Static Program Analysis

Lecture 8: Dataflow Analysis VII (Narrowing & DFA with Conditional Branches)

Thomas Noll

Lehrstuhl für Informatik 2
(Software Modeling and Verification)

RWTH AACHEN UNIVERSITY

noll@cs.rwth-aachen.de

http://moves.rwth-aachen.de/teaching/ws-1415/spa/

Winter Semester 2014/15

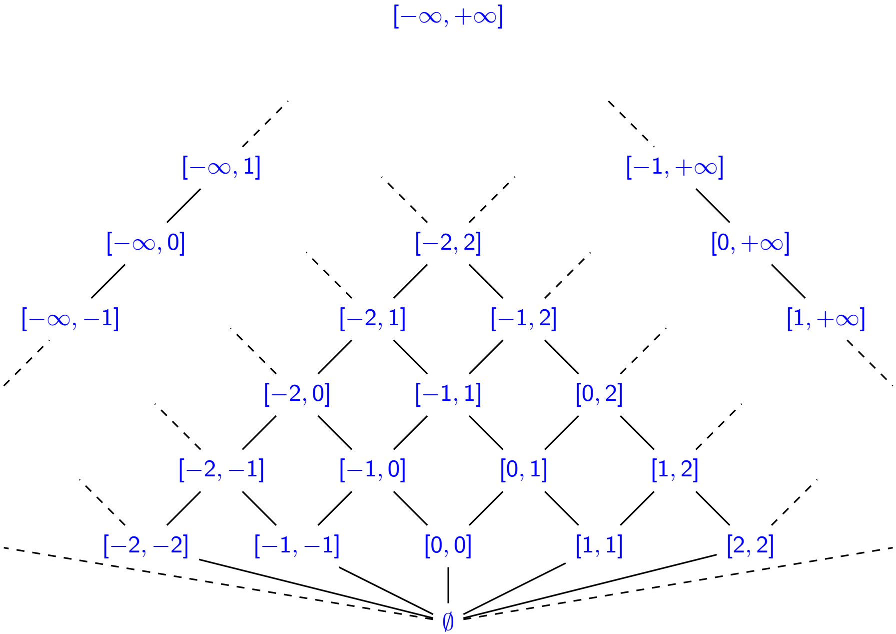
Outline

1. Recap: Interval Analysis
2. Narrowing
3. Taking Conditional Branches into Account
4. Constant Propagation Analysis with Assertions

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
  $$\begin{aligned} Z_1 &:= \textstyle\bigsqcap_{\mathbb{Z} \cup \{-\infty\}} \{z_1 \mid [z_1, z_2] \in \mathcal{I}\} \\ Z_2 &:= \textstyle\bigsqcup_{\mathbb{Z} \cup \{+\infty\}} \{z_2 \mid [z_1, z_2] \in \mathcal{I}\} \end{aligned}$$
  (and thus $\bot = \emptyset$, $\top = [-\infty, +\infty]$)
- Clearly $(Int, \subseteq)$ has infinite ascending chains, such as
  $$\emptyset \subseteq [1, 1] \subseteq [1, 2] \subseteq [1, 3] \subseteq \ldots$$

The Complete Lattice of Interval Analysis



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 $E := \{\mathrm{init}(c)\}$ (forward problem)
- flow relation $F := \mathrm{flow}(c)$ (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

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

where

$$\begin{aligned} val_\delta(x) &:= \delta(x) \\ val_\delta(z) &:= [z, z] \end{aligned} \qquad \begin{aligned} val_\delta(a_1\texttt{+}a_2) &:= val_\delta(a_1) \oplus val_\delta(a_2) \\ val_\delta(a_1\texttt{-}a_2) &:= val_\delta(a_1) \ominus val_\delta(a_2) \\ val_\delta(a_1\texttt{*}a_2) &:= val_\delta(a_1) \odot val_\delta(a_2) \end{aligned}$$

with

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

$$\begin{aligned} [y_1, y_2] \oplus [z_1, z_2] &:= [y_1 + z_1, y_2 + z_2] \\ [y_1, y_2] \ominus [z_1, z_2] &:= [y_1 - z_2, y_2 - z_1] \\ [y_1, y_2] \odot [z_1, z_2] &:= [\textstyle\bigsqcap\{y_1z_1, y_1z_2, y_2z_1, y_2z_2\}, \bigsqcup\{y_1z_1, y_1z_2, y_2z_1, y_2z_2\}] \end{aligned}$$

