Theorem (Undecidability of MOP solution): The MOP solution for Constant Propagation is undecidable.

Proof.

Based on undecidability of Modified Post Correspondence Problem (MPCP): Let $\Gamma$ be some alphabet, $n \in \mathbb{N}$, and $u_1, \ldots, u_n, v_1, \ldots, v_n \in \Gamma^+$. Do there exist $i_1, \ldots, i_m \in \{1, \ldots, n\}$ with $m \geq 1$ and $i_1 = 1$ such that $u_{i_1} u_{i_2} \ldots u_{i_m} = v_{i_1} v_{i_2} \ldots v_{i_m}$?

Given a MPCP, we construct a WHILE program (with strings and Booleans) whose MOP analysis detects a constant property iff the MPCP has no solution (see next slide).
Proof (continued).

\[ x := u_1; y := v_1; \]
\[ \text{while ... do} \]
\[ \quad \text{if ... then} \]
\[ \quad \quad x := x ++ u_1; \]
\[ \quad \quad y := y ++ v_1 \]
\[ \quad \text{else if ... then} \]
\[ \quad \quad : \]
\[ \quad \text{else} \]
\[ \quad \quad x := x ++ u_n; \]
\[ \quad \quad y := y ++ v_n \]
\[ \text{end ... end} \]
\[ \text{if } x = y \text{ then } z := 1 \text{ else } z := 0 \text{ end; } \]
\[ z := (x = y); \]
\[ \text{[skip']} \]

Then: \( \text{mop}(l)(z) = \text{false} \)
\[ \iff \ x \neq y \text{ at the end of every path up to } l \]
\[ \iff \text{the MPCP has no solution} \]
Recap: Interval Analysis

The Complete Lattice of Interval Analysis

\[-\infty, +\infty\]
Recap: Interval Analysis

Formalising Interval 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
  - $D := \{\delta \mid \delta : \text{Var}_c \rightarrow \text{Int}\}$
  - $\delta_1 \sqsubseteq \delta_2$ iff $\delta_1(x) \subseteq \delta_2(x)$ for every $x \in \text{Var}_c$
- $\iota := \top_D : \text{Var}_c \rightarrow \text{Int} : x \mapsto \top_{\text{Int}}$ (with $\top_{\text{Int}} = [-\infty, +\infty]$)
- $\varphi$: see next slide
Recap: Interval Analysis

Formalising Interval Analysis II

Transfer functions $\{\varphi_l \mid l \in \text{Lab}\}$ are defined by

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

where

$$\text{val}_\delta(x) := \delta(x) \quad \text{val}_\delta(a_1 + a_2) := \text{val}_\delta(a_1) \oplus \text{val}_\delta(a_2)$$

$$\text{val}_\delta(z) := [z, z] \quad \text{val}_\delta(a_1 - a_2) := \text{val}_\delta(a_1) \ominus \text{val}_\delta(a_2)$$

$$\text{val}_\delta(a_1 \cdot a_2) := \text{val}_\delta(a_1) \odot \text{val}_\delta(a_2)$$

with

$$\emptyset \oplus J := J \ominus \emptyset := \emptyset \ominus J := \ldots := \emptyset$$

$$[y_1, y_2] \ominus [z_1, z_2] := [y_1 - z_2, y_2 - z_1]$$

$$[y_1, y_2] \odot [z_1, z_2] := \begin{cases} 
[y_1 z_1, y_1 z_2, y_2 z_1, y_2 z_2], & \text{if } y_1 \neq 0 \\
\{y_1 z_1, y_1 z_2, y_2 z_1, y_2 z_2\}, & \text{otherwise}
\end{cases}$$
Recap: Interval Analysis

Widening Operators

Definition (Widening operator)

Let \((D, \sqsubseteq)\) be a complete lattice. A mapping \(\nabla: D \times D \rightarrow D\) is called **widening operator** if

- for every \(d_1, d_2 \in D\),

\[ d_1 \sqcup d_2 \sqsubseteq d_1 \nabla d_2 \]

and

- for all ascending chains \(d_0 \sqsubseteq d_1 \sqsubseteq \ldots\), the ascending chain \(d_0^{\nabla} \sqsubseteq d_1^{\nabla} \sqsubseteq \ldots\) eventually stabilises where

\[ d_0^{\nabla} := d_0 \text{ and } d_{i+1}^{\nabla} := d_i^{\nabla} \nabla d_{i+1} \text{ for each } i \in \mathbb{N} \]

Remarks:

