Static Program Analysis

Lecture 7: Dataflow Analysis VI (Undecidability of MOP Solution & Non-ACC Domains)

Winter Semester 2016/17

Thomas Noll
Software Modeling and Verification Group
RWTH Aachen University

https://moves.rwth-aachen.de/teaching/ws-1617/spa/

Software Modeling and Verification Chair

RWTH AACHEN UNIVERSITY

Software Architektur– Praxis-Workshop

itestra
Software Productivity

Bringen Sie Informatik zur Wirkung!

Serviceorientiert, zustandslos, geschichtet, lose gekoppelt, ad hoc entstanden oder model-driven… was ist die beste Softwarearchitektur?

Inhalte des Workshops:

- Anhand eines realen Beispiels aus der Praxis entwickelt ihr in kleinen Teams die Architektur für ein Informationssystem.
- Da sich Software fortlaufend weiterentwickelt, prüfen wir, wie sich die Architektur gegenüber neuen, auch unerwarteten Anforderungen verhält und daran anpassen lässt. Gemeinsam diskutieren wir den Einfluss verschiedener Anforderungen, mögliche Varianten und ihre Auswirkungen.
- Außerdem zeigen wir euch die Aufgaben eines Softwarearchitekten und wie er Projekte zum Erfolg führt.

Mittwoch, **14.12.2016**, 09.00-16.00 Uhr

Seminarraum 5053.2, B-IT Research School, RWTH Aachen (Ahornstraße 55)

Melden Sie sich unter Angabe Ihres Semesters bis zum **4.12.2016** an.
Wir freuen uns auf Sie!

Kontakt: Anne-Kristin Hauk (**hauk@itestra.de**) – www.itestra.com

Outline of Lecture 7

The MOP Solution

Definition (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

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

where, for every $l \in Lab$,

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

Remark:

- $Path(l)$ is generally infinite

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

- In fact: MOP solution generally undecidable (later)

MOP vs. Fixpoint Solution I

Example (Constant Propagation)

$c :=$ `if` $[\mathtt{z} > 0]^1$ `then`
$[\mathtt{x} := 2]^2; [\mathtt{y} := 3]^3$
`else`
$[\mathtt{x} := 3]^4; [\mathtt{y} := 2]^5$
`end;`
$[\mathtt{z} := \mathtt{x}+\mathtt{y}]^6; [\ldots]^7$

Transfer functions
(for $\delta = (\delta(\mathtt{x}), \delta(\mathtt{y}), \delta(\mathtt{z})) \in D$):

$\varphi_1(a,b,c) = (a,b,c)$
$\varphi_2(a,b,c) = (2,b,c)$
$\varphi_3(a,b,c) = (a,3,c)$
$\varphi_4(a,b,c) = (3,b,c)$
$\varphi_5(a,b,c) = (a,2,c)$
$\varphi_6(a,b,c) = (a,b,a+b)$

1. Fixpoint solution:

$$\begin{aligned}
CP_1 &= \iota && = (\top,\top,\top)\\
CP_2 &= \varphi_1(CP_1) && = (\top,\top,\top)\\
CP_3 &= \varphi_2(CP_2) && = (2,\top,\top)\\
CP_4 &= \varphi_1(CP_1) && = (\top,\top,\top)\\
CP_5 &= \varphi_4(CP_4) && = (3,\top,\top)\\
CP_6 &= \varphi_3(CP_3) \sqcup \varphi_5(CP_5) &&\\
&= (2,3,\top) \sqcup (3,2,\top) && = (\top,\top,\top)\\
CP_7 &= \varphi_6(CP_6) && = (\top,\top,\top)
\end{aligned}$$

2. MOP solution:

$$\begin{aligned}
mop(7) &= \varphi_{[1,2,3,6]}(\top,\top,\top) \sqcup\\
&\quad\ \varphi_{[1,4,5,6]}(\top,\top,\top)\\
&= (2,3,5) \sqcup (3,2,5)\\
&= (\top,\top,5)
\end{aligned}$$

MOP vs. Fixpoint Solution II

Theorem (MOP vs. Fixpoint Solution)