Remarks:

- Possible refinement of DFA to take conditional blocks b^l 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 small as possible

Widening Operators

Definition (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: $\nabla : Int \times Int \to Int$ with

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

- 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:

$$\begin{aligned}
J_0^\nabla &= J_0 = \emptyset \\
J_1^\nabla &= J_0^\nabla \nabla J_1 = \emptyset \nabla [1,1] = [1,1] \\
J_2^\nabla &= J_1^\nabla \nabla J_2 = [1,1] \nabla [1,2] = [1,+\infty] \\
J_3^\nabla &= J_2^\nabla \nabla J_3 = [1,+\infty] \nabla [1,3] = [1,+\infty]
\end{aligned}$$

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

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

Worklist Algorithm with Widening

Goal: extend Algorithm 5.3 by widening to ensure termination

Algorithm (Worklist algorithm with widening)

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

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

Procedure: $W := \varepsilon$; **for** $(l, l') \in F$ **do** $W := W \cdot (l, l')$; *% Initialize W*
  for $l \in Lab$ **do** *% Initialise AI*
    if $l \in E$ **then** $\mathrm{AI}_l := \iota$ **else** $\mathrm{AI}_l := \bot_D$;
  while $W \neq \varepsilon$ **do**
    $(l, l') :=$ **head**(W); $W :=$ **tail**(W);
    if $\varphi_l(\mathrm{AI}_l) \not\sqsubseteq \mathrm{AI}_{l'}$ **then** *% Fixpoint not yet reached*
      $\mathrm{AI}_{l'} := \mathrm{AI}_{l'} \nabla \varphi_l(\mathrm{AI}_l)$;
      for $(l', l'') \in F$ **do**
        if (l', l'') *not in* W **then** $W := (l', l'') \cdot W$;

Output: $\{\mathrm{AI}_l \mid l \in Lab\}$, *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. 5.6)

Outline

1 Recap: Interval Analysis

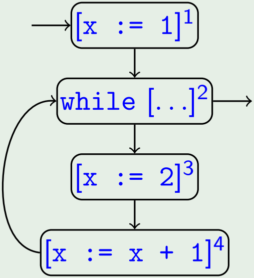
2 Narrowing

3 Taking Conditional Branches into Account

4 Constant Propagation Analysis with Assertions

Another Widening Example

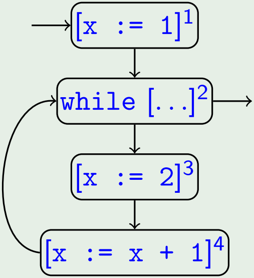
Example 8.1



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

$$
\begin{aligned}
\varphi_1(J) &= [1,1] \\
\varphi_2(J) &= J \\
\varphi_3(J) &= [2,2] \\
\varphi_4(\emptyset) &= \emptyset \\
\varphi_4([x_1,x_2]) &= [x_1+1, x_2+1]
\end{aligned}
$$

Example 8.1



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

$$
\begin{aligned}
\varphi_1(J) &= [1, 1] \\
\varphi_2(J) &= J \\
\varphi_3(J) &= [2, 2] \\
\varphi_4(\emptyset) &= \emptyset \\
\varphi_4([x_1, x_2]) &= [x_1 + 1, x_2 + 1]
\end{aligned}
$$

Application of worklist algorithm

1. without widening (omitted):
   terminates with expected result for AI_2 ($[1, 3]$)
2. with widening (on the board):
   terminates with unexpected result for AI_2 ($[1, +\infty]$)

Idea of Narrowing

- **Observation:** widening can lead to unnecessarily imprecise results

Idea of Narrowing

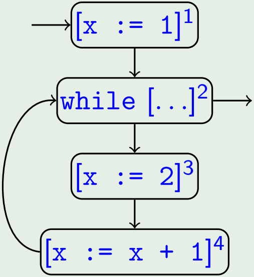
- **Observation:** widening can lