mop(l_1), \ldots, mop(l_n)) \in D^n
\]

where, for every \( l \in \text{Lab} \),

\[
mop(l) := \bigsqcup \{\varphi_\pi(l) \mid \pi \in \text{Path}(l)\}.
\]

Remark:

\( \bullet \) \( \text{Path}(l) \) is generally infinite \( \Rightarrow \) not clear how to compute \( \text{mop}(l) \)

\( \bullet \) In fact: \textbf{MOP solution} generally undecidable (later)
The MOP Solution

The MOP Solution II

Definition 5.6 (MOP solution)

Let $S = (\text{Lab}, E, F, (D, \sqsubseteq), \iota, \varphi)$ be a dataflow system where $\text{Lab} = \{l_1, \ldots, l_n\}$. The MOP solution for $S$ is determined by

$$\text{mop}(S) := (\text{mop}(l_1), \ldots, \text{mop}(l_n)) \in D^n$$

where, for every $l \in \text{Lab}$,

$$\text{mop}(l) := \bigsqcup \{ \varphi_{\pi}(l) \mid \pi \in \text{Path}(l) \}.$$ 

Remark:

- $\text{Path}(l)$ is generally infinite
- $\Rightarrow$ not clear how to compute $\text{mop}(l)$
The MOP Solution

The MOP Solution II

Definition 5.6 (MOP solution)

Let $S = (Lab, E, F, (D, \sqsubseteq), \iota, \varphi)$ be a dataflow system where $Lab = \{l_1, \ldots, l_n\}$. The MOP solution for $S$ is determined by

$$mop(S) := (mop(l_1), \ldots, mop(l_n)) \in D^n$$

where, for every $l \in Lab$,

$$mop(l) := \bigsqcup \{\varphi_{\pi}(\iota) \mid \pi \in Path(l)\}.$$  

Remark:

- $Path(l)$ is generally infinite

$\Rightarrow$ not clear how to compute $mop(l)$

- In fact: MOP solution generally undecidable (later)
Example 5.7 (Live Variables; cf. Examples 2.12 and 4.12)

\[
c = \begin{align*}
&[x := 2]^1; \\
&[y := 4]^2; \\
&[x := 1]^3; \\
&\text{if } [y > 0]^4 \text{ then} \\
&\quad [z := x]^5 \\
&\text{else} \\
&\quad [z := y*y]^6; \\
&[x := z]^7
\end{align*}
\]
Example 5.7 (Live Variables; cf. Examples 2.12 and 4.12)

\[
c = [x := 2]^1; \\
y := 4]^2; \\
x := 1]^3; \\
\text{if } [y > 0]^4 \text{ then} \\
\quad [z := x]^5 \\
\text{else} \\
\quad [z := y*y]^6; \\
x := z]^7
\]

\[\implies \text{Path}(1) = \{[7, 5, 4, 3, 2], \\
\quad [7, 6, 4, 3, 2]\}\]
### Example 5.7 (Live Variables; cf. Examples 2.12 and 4.12)

\( c = [x := 2] ; \\
    [y := 4] ; \\
    [x := 1] ; \\
    \text{if } [y > 0] \text{ then} \\
    \quad [z := x] ; \\
    \text{else} \\
    \quad [z := y * y] ; \\
    [x := z] ; \\
\)

\( \implies \text{mop}(1) = \varphi_{[7,5,4,3,2]}(l) \sqcup \varphi_{[7,6,4,3,2]}(l) \)

\( \implies \text{Path}(1) = \{ [7, 5, 4, 3, 2], [7, 6, 4, 3, 2] \} \)
Example 5.7 (Live Variables; cf. Examples 2.12 and 4.12)

\[
c = \begin{cases} x := 2 \\ y := 4 \\ x := 1 \\ \text{if } y > 0 \text{ then } \\ \quad z := x \\ \text{else} \\ \quad z := y \times y \\ \quad x := z \end{cases}
\]

\[\Rightarrow \text{Path}(1) = \{ [7, 5, 4, 3, 2], [7, 6, 4, 3, 2] \}\]

\[\Rightarrow \text{mop}(1) = \varphi_{[7,5,4,3,2]}(\iota) \sqcup \varphi_{[7,6,4,3,2]}(\iota)\]
\[= \varphi_2(\varphi_3(\varphi_4(\varphi_5(\varphi_7(\{x, y, z\})))))) \sqcup \varphi_2(\varphi_3(\varphi_4(\varphi_6(\varphi_7(\{x, y, z\}))))))\]
Example 5.7 (Live Variables; cf. Examples 2.12 and 4.12)

c = [x := 2]¹;  
[y := 4]²;  
[x := 1]³;  
if [y > 0]⁴ then  
  [z := x]⁵  
else  
  [z := y*y]⁶;  
  [x := z]⁷