Let $S = (Lab, E, F, (D, \sqsubseteq), \iota, \varphi)$ *be a dataflow system. Then*

$$\mathrm{mop}(S) \sqsubseteq \mathrm{fix}(\Phi_S)$$

Reminder: by Definition 4.9,

$$\Phi_S : D^n \to D^n : (d_1, \ldots, d_n) \mapsto (d'_1, \ldots, d'_n)$$

where $Lab = \{1, \ldots, n\}$ and, for each $l \in Lab$,

$$d'_l := \begin{cases} \iota & \text{if } l \in E \\ \bigsqcup \{\varphi_{l'}(d_{l'}) \mid (l', l) \in F\} & \text{otherwise} \end{cases}$$

Proof.

on the board □

Remark: as Example **??** shows, $\mathrm{mop}(S) \neq \mathrm{fix}(\Phi_S)$ is possible

Recap: The MOP Solution

Distributivity of Transfer Functions

A sufficient condition for the coincidence of MOP and Fixpoint Solution is the distributivity of the transfer functions.

Definition (Distributivity)

- Let $(D, \sqsubseteq)$ and $(D', \sqsubseteq')$ be complete lattices. Function $F : D \to D'$ is called distributive (w.r.t. $(D, \sqsubseteq)$ and $(D', \sqsubseteq')$) if, for every $d_1, d_2 \in D$,

$$F(d_1 \sqcup_D d_2) = F(d_1) \sqcup_{D'} F(d_2).$$

- A dataflow system $S = (Lab, E, F, (D, \sqsubseteq), \iota, \varphi)$ is called distributive if every $\varphi_l : D \to D$ ($l \in Lab$) is so.

Coincidence of MOP and Fixpoint Solution

Theorem (MOP vs. Fixpoint Solution)

Let $S = (Lab, E, F, (D, \sqsubseteq), \iota, \varphi)$ *be a distributive dataflow system. Then*

$$\text{mop}(S) = \text{fix}(\Phi_S)$$

Proof.

- $\text{mop}(S) \sqsubseteq \text{fix}(\Phi_S)$: Theorem **??**
- $\text{fix}(\Phi_S) \sqsubseteq \text{mop}(S)$: as $\text{fix}(\Phi_S)$ is the *least* fixpoint of Φ_S, it suffices to show that $\Phi_S(\text{mop}(S)) = \text{mop}(S)$ (on the board)

□

Outline of Lecture 7

Undecidability of the MOP Solution I

Theorem 7.1 (Undecidability of MOP solution)

The MOP solution for Constant Propagation is undecidable.

Undecidability of the MOP Solution I

Theorem 7.1 (Undecidability of MOP solution)

The MOP solution for Constant Propagation is undecidable.

Proof.

Based on undecidability of Modified Post Correspondence Problem:
Let Γ 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 $u_{i_1} u_{i_2} \ldots u_{i_m} = v_{i_1} v_{i_2} \ldots v_{i_m}$?

Undecidability of the MOP Solution I

Theorem 7.1 (Undecidability of MOP solution)

The MOP solution for Constant Propagation is undecidable.

Proof.

Based on undecidability of Modified Post Correspondence Problem:
Let Γ 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). □

Undecidability of the MOP Solution II

Proof (continued).

```
x := u_1;y := v_1;
while ... do
  if ... then
    x := x ++ u_1;
    y := y ++ v_1
  else if ... then
    ⋮
  else
    x := x ++ u_n;
    y := y ++ v_n
  end ... end
end;
z := (x = y);
[skip]^l
```

Undecidability of the MOP Solution II

Proof (continued).

```
x := u_1;y := v_1;
while ... do
  if ... then
    x := x ++ u_1;
    y := y ++ v_1
  else if ... then
    ⋮
  else
    x := x ++ u_n;
    y := y ++ v_n
  end ... end
end;
z := (x = y);
[skip]^l
```

Then: $\mathrm{mop}(l)(z) = \mathrm{false}$

Undecidability of the MOP Solution II

Proof (continued).

```
x := u_1;y := v_1;
while ... do
  if ... then
    x := x ++ u_1;
    y := y ++ v_1
  else if ... then
    ⋮
  else
    x := x ++ u_n;
    y := y ++ v_n
  end ... end
end;
z := (x = y);
[skip]^l
```