- \((d_i^{\nabla})_{i \in \mathbb{N}}\) is clearly an **ascending chain** as \(d_{i+1}^{\nabla} = d_i^{\nabla} \nabla d_{i+1} \sqsubseteq d_i^{\nabla} \sqcup d_{i+1} \sqsupseteq d_i^{\nabla}\)

- In contrast to \(\sqcup\), \(\nabla\) does **not** have to be commutative, associative, monotonic, nor absorptive \((d \nabla d = d)\)

- The requirement \(d_1 \sqcup d_2 \sqsubseteq d_1 \nabla d_2\) guarantees **soundness** of widening
Recap: Interval Analysis

Applying Widening to Interval Analysis

- A widening operator: \( \triangledown : \text{Int} \times \text{Int} \rightarrow \text{Int} \) with
  \[
  \emptyset \triangledown J := J, \quad J \triangledown \emptyset := J
  \]
  
  \[
  [x_1, x_2] \triangledown [y_1, y_2] := [z_1, z_2]
  \]
  where
  \[
  z_1 := \begin{cases} 
  x_1 & \text{if } x_1 \leq y_1 \\
  -\infty & \text{otherwise}
  \end{cases}
  \]
  \[
  z_2 := \begin{cases} 
  x_2 & \text{if } x_2 \geq y_2 \\
  +\infty & \text{otherwise}
  \end{cases}
  \]

- Widening turns infinite ascending chain
  \[
  J_0 = \emptyset \subseteq J_1 = [1, 1] \subseteq J_2 = [1, 2] \subseteq J_3 = [1, 3] \subseteq \ldots
  \]
  into a finite one:
  \[
  \begin{align*}
  J_0^{\triangledown} &= J_0 = \emptyset \\
  J_1^{\triangledown} &= J_0^{\triangledown} \triangledown J_1 = \emptyset \triangledown [1, 1] = [1, 1] \\
  J_2^{\triangledown} &= J_1^{\triangledown} \triangledown J_2 = [1, 1] \triangledown [1, 2] = [1, +\infty] \\
  J_3^{\triangledown} &= J_2^{\triangledown} \triangledown J_3 = [1, +\infty] \triangledown [1, 3] = [1, +\infty]
  \end{align*}
  \]

- In fact, the maximal chain size arising with this operator is 4:
  \[
  \emptyset \subseteq [3, 7] \subseteq [3, +\infty] \subseteq [-\infty, +\infty]
  \]
Recap: Interval Analysis

Worklist Algorithm with Widening

**Goal:** extend Algorithm 5.1 by widening to ensure termination

**Algorithm (Worklist algorithm with widening)**

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:

$W := \varepsilon$; \hspace{1em} for $(l, l') \in F$ do $W := W \cdot (l, l')$; \hspace{1em} % Initialise $W$

\hspace{2em} for $l \in \text{Lab}$ do

\hspace{4em} if $l \in E$ then $A_{l} := \iota$ else $A_{l} := \bot_D$; \hspace{1em} % Initialise $A_{l}$

\hspace{1.5em} while $W \neq \varepsilon$ do

\hspace{3em} $(l, l') := \text{head}(W)$; $W := \text{tail}(W)$; \hspace{1em} % Next control-flow edge

\hspace{3em} if $\varphi_{l}(A_{l}) \not\sqsubseteq A_{l'}$ then \hspace{1em} % Fixpoint not yet reached

\hspace{4em} $A_{l'} := A_{l'} \sqcup \varphi_{l}(A_{l})$;

\hspace{3em} for $(l'', l''') \in F$ do

\hspace{4em} if $(l'', l''')$ not in $W$ then $W := (l', l''') \cdot W$; \hspace{1em} % Propagate modification

Output: $\{A_{l} \mid l \in \text{Lab}\}$, denoted by $\text{fix}^\nabla(\Phi_S)$

**Remark:** due to widening, only $\text{fix}^\nabla(\Phi_S) \sqsubseteq \text{fix}(\Phi_S)$ is guaranteed (cf. Thm. 5.4)
Narrowing

Another Widening Example

Example 8.1

Transfer functions (for $\delta = (J_x, J_y)$):

$\varphi_1(J_x, J_y) = ([1, 1], J_y)$

$\varphi_2(J_x, J_y) = (J_x, [2, 2])$

$\varphi_3(J_x, J_y) = (J_x, J_y)$

$\varphi_4(J_x, J_y) = ([3, 3], J_y)$

$\varphi_5(J_x, \emptyset) = (J_x, \emptyset)$

$\varphi_5(J_x, [y_1, y_2]) = (J_x, [y_1 + 1, y_2 + 1])$

Application of worklist algorithm

1. without widening (omitted): diverges (for $y$) with expected result for $x$: $\text{AI}_3(x) = [1, 3]$