⇒ Path(1) = {[7, 5, 4, 3, 2],  
              [7, 6, 4, 3, 2]}

mop(1) = \phi_{[7,5,4,3,2]}(l) \sqcup \phi_{[7,6,4,3,2]}(l)  
= \phi_2(\phi_3(\phi_4(\phi_5(\phi_7(\{x, y, z\})))))) \sqcup  
\phi_2(\phi_3(\phi_4(\phi_6(\phi_7(\{x, y, z\}))))))  
= \phi_2(\phi_3(\phi_4(\phi_5(\{y, z\})))) \sqcup  
\phi_2(\phi_3(\phi_4(\phi_6(\{y, z\}))))
Example 5.7 (Live Variables; cf. Examples 2.12 and 4.12)

\[
\begin{align*}
c &= \begin{cases}
[ x := 2 ]^1; \\
[y := 4]^2; \\
[x := 1]^3; \\
\text{if } [y > 0]^4 \text{ then} \\
[z := x]^5 \\
\text{else} \\
[z := y*y]^6; \\
[x := z]^7
\end{cases}
\Rightarrow \text{Path}(1) = \{ [7, 5, 4, 3, 2], \\
[7, 6, 4, 3, 2] \}
\end{align*}
\]

\[
\begin{align*}
\text{mop}(1) &= \varphi_{[7,5,4,3,2]}(\iota) \sqcup \varphi_{[7,6,4,3,2]}(\iota) \\
&= \varphi_2(\varphi_3(\varphi_4(\varphi_7(\varphi_{[x, y, z]})))) \\
&= \varphi_2(\varphi_3(\varphi_4(\varphi_6(\varphi_{[x, y, z]})))) \\
&= \varphi_2(\varphi_3(\varphi_4(\varphi_5(\{y, z\})))) \\
&= \varphi_2(\varphi_3(\varphi_4(\{x, y\}))) \\
&= \varphi_2(\varphi_3(\{y\})) \\
&= \varnothing \\
&= \varnothing
\end{align*}
\]
The MOP Solution

Example 5.7 (Live Variables; cf. Examples 2.12 and 4.12)

\[ c = [x := 2]; \]
\[ [y := 4]; \]
\[ [x := 1]; \]
\[ \text{if } [y > 0] \text{ then } \]
\[ [z := x]; \]
\[ \text{else } \]
\[ [z := y*y]; \]
\[ [x := z]; \]

\[ \implies \text{Path}(1) = \{[7, 5, 4, 3, 2], [7, 6, 4, 3, 2]\} \]

\[ \implies \text{mop}(1) = \varphi_{[7,5,4,3,2]}(\ell) \sqcup \varphi_{[7,6,4,3,2]}(\ell) \]
\[ = \varphi_2(\varphi_3(\varphi_4(\varphi_5(\varphi_7(\{x, y, z\})))))) \sqcup \]
\[ = \varphi_2(\varphi_3(\varphi_4(\varphi_6(\{x, y, z\})))))) \sqcup \]
\[ = \varphi_2(\varphi_3(\varphi_4(\{y\})))) \sqcup \varphi_2(\varphi_3(\{y\}))) \]
\[ = \varphi_2(\varphi_3(\{x, y\}))) \sqcup \varphi_2(\varphi_3(\{y\}))) \]
\[ = \varphi_2(\varphi_3(\{y\}))) \sqcup \varphi_2(\varphi_3(\{y\}))) \]
\[ = \varphi_2(\varphi_3(\{y\}))) \]
\[ = \emptyset \sqcup \emptyset = \emptyset \]