Then: $\mathrm{mop}(l)(\mathtt{z}) = \mathrm{false}$

$\iff \mathtt{x} \neq \mathtt{y}$ at the end of every path to l

Undecidability of the MOP Solution II

Proof (continued).

```
x := u_1;y := v_1;
while ... do
  if ... then
    x := x ++ u_1;
    y := y ++ v_1
  else if ... then
    ⋮
  else
    x := x ++ u_n;
    y := y ++ v_n
  end ... end
end;
z := (x = y);
[skip]^l
```

Then: $\mathrm{mop}(l)(\mathtt{z}) = \mathrm{false}$

$\iff$ $\mathtt{x} \neq \mathtt{y}$ at the end of every path to l

$\iff$ the MPCP has no solution

□

Outline of Lecture 7

Dataflow Analysis with Non-ACC Domains

- **Reminder:** $(D, \sqsubseteq)$ satisfies ACC if each ascending chain $d_0 \sqsubseteq d_1 \sqsubseteq \ldots$ eventually stabilises, i.e., there exists $n \in \mathbb{N}$ such that $d_n = d_{n+1} = \ldots$
- If height (= maximal chain size - 1) of $(D, \sqsubseteq)$ is m, then fixpoint computation terminates after at most $|Lab| \cdot m$ iterations

Dataflow Analysis with Non-ACC Domains

- **Reminder:** $(D, \sqsubseteq)$ satisfies ACC if each ascending chain $d_0 \sqsubseteq d_1 \sqsubseteq \ldots$ eventually stabilises, i.e., there exists $n \in \mathbb{N}$ such that $d_n = d_{n+1} = \ldots$
- If height (= maximal chain size - 1) of $(D, \sqsubseteq)$ is m, then fixpoint computation terminates after at most $|Lab| \cdot m$ iterations
- **But:** if $(D, \sqsubseteq)$ has non-stabilising ascending chains
  $\Longrightarrow$ algorithm may not terminate

Dataflow Analysis with Non-ACC Domains

- **Reminder:** $(D, \sqsubseteq)$ satisfies ACC if each ascending chain $d_0 \sqsubseteq d_1 \sqsubseteq \ldots$ eventually stabilises, i.e., there exists $n \in \mathbb{N}$ such that $d_n = d_{n+1} = \ldots$
- If height (= maximal chain size - 1) of $(D, \sqsubseteq)$ is m, then fixpoint computation terminates after at most $|Lab| \cdot m$ iterations
- **But:** if $(D, \sqsubseteq)$ has non-stabilising ascending chains
  $\Longrightarrow$ algorithm may not terminate
- **Solution:** use widening operators to enforce termination

Outline of Lecture 7

Example: Interval Analysis

Interval Analysis

The goal of Interval Analysis is to determine, for each (interesting) program point, a safe interval for the values of the (interesting) program variables.

Interval analysis is actually a generalisation of constant propagation
($\approx$ interval analysis with 1-element intervals)

Example: Interval Analysis

Interval Analysis

The goal of Interval Analysis is to determine, for each (interesting) program point, a safe interval for the values of the (interesting) program variables.

Interval analysis is actually a generalisation of constant propagation
($\approx$ interval analysis with 1-element intervals)

Example 7.2 (Interval Analysis)

```
var a[100]:  int;
i := 0;
while i <= 42 do
  if i >= 0 ∧ i < 100 then
    a[i] := i
  end;
  i := i + 1;
end;
```

Example: Interval Analysis

Interval Analysis

The goal of Interval Analysis is to determine, for each (interesting) program point, a safe interval for the values of the (interesting) program variables.

Interval analysis is actually a generalisation of constant propagation
($\approx$ interval analysis with 1-element intervals)

Example 7.2 (Interval Analysis)

```
var a[100]:  int;
i := 0;
while i <= 42 do
  if i >= 0 ∧ i < 100 then      ⟸ redundant array bounds check
    a[i] := i
  end;
  i := i + 1;
end;
```

The Domain of Interval Analysis

- The domain $(\mathit{Int}, \subseteq)$ of intervals over $\mathbb{Z}$ is defined by

$$\mathit{Int} := \{[z_1, z_2] \mid z_1 \in \mathbb{Z} \cup \{-\infty\}, z_2 \in \mathbb{Z} \cup \{+\infty\}\}, z_1 \leq z_2\} \cup \{\emptyset\}$$

  where
  - $-\infty \leq z$ and $z \leq +\infty$ (for all $z \in \mathbb{Z}$)
  - $\emptyset \subseteq J$ (for all $J \in \mathit{Int}$)
  - $[y_1, y_2] \subseteq [z_1, z_2]$ iff $y_1 \geq z_1$ and $y_2 \leq z_2$