2. with widening (on the board): terminates with unexpected result for $x$: $\text{AI}_3(x) = [1, +\infty]$
Narrowing

Idea of Narrowing

- **Observation:** widening can “shoot above the target”, i.e., lead to unnecessarily imprecise results
- **Solution:** improvement by iterating again from the result obtained by widening (i.e., from $\text{fix}^\nabla (\Phi_S)$)

$$\implies \text{compute } \Phi_S^k(\text{fix}^\nabla (\Phi_S)) \text{ for } k = 1, 2, \ldots$$

- **Soundness:** $\text{fix}^\nabla (\Phi_S) \sqsubseteq \text{fix}(\Phi_S)$ (cf. Alg. 7.5)

$$\implies \Phi_S^k(\text{fix}^\nabla (\Phi_S)) \sqsubseteq \Phi_S^k(\text{fix}(\Phi_S)) = \text{fix}(\Phi_S)$$

(since $\Phi_S$ and thus $\Phi_S^k$ monotonic)
Narrowing Example

Example 8.2 (cf. Example 8.1)

Transfer functions (for $\delta = (J_x, J_y)$):

$\varphi_1(J_x, J_y) = ([1, 1], J_y)$
$\varphi_2(J_x, J_y) = ([1, 1], J_y)$
$\varphi_3(J_x, J_y) = (J_x, [2, 2])$
$\varphi_4(J_x, J_y) = ([3, 3], J_y)$
$\varphi_5(J_x, [y_1, y_2]) = (J_x, [y_1 + 1, y_2 + 1])$

<table>
<thead>
<tr>
<th>Narrowing</th>
<th>$\text{Al}_1$</th>
<th>$\text{Al}_2$</th>
<th>$\text{Al}_3$</th>
<th>$\text{Al}_4$</th>
<th>$\text{Al}_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{fix}^\uparrow(\Phi_S)$</td>
<td>$\langle T, T \rangle$</td>
<td>$([1, 1], T)$</td>
<td>$([1, +\infty], [2, +\infty])$</td>
<td>$([1, +\infty], [2, +\infty])$</td>
<td>$([3, 3], [2, +\infty])$</td>
</tr>
<tr>
<td>$\Phi_S(\text{fix}^\uparrow(\Phi_S))$</td>
<td>$\langle T, T \rangle$</td>
<td>$([1, 1], T)$</td>
<td>$([1, 3], [2, +\infty])$</td>
<td>$([1, +\infty], [2, +\infty])$</td>
<td>$([3, 3], [2, +\infty])$</td>
</tr>
<tr>
<td>$\Phi_2^3(\text{fix}^\uparrow(\Phi_S))$</td>
<td>$\langle T, T \rangle$</td>
<td>$([1, 1], T)$</td>
<td>$([1, 3], [2, +\infty])$</td>
<td>$([1, 3], [2, +\infty])$</td>
<td>$([3, 3], [2, +\infty])$</td>
</tr>
</tbody>
</table>
Narrowing

Narrowing in Practice

- **Problem:** narrowing may not terminate (due to infinite descending chains)
- **But:** possible to stop after each step without losing soundness ($\Phi_S^k(\text{fix} \nabla (\Phi_S)) \sqsupseteq \text{fix}(\Phi_S)$)
- **In practice:** termination often ensured by using narrowing operators
  ($\approx$ counterpart of widening operator; definition omitted)
Taking Conditional Branches into Account

Taking Conditional Branches into Account

- **So far:** values of conditions have been ignored in analysis
- Essentially: if and while statements treated as nondeterministic choice between the two branches

**Example 8.3**

```plaintext
y := 0;
z := 0;
while [x > 0] do
  if y < 17 then
    y := y + 1
  end;
z := z + x;
x := x - 1
end;
```

- Interval analysis (with widening) yields for /:
  - \( x \in [-\infty, +\infty] \)
  - \( y \in [0, +\infty] \)
  - \( z \in [-\infty, +\infty] \)

- Too pessimistic! In fact,
  - \( x \in [-\infty, +\infty] \)
  - \( y \in [0, 17] \)
  - \( z \in [0, +\infty] \)
Taking Conditional Branches into Account

Taking Conditional Branches into Account II

- **Solution:** introduce transfer functions for branches
- **First approach:** attach (negated) conditions as labels to control flow edges
  - advantage: no language modification required
  - disadvantage: entails extension of DFA framework
  - will not further be considered here
- **Second approach:** encode conditions as assertions (proper statements)
  - advantage: DFA framework can be reused
  - disadvantage: entails extension of WHILE language
  - the way we will follow
Taking Conditional Branches into Account