(same as fix(\Phi_S[1]) – Ex. 4.12)
Example 5.7 (Live Variables; cf. Examples 2.12 and 4.12)

\[ c = [x := 2]^1; \]
\[ [y := 4]^2; \]
\[ [x := 1]^3; \]
\[ \text{if } [y > 0]^4 \text{ then} \]
\[ [z := x]^5 \]
\[ \text{else} \]
\[ [z := y*y]^6; \]
\[ [x := z]^7 \]

\[ \Rightarrow \text{Path}(1) = \{[7, 5, 4, 3, 2], [7, 6, 4, 3, 2]\} \]

\[ \Rightarrow \text{mop}(1) = \varphi[7,5,4,3,2](l) \sqcup \varphi[7,6,4,3,2](l) \]
\[ = \varphi_2(\varphi_3(\varphi_4(\varphi_5(\varphi_7(\{x, y, z\})))))) \sqcup \]
\[ \varphi_2(\varphi_3(\varphi_4(\varphi_6(\varphi_7(\{x, y, z\})))))) \]
\[ = \varphi_2(\varphi_3(\varphi_4(\varphi_5(\{y, z\})))) ) \sqcup \]
\[ \varphi_2(\varphi_3(\varphi_4(\{y\}))) \]
\[ = \varphi_2(\{y\}) \sqcup \varphi_2(\{y\}) \]
\[ = \varphi_2(\{y\}) \sqcup \varphi_2(\{y\}) \]
\[ = \emptyset \sqcup \emptyset \]
\[ = \emptyset \]
Example 5.7 (Live Variables; cf. Examples 2.12 and 4.12)

\[c = [x := 2]_1; \]
\[[y := 4]_2; \]
\[[x := 1]_3; \]
\[\text{if } [y > 0]_4 \text{ then} \]
\[[z := x]_5 \]
\[\text{else} \]
\[[z := y*y]_6; \]
\[[x := z]_7 \]

\[\implies \text{Path}(1) = \{[7, 5, 4, 3, 2], [7, 6, 4, 3, 2]\} \]

\[\implies \text{mop}(1) = \varphi_{[7,5,4,3,2]}(l) \sqcup \varphi_{[7,6,4,3,2]}(l) \]
\[= \varphi_2(\varphi_3(\varphi_4(\varphi_5(\varphi_7(\{x, y, z\})))))) \sqcup \]
\[\varphi_2(\varphi_3(\varphi_4(\varphi_6(\{x, y, z\})))))))) \]
\[= \varphi_2(\varphi_3(\varphi_4(\{y, z\})))) \sqcup \]
\[\varphi_2(\varphi_3(\varphi_4(\{y\})))) \]
\[= \varphi_2(\varphi_3(\{x, y\}))) \sqcup \varphi_2(\varphi_3(\{y\})) \]
\[= \varphi_2(\{y\}) \sqcup \varphi_2(\{y\}) \]
\[= \emptyset \sqcup \emptyset \]
Example 5.7 (Live Variables; cf. Examples 2.12 and 4.12)

c = \[
x := 2
\]
\[
y := 4
\]
\[
x := 1
\]
if \[y > 0\] then
\[
z := x
\]
else
\[
z := y*y
\]
x := z
\[
\implies \text{Path}(1) = \{[7, 5, 4, 3, 2], [7, 6, 4, 3, 2]\}
\]

\[
mop(1) = \varphi[7,5,4,3,2](\iota) \sqcup \varphi[7,6,4,3,2](\iota)
\]
\[
= \varphi_2(\varphi_3(\varphi_4(\varphi_5(\varphi_7(\{x, y, z\})]))) \sqcup
\]
\[
\varphi_2(\varphi_3(\varphi_4(\varphi_6(\varphi_7(\{x, y, z\})))
\]
\[
= \varphi_2(\varphi_3(\varphi_4(\varphi_5(\{y, z\})))) \sqcup
\]
\[
\varphi_2(\varphi_3(\varphi_4(\{y\}))
\]
\[
= \varphi_2(\varphi_3(\{x, y\})) \sqcup \varphi_2(\varphi_3(\{y\}))
\]
\[
= \varphi_2(\{y\}) \sqcup \varphi_2(\varphi_3(\{y\}))
\]
\[
= \varnothing \sqcup \varnothing
\]
\[
= \varnothing \quad (\text{same as fix}(\Phi_S)[1] – \text{Ex. 4.12})
Another Analysis: Constant Propagation

Outline of Lecture 5

Recap: The Fixpoint Approach

Efficient Fixpoint Computation

The MOP Solution

Another Analysis: Constant Propagation
Another Analysis: Constant Propagation

Goal of Constant Propagation Analysis

The goal of **Constant Propagation Analysis** is to determine, for each program point, whether a variable has a constant value whenever execution reaches that point.