Example: Interval Analysis

The Domain of Interval Analysis

- The domain $(\mathit{Int}, \subseteq)$ of intervals over $\mathbb{Z}$ is defined by

$$\mathit{Int} := \{[z_1, z_2] \mid z_1 \in \mathbb{Z} \cup \{-\infty\}, z_2 \in \mathbb{Z} \cup \{+\infty\}\}, z_1 \leq z_2\} \cup \{\emptyset\}$$

  where
  - $-\infty \leq z$ and $z \leq +\infty$ (for all $z \in \mathbb{Z}$)
  - $\emptyset \subseteq J$ (for all $J \in \mathit{Int}$)
  - $[y_1, y_2] \subseteq [z_1, z_2]$ iff $y_1 \geq z_1$ and $y_2 \leq z_2$
- $(\mathit{Int}, \subseteq)$ is a complete lattice with (for every $\mathcal{I} \subseteq \mathit{Int}$)

$$\bigsqcup \mathcal{I} = \begin{cases} \emptyset & \text{if } \mathcal{I} = \emptyset \text{ or } \mathcal{I} = \{\emptyset\} \\ [Z_1, Z_2] & \text{otherwise} \end{cases}$$

  where

$$Z_1 := \bigsqcap_{\mathbb{Z} \cup \{-\infty\}} \{z_1 \mid [z_1, z_2] \in \mathcal{I}\}$$
$$Z_2 := \bigsqcup_{\mathbb{Z} \cup \{+\infty\}} \{z_2 \mid [z_1, z_2] \in \mathcal{I}\}$$

  (and thus $\bot = \emptyset$, $\top = [-\infty, +\infty]$)

Example: Interval Analysis

The Domain of Interval Analysis

- The domain $(Int, \subseteq)$ of intervals over $\mathbb{Z}$ is defined by

$$Int := \{[z_1, z_2] \mid z_1 \in \mathbb{Z} \cup \{-\infty\}, z_2 \in \mathbb{Z} \cup \{+\infty\}\}, z_1 \leq z_2\} \cup \{\emptyset\}$$

  where
  - $-\infty \leq z$ and $z \leq +\infty$ (for all $z \in \mathbb{Z}$)
  - $\emptyset \subseteq J$ (for all $J \in Int$)
  - $[y_1, y_2] \subseteq [z_1, z_2]$ iff $y_1 \geq z_1$ and $y_2 \leq z_2$
- $(Int, \subseteq)$ is a complete lattice with (for every $\mathcal{I} \subseteq Int$)

$$\bigsqcup \mathcal{I} = \begin{cases} \emptyset & \text{if } \mathcal{I} = \emptyset \text{ or } \mathcal{I} = \{\emptyset\} \\ [Z_1, Z_2] & \text{otherwise} \end{cases}$$