Conditions as Edge Labels vs. Conditions as Assertions

Example 8.4 (cf. Example 8.3)

\[
\begin{align*}
[y & := 0] \quad \text{(1)} \\
[z & := 0] \quad \text{(2)} \\
\textbf{while} [x > 0] \quad \text{(3)} \\
\text{if } [y < 17] & \quad \text{(4)} \\
[y & := y + 1] \quad \text{(5)} \\
[z & := z + x] \quad \text{(6)} \\
[x & := x - 1] \quad \text{(7)} \\
\neg (x > 0) \quad \text{(8)} \\
\neg (y < 17) \quad \text{(9)}
\end{align*}
\]

\[
y := 0; \\
z := 0; \\
\textbf{while } x > 0 \textbf{ do} \\
\quad \text{assert } x > 0; \\
\quad \text{if } y < 17 \textbf{ then} \\
\qquad \text{assert } y < 17; \\
\qquad y := y + 1 \\
\qquad \textbf{end}; \\
\quad z := z + x; \\
\quad x := x - 1 \\
\quad \textbf{end}; \\
\quad \text{assert } \neg (x > 0);
\]
Definition 8.5 (Labelled WHILE programs with assertions)

The syntax of labelled WHILE programs with assertions is defined by the following context-free grammar:

\[
\begin{align*}
a &::= z \mid x \mid a_1+a_2 \mid a_1-a_2 \mid a_1*a_2 \in AExp \\
b &::= t \mid a_1=a_2 \mid a_1>a_2 \mid \neg b \mid b_1 \land b_2 \mid b_1 \lor b_2 \in BExp \\
c &::= \{\text{skip}\} \mid \{x := a\} \mid c_1 ; c_2 \\
&\quad \mid \text{if } [b] \text{ then } c_1 \text{ else } c_2 \text{ end } \mid \text{while } [b] \text{ do } c \text{ end } \mid [\text{assert } b] \in \text{Cmd}
\end{align*}
\]

To be done:

- Definition of transfer functions for assert blocks (depending on analysis problem)
- Idea: assertions as filters that let only “compatible” information pass
Constant Propagation Analysis with Assertions

Original Constant Propagation Analysis

So far:

- complete lattice \((D, \sqsubseteq)\) where
  
  \[ \begin{align*}
  D &= \{ \delta \mid \delta : \text{Var}_c \to \mathbb{Z} \cup \{ \bot, \top \} \} \\
  \delta(x) &= z \in \mathbb{Z}: x \text{ has constant value } z \text{ (i.e., possible values in } \{z\}) \\
  \delta(x) &= \bot: x \text{ undefined (i.e., possible values in } \emptyset) \\
  \delta(x) &= \top: x \text{ overdefined (i.e., possible values in } \mathbb{Z})
  \end{align*} \]

- \(\sqsubseteq \subseteq D \times D\) defined by pointwise extension of \(\bot \subseteq z \subseteq \top\) (for every \(z \in \mathbb{Z}\))

- transfer functions \(\{ \varphi_l \mid l \in \text{Lab} \}\) defined by

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

where

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

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

Transfer Functions of Assertions I

Additionally for \( B^l = (\text{assert } b) \), \( \delta : \text{Var}_c \rightarrow \mathbb{Z} \cup \{\bot, \top\} \) and \( x \in \text{Var}_c \):

\[
\varphi_l(\delta)(x) := \begin{cases} 
\bot & \text{if } \not\exists \sigma \in \Sigma_\delta : \text{val}_\sigma(b) = \text{true} \\
z & \text{if } \forall \sigma \in \Sigma_\delta : \text{val}_\sigma(b) = \text{true} \implies \sigma(x) = z \\
\top & \text{otherwise}
\end{cases}
\]

where

- the set of \( \delta \)-states is given by

\[
\Sigma_\delta := \left\{ \sigma : \text{Var}_c \rightarrow \mathbb{Z} \mid \forall y \in \text{Var}_c : \sigma(y) \in \left\{ \begin{array}{ll}
\emptyset & \text{if } \delta(y) = \bot \\
\{z\} & \text{if } \delta(y) = z \\
\mathbb{Z} & \text{if } \delta(y) = \top
\end{array} \right. \right\}
\]

(and thus \( \Sigma_\delta = \emptyset \) iff \( \delta(y) = \bot \) for some \( y \in \text{Var}_c \))