Example 5.8 (Constant Propagation Analysis)

\[
\begin{align*}
x &:= 1 \\
y &:= 1 \\
z &:= 1 \\
\text{while } z > 0 \\
w &:= x + y \\
\text{if } w = 2 \\
\text{then } x &:= y + 2 \\
\text{end}
\end{align*}
\]

- \( y = 1 \) at labels 4–7
- \( w, x \) not constant at labels 4–7
- possible optimisations:
  \[
  \begin{align*}
  w &:= x + 1 \\
x &:= 3
  \end{align*}
  \]
Goal of Constant Propagation Analysis

The goal of Constant Propagation Analysis is to determine, for each program point, whether a variable has a constant value whenever execution reaches that point.

Used for Constant Folding: replace reference to variable by constant value and evaluate constant expressions.
Another Analysis: Constant Propagation

Goal of Constant Propagation Analysis

Constant Propagation Analysis

The goal of Constant Propagation Analysis is to determine, for each program point, whether a variable has a constant value whenever execution reaches that point.

Used for Constant Folding: replace reference to variable by constant value and evaluate constant expressions

Example 5.8 (Constant Propagation Analysis)

```
x := 1;[y := 1];[z := 1];
while [z > 0] do
  [w := x+y];
  if [w = 2] then
    [x := y+2]
  end
end
```
Another Analysis: Constant Propagation

Goal of Constant Propagation Analysis

The goal of **Constant Propagation Analysis** is to determine, for each program point, whether a variable has a constant value whenever execution reaches that point.

Used for **Constant Folding**: replace reference to variable by constant value and evaluate constant expressions

**Example 5.8 (Constant Propagation Analysis)**

\[
x := 1^1; \quad y := 1^2; \quad z := 1^3;
\]

\[
\text{while } [z > 0]^4 \text{ do}
\]

\[
w := x+y^5;
\]

\[
\text{if } [w = 2]^6 \text{ then}
\]

\[
x := y+2^7
\]

\]

\* y = z = 1 at labels 4–7\*
Another Analysis: Constant Propagation

Goal of Constant Propagation Analysis

The goal of **Constant Propagation Analysis** is to determine, for each program point, whether a variable has a constant value whenever execution reaches that point.

Used for **Constant Folding**: replace reference to variable by constant value and evaluate constant expressions

**Example 5.8 (Constant Propagation Analysis)**

\[
\begin{align*}
x & := 1 \\
y & := 1 \\
z & := 1 \\
\text{while } [z > 0] & \text{ do} \\
&w & := x+y \\
& \text{if } [w = 2] \text{ then} \\
& [x := y+2] \\
\text{end} \\
\text{end}
\end{align*}
\]

- \( y = z = 1 \) at labels 4–7
- \( w, x \) not constant at labels 4–7
Another Analysis: Constant Propagation

Goal of Constant Propagation Analysis

The goal of **Constant Propagation Analysis** is to determine, for each program point, whether a variable has a constant value whenever execution reaches that point.

Used for **Constant Folding**: replace reference to variable by constant value and evaluate constant expressions

**Example 5.8 (Constant Propagation Analysis)**

\[
\begin{align*}
\text{while } [z > 0] &; [w := x+y] \\
\text{if } [w = 2] &; [x := y+2] \\
\end{align*}
\]

- **y = z = 1** at labels 4–7
- **w, x** not constant at labels 4–7
- **possible optimisations:**
  - \([\text{true}]\)
  - \([w := x+1]\)
  - \([x := 3]\)
Another Analysis: Constant Propagation

Formalising Constant Propagation Analysis I

The dataflow system \( S = (Lab, E, F, (D, \sqsubseteq), \iota, \varphi) \) is given by

- set of labels \( Lab := Lab_c \),
- extremal labels \( E := \{\text{init}(c)\} \) (forward problem)
- flow relation \( F := \text{flow}(c) \) (forward problem)
Another Analysis: Constant Propagation

Formalising Constant Propagation Analysis I

The dataflow system $S = (\text{Lab}, E, F, (D, \sqsubseteq), \iota, \varphi)$ is given by