  where

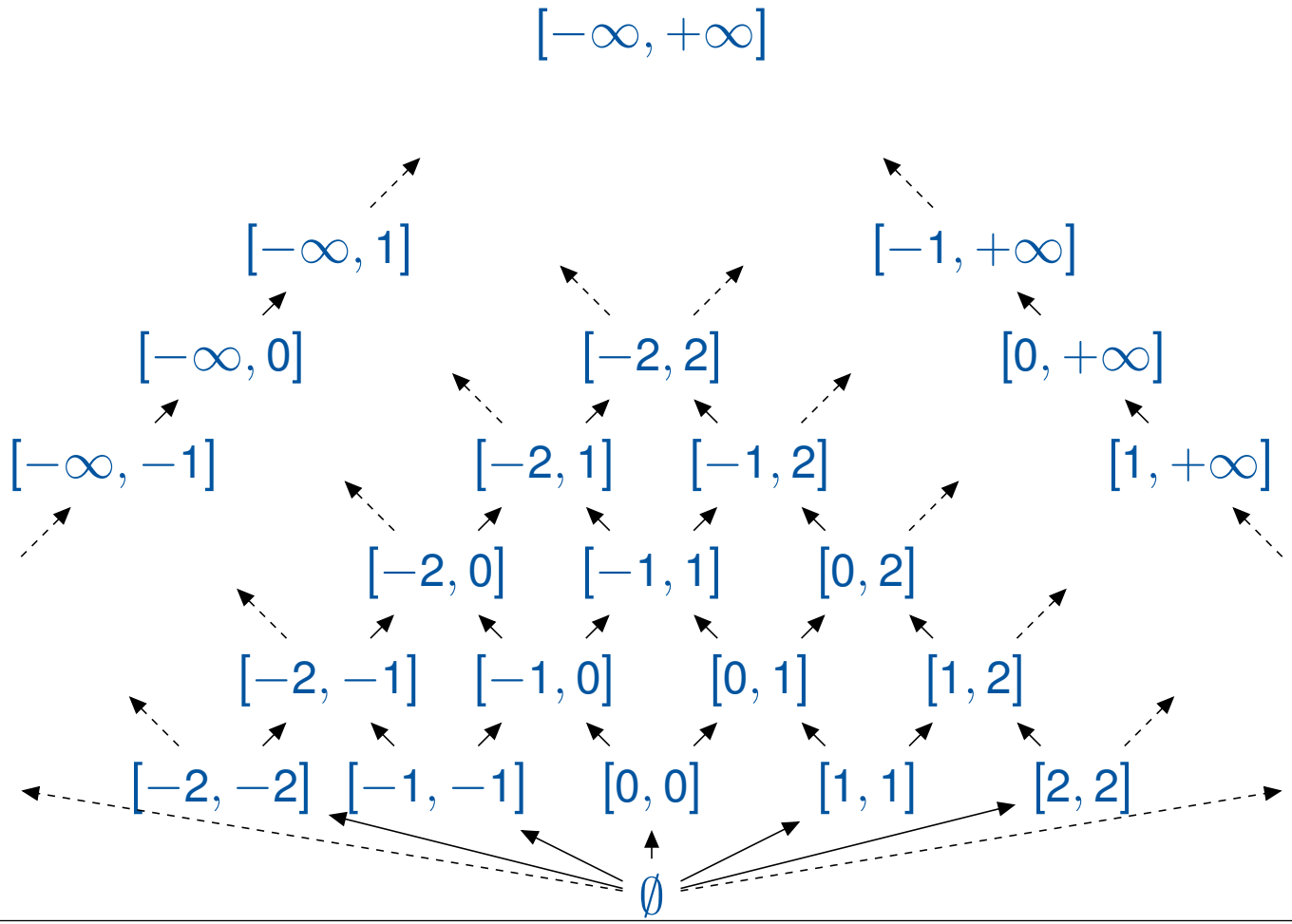
$$Z_1 := \bigsqcap_{\mathbb{Z} \cup \{-\infty\}} \{z_1 \mid [z_1, z_2] \in \mathcal{I}\}$$
$$Z_2 := \bigsqcup_{\mathbb{Z} \cup \{+\infty\}} \{z_2 \mid [z_1, z_2] \in \mathcal{I}\}$$

  (and thus $\bot = \emptyset$, $\top = [-\infty, +\infty]$)
- Clearly $(Int, \subseteq)$ has infinite ascending chains, such as

$$\emptyset \subseteq [1,1] \subseteq [1,2] \subseteq [1,3] \subseteq \ldots$$

Example: Interval Analysis

The Complete Lattice of Interval Analysis



Outline of Lecture 7

Formalising Interval 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)
- complete lattice $(D, \sqsubseteq)$ where
  - $D := \{\delta \mid \delta : Var_c \to Int\}$
  - $\delta_1 \sqsubseteq \delta_2$ iff $\delta_1(x) \subseteq \delta_2(x)$ for every $x \in Var_c$
- $\iota := \top_D : Var_c \to Int : x \mapsto \top_{Int}$ (with $\top_{Int} = [-\infty, +\infty]$)
- φ: see next slide