- the evaluation function \( \text{val}_\sigma : \text{BExp} \rightarrow \mathbb{B} \) for \( \sigma : \text{Var}_c \rightarrow \mathbb{Z} \) is defined by

\[
\text{val}_\sigma(b_1 = b_2) := (\text{val}_\sigma(a_1) = \text{val}_\sigma(a_2))
\]

\[
(\text{val}_\sigma : \text{AExp} \rightarrow \mathbb{Z}\text{ on previous slide}) \quad \text{val}_\sigma(b_1 \land b_2) := \left\{ \begin{array}{ll}
\text{true} & \text{if } \text{val}_\sigma(b_1) = \text{false} \\
\text{false} & \text{otherwise}
\end{array} \right.
\]

\[
\text{val}_\sigma(\neg b) := \left\{ \begin{array}{ll}
\text{true} & \text{if } \text{val}_\sigma(b) = \text{false} \\
\text{false} & \text{otherwise}
\end{array} \right.
\]

\[
\text{val}_\sigma(b_1 = b_2) := \left\{ \begin{array}{ll}
\text{true} & \text{if } \text{val}_\sigma(b_1) = \text{val}_\sigma(b_2) = \text{true} \\
\text{false} & \text{otherwise}
\end{array} \right.
\]
Constant Propagation Analysis with Assertions

Transfer Functions of Assertions II

Example 8.6

1. \( \text{Var}_c = \{x, y, z\}, \delta = (\perp, 1, \top) \)
   
   \[ \Rightarrow \Sigma_{\delta} = \emptyset \Rightarrow \varphi_{\text{assert } b(\delta)} = (\perp, \perp, \perp) \text{ for every } b \in B_{\text{Exp}} \]

2. \( \text{Var}_c = \{x, y, z\}, \delta = (1, 2, \top) \)
   
   \[ \Rightarrow \Sigma_{\delta} = \{(1, 2, z) \mid z \in \mathbb{Z}\} \Rightarrow \varphi_{\text{assert } x=y} = (1, 1, \top) \]
   \[ \varphi_{\text{assert } y=z} = (1, 2, 2) \]
   \[ \varphi_{\text{assert } y<z} = (1, 2, \top) \]

3. \( \text{Var}_c = \{x, y, z\}, \delta = (1, 1, \top) \)
   
   \[ \Rightarrow \Sigma_{\delta} = \{(1, z_1, z_2) \mid z_1, z_2 \in \mathbb{Z}\} \Rightarrow \varphi_{\text{assert } x=y} = (1, 1, \top) \]
   \[ \varphi_{\text{assert } y=z} = (1, 1, \top) \]
Constant Propagation Analysis with Assertions

Transfer Functions of Assertions III

Remarks:

• For $B' = (\text{assert } b)$ and $\delta : \text{Var}_c \rightarrow \mathbb{Z} \cup \{\bot, \top\}$, $\varphi_l(\delta) \subseteq \delta$ and hence $\Sigma_{\varphi_l(\delta)} \subseteq \Sigma_\delta$ (“filter”)

• Constant propagation captures interdependencies between variables only when both are constant (cf. “assert y=z” in Example 8.6(3))

• $\varphi_l(\delta)$ can be determined (or at least approximated) by Satisfiability Modulo Theories (SMT) techniques

• If $\text{CP}_l(x) = \bot$ for some $l \in \text{Lab}_c$ and $x \in \text{Var}_c$, then $l$ is unreachable (and $\text{CP}_l(y) = \bot$ for all $y \in \text{Var}_c$)
Constant Propagation Analysis with Assertions

An Example

Example 8.7

\[
\text{if } [x = 1] \text{ then } \\
\text{assert } x = 1; \\
[y := x + 1] \\
\text{else } \\
\text{assert } \neg(x = 1); \\
[y := 2] \\
\text{end; } \\
\text{[skip]} \\
\]

<table>
<thead>
<tr>
<th>Without assertions</th>
<th>With assertions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( CP_1 = (\top, \top) )</td>
<td>( CP_1 = (\top, \top) )</td>
</tr>
<tr>
<td>( CP_3 = (\top, \top) )</td>
<td>( CP_3 = (1, \top) )</td>
</tr>
<tr>
<td>( CP_5 = (\top, \top) )</td>
<td>( CP_5 = (\top, \top) )</td>
</tr>
<tr>
<td>( CP_6 = (\top, \top) \sqcup (\top, 2) )</td>
<td>( CP_6 = (1, 2) \sqcup (\top, 2) )</td>
</tr>
</tbody>
</table>

Without assertions: \( (\top, \top) \)

With assertions: \( (\top, 2) \)