- set of labels $\text{Lab} := \text{Lab}_c$,
- extremal labels $E := \{\text{init}(c)\}$ (forward problem)
- flow relation $F := \text{flow}(c)$ (forward problem)
- complete lattice $(D, \sqsubseteq)$ where
  - $D := \{\delta \mid \delta : \Var_c \to \mathbb{Z} \cup \{\bot, \top\}\}$
    - $\delta(x) = z \in \mathbb{Z}$: $x$ has constant value $z$ (i.e., possible values in $\{z\}$)
    - $\delta(x) = \bot$: $x$ undefined (i.e., possible values in $\emptyset$)
    - $\delta(x) = \top$: $x$ overdefined (i.e., possible values in $\mathbb{Z}$)
  - $\sqsubseteq \subseteq D \times D$ defined by pointwise extension of $\bot \sqsubseteq z \sqsubseteq \top$ (for every $z \in \mathbb{Z}$)
Another Analysis: Constant Propagation

Formalising Constant Propagation Analysis I

The dataflow system \( S = (\text{Lab}, E, F, (D, \sqsubseteq), \iota, \varphi) \) is given by

- set of labels \( \text{Lab} := \text{Lab}_c \),
- extremal labels \( E := \{\text{init}(c)\} \) (forward problem)
- flow relation \( F := \text{flow}(c) \) (forward problem)
- complete lattice \( (D, \sqsubseteq) \) where
  - \( D := \{\delta \mid \delta : \text{Var}_c \to \mathbb{Z} \cup \{\bot, \top\}\} \)
    - \( \delta(x) = z \in \mathbb{Z} \): \( x \) has constant value \( z \) (i.e., possible values in \{\( z \)\})
    - \( \delta(x) = \bot \): \( x \) undefined (i.e., possible values in \( \emptyset \))
    - \( \delta(x) = \top \): \( x \) overdefined (i.e., possible values in \( \mathbb{Z} \))
  - \( \sqsubseteq \subseteq D \times D \) defined by pointwise extension of \( \bot \sqsubseteq z \sqsubseteq \top \) (for every \( z \in \mathbb{Z} \))

Example 5.9

\( \text{Var}_c = \{w, x, y, z\} \), \( \delta_1 = (\bot, 1, 2, \top) \), \( \delta_2 = (3, 1, 4, \top) \)

\[ \Rightarrow \delta_1 \sqcup \delta_2 = (3, 1, \top, \top) \]
Another Analysis: Constant Propagation

Formalising Constant Propagation Analysis II

Dataflow system $S = (\text{Lab}, E, F, (D, \sqsubseteq), \iota, \varphi)$ (continued):

- extremal value $\iota := \delta_\top \in D$ where $\delta_\top(x) := \top$ for every $x \in \text{Var}_c$
  (i.e., every $x$ has (unknown) default value)
Another Analysis: Constant Propagation

Formalising Constant Propagation Analysis II

Dataflow system $S = (\text{Lab}, E, F, (D, \sqsubseteq), \iota, \varphi)$ (continued):

- extremal value $\iota := \delta_{\top} \in D$ where $\delta_{\top}(x) := \top$ for every $x \in \text{Var}_c$ (i.e., every $x$ has (unknown) default value)
- transfer functions $\{\varphi_l \mid l \in \text{Lab}\}$ defined by

\[
\varphi_l(\delta) := \begin{cases} 
\delta & \text{if } B^l = \text{skip} \text{ or } B^l \in \text{BExp} \\
\delta[x \mapsto \text{val}_\delta(a)] & \text{if } B^l = (x := a)
\end{cases}
\]

where

\[
\text{val}_\delta(x) := \delta(x) \\
\text{val}_\delta(z) := z \\
\text{val}_\delta(a_1 \text{ op } a_2) := \begin{cases} 
z_1 \text{ op } z_2 & \text{if } z_1, z_2 \in \mathbb{Z} \\
\bot & \text{if } z_1 = \bot \text{ or } z_2 = \bot \\
\top & \text{otherwise}
\end{cases}
\]

for $z_1 := \text{val}_\delta(a_1)$ and $z_2 := \text{val}_\delta(a_2)$
Another Analysis: Constant Propagation

Formalising Constant Propagation Analysis III

Example 5.10

If \( \delta = (\perp_w, 1_x, 2_y, \top_z) \), then

\[
\varphi_l(\delta) = \begin{cases} 
(0_w, 1_x, 2_y, \top_z) & \text{if } B_l = (w := 0) \\
(3_w, 1_x, 2_y, \top_z) & \text{if } B_l = (w := y+1) \\
(\perp_w, 1_x, 2_y, \top_z) & \text{if } B_l = (w := w+x) \\
(\top_w, 1_x, 2_y, \top_z) & \text{if } B_l = (w := z+2)
\end{cases}
\]