Formalising Interval Analysis II

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

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

where

$$val_\delta(x) := \delta(x) \qquad val_\delta(a_1 + a_2) := val_\delta(a_1) \oplus val_\delta(a_2)$$
$$val_\delta(z) := [z, z] \qquad val_\delta(a_1 - a_2) := val_\delta(a_1) \ominus val_\delta(a_2)$$
$$val_\delta(a_1 * a_2) := val_\delta(a_1) \odot val_\delta(a_2)$$

with

$$\emptyset \oplus J := J \oplus \emptyset := \emptyset \ominus J := \ldots := \emptyset$$
$$[y_1, y_2] \oplus [z_1, z_2] := [y_1 + z_1, y_2 + z_2]$$
$$[y_1, y_2] \ominus [z_1, z_2] := [y_1 - z_2, y_2 - z_1]$$
$$[y_1, y_2] \odot [z_1, z_2] := [\textstyle\bigsqcap\{y_1 z_1, y_1 z_2, y_2 z_1, y_2 z_2\}, \bigsqcup\{y_1 z_1, y_1 z_2, y_2 z_1, y_2 z_2\}]$$

Formalising Interval Analysis

Remarks

- Possible refinement of DFA to take conditional blocks b^l ($b \in BExp$) into account
  - essentially: b as edge label, $\varphi_l(\delta)(x) = \delta(x) \setminus \{z \in \mathbb{Z} \mid x = z \implies \neg b\}$
    (cf. “DFA with Conditional Branches” later)

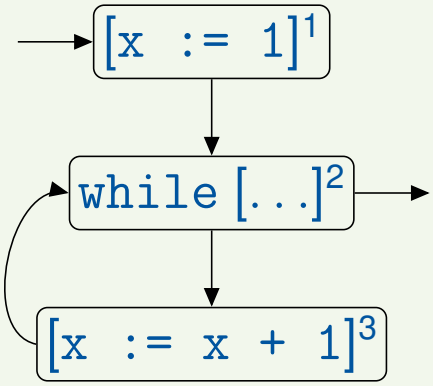
Formalising Interval Analysis

Remarks

- Possible refinement of DFA to take conditional blocks b^l ($b \in BExp$) into account
  - essentially: b as edge label, $\varphi_l(\delta)(x) = \delta(x) \setminus \{z \in \mathbb{Z} \mid x = z \implies \neg b\}$
    (cf. “DFA with Conditional Branches” later)
- Important: soundness and optimality of abstract operations, e.g., $\oplus$:
  - soundness: $z_1 \in J_1, z_2 \in J_2 \implies z_1 + z_2 \in J_1 \oplus J_2$
  - optimality: $J_1 \oplus J_2$ as precise (i.e., small) as possible

Interval Analysis without Widening

Example 7.3



Transfer functions (for $\delta(x) = J$):

$$\varphi_1(J) = [1, 1]$$

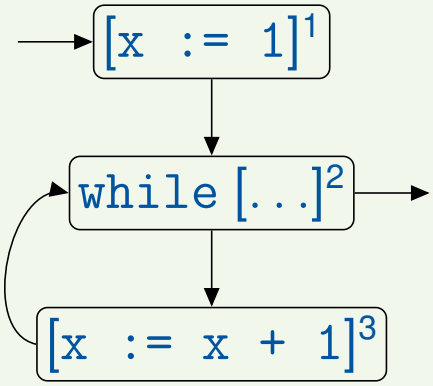
$$\varphi_2(J) = J$$

$$\varphi_3(\emptyset) = \emptyset$$

$$\varphi_3([x_1, x_2]) = [x_1 + 1, x_2 + 1]$$

Interval Analysis without Widening

Example 7.3



Transfer functions (for $\delta(x) = J$):

$$\varphi_1(J) = [1, 1]$$
$$\varphi_2(J) = J$$
$$\varphi_3(\emptyset) = \emptyset$$
$$\varphi_3([x_1, x_2]) = [x_1 + 1, x_2 + 1]$$

Application of worklist algorithm without widening: does not terminate (on the board)

Outline of Lecture 7

Widening Operators

Definition 7.4 (Widening operator)

Let $(D, \sqsubseteq)$ be a complete lattice. A mapping $\nabla : D \times D \to D$ 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

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

Widening Operators

Definition 7.4 (Widening operator)

Let $(D, \sqsubseteq)$ be a complete lattice. A mapping $\nabla : D \times D \to D$ 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 $d_{i+1}^\nabla = d_i^\nabla \nabla d_{i+1} \sqsupseteq d_i^\nabla \sqcup d_{i+1} \sqsupseteq d_i^\nabla$
- In contrast to $\sqcup$, ∇ 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

Applying Widening to Interval Analysis

- A widening operator: $\nabla : \mathit{Int} \times \mathit{Int} \to \mathit{Int}$ with

$$\emptyset \nabla J := J \nabla \emptyset := J$$

$$[x_1, x_2] \nabla [y_1, y_2] := [z_1, z_2] \quad \text{where}$$

$$z_1 := \begin{cases} x_1 & \text{if } x_1 \leq y_1 \\ -\infty & \text{otherwise} \end{cases}$$

$$z_2 := \begin{cases} x_2 & \text{if } x_2 \geq y_2 \\ +\infty & \text{otherwise} \end{cases}$$

Applying Widening to Interval Analysis

- A widening operator: $\nabla : \mathit{Int} \times \mathit{Int} \to \mathit{Int}$ with

$$\emptyset \nabla J := J \nabla \emptyset := J$$
$$[x_1, x_2] \nabla [y_1, y_2] := [z_1, z_2] \quad \text{where}$$
$$z_1 := \begin{cases} x_1 & \text{if } x_1 \leq y_1 \\ -\infty & \text{otherwise} \end{cases}$$
$$z_2 := \begin{cases} x_2 & \text{if } x_2 \geq y_2 \\ +\infty & \text{otherwise} \end{cases}$$

- 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:

$$J_0^\nabla = J_0 = \emptyset$$
$$J_1^\nabla = J_0^\nabla \nabla J_1 = \emptyset \nabla [1,1] = [1,1]$$
$$J_2^\nabla = J_1^\nabla \nabla J_2 = [1,1] \nabla [1,2] = [1,+\infty]$$
$$J_3^\nabla = J_2^\nabla \nabla J_3 = [1,+\infty] \nabla [1,3] = [1,+\infty]$$

Applying Widening to Interval Analysis

- A widening operator: $\nabla : \mathit{Int} \times \mathit{Int} \to \mathit{Int}$ with

$$\emptyset \nabla J := J \nabla \emptyset := J$$

$$[x_1, x_2] \nabla [y_1, y_2] := [z_1, z_2] \quad \text{where}$$

$$z_1 := \begin{cases} x_1 & \text{if } x_1 \leq y_1 \\ -\infty & \text{otherwise} \end{cases}$$

$$z_2 := \begin{cases} x_2 & \text{if } x_2 \geq y_2 \\ +\infty & \text{otherwise} \end{cases}$$

- 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:

$$J_0^\nabla = J_0 = \emptyset$$
$$J_1^\nabla = J_0^\nabla \nabla J_1 = \emptyset \nabla [1,1] = [1,1]$$
$$J_2^\nabla = J_1^\nabla \nabla J_2 = [1,1] \nabla [1,2] = [1,+\infty]$$
$$J_3^\nabla = J_2^\nabla \nabla J_3 = [1,+\infty] \nabla [1,3] = [1,+\infty]$$

- In fact, the maximal chain size arising with this operator is 4:

$$\emptyset \subseteq [3,7] \subseteq [3,+\infty] \subseteq [-\infty,+\infty]$$

Applying Widening to Interval Analysis

Worklist Algorithm with Widening I

Goal: extend Algorithm 5.1 by widening to ensure termination

Algorithm 7.5 (Worklist algorithm)

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

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

Procedure:

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

  for $l \in Lab$ **do**

    if $l \in E$ **then** $\mathsf{AI}_l := \iota$ **else** $\mathsf{AI}_l := \bot_D$; *% Initialise* AI

  while $W \neq \varepsilon$ **do**

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

    if $\varphi_l(\mathsf{AI}_l) \not\sqsubseteq \mathsf{AI}_{l'}$ **then** *% Fixpoint not yet reached*

      $\mathsf{AI}_{l'} := \mathsf{AI}_{l'} \sqcup \varphi_l(\mathsf{AI}_l)$; *% Update analysis information*

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

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

Output: $\{\mathsf{AI}_l \mid l \in Lab\}$

Applying Widening to Interval Analysis

Worklist Algorithm with Widening I

Goal: extend Algorithm 5.1 by widening to ensure termination

Algorithm 7.5 (Worklist algorithm with widening)

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

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

Procedure:

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

for $l \in Lab$ **do**

if $l \in E$ **then** $\mathsf{AI}_l := \iota$ **else** $\mathsf{AI}_l := \bot_D$; *% Initialise* AI

while $W \neq \varepsilon$ **do**

$(l, l') :=$ **head**(W); $W :=$ **tail**(W); *% Next control-flow edge*

if $\varphi_l(\mathsf{AI}_l) \not\sqsubseteq \mathsf{AI}_{l'}$ **then** *% Fixpoint not yet reached*

$\mathsf{AI}_{l'} := \mathsf{AI}_{l'} \nabla \varphi_l(\mathsf{AI}_l)$; *% Update analysis information*

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

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

Output: $\{\mathsf{AI}_l \mid l \in Lab\}$, *denoted by* $\mathrm{fix}^{\nabla}(\Phi_S)$

Applying Widening to Interval Analysis

Worklist Algorithm with Widening I

Goal: extend Algorithm 5.1 by widening to ensure termination

Algorithm 7.5 (Worklist algorithm with widening)

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

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

Procedure:

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

for $l \in Lab$ **do**

if $l \in E$ **then** $\mathsf{AI}_l := \iota$ **else** $\mathsf{AI}_l := \bot_D$; *% Initialise* AI

while $W \neq \varepsilon$ **do**

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

if $\varphi_l(\mathsf{AI}_l) \not\sqsubseteq \mathsf{AI}_{l'}$ **then** *% Fixpoint not yet reached*

$\mathsf{AI}_{l'} := \mathsf{AI}_{l'} \nabla \varphi_l(\mathsf{AI}_l)$; *% Update analysis information*

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

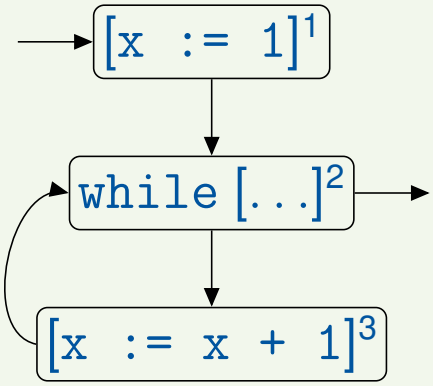
if (l', l'') *not in* W **then** $W := (l', l'') \cdot W$; *% Propagate modification*

Output: $\{\mathsf{AI}_l \mid l \in Lab\}$, *denoted by* $\mathrm{fix}^{\nabla}(\Phi_S)$

Remark: due to widening, only $\mathrm{fix}^{\nabla}(\Phi_S) \sqsupseteq \mathrm{fix}(\Phi_S)$ is guaranteed (cf. Thm. 5.4)

Worklist Algorithm with Widening II

Example 7.6



Transfer functions (for $\delta(x) = J$):

$$\varphi_1(J) = [1, 1]$$

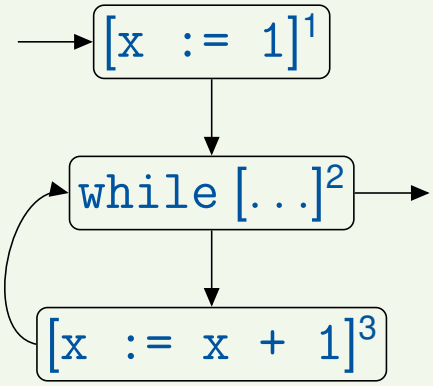
$$\varphi_2(J) = J$$

$$\varphi_3(\emptyset) = \emptyset$$

$$\varphi_3([x_1, x_2]) = [x_1 + 1, x_2 + 1]$$

Worklist Algorithm with Widening II

Example 7.6



Transfer functions (for $\delta(x) = J$):

$$\varphi_1(J) = [1, 1]$$
$$\varphi_2(J) = J$$
$$\varphi_3(\emptyset) = \emptyset$$
$$\varphi_3([x_1, x_2]) = [x_1 + 1, x_2 + 1]$$

Application of worklist algorithm with widening:
terminates with expected result for AI_2 ($[1, +\infty]$) (on the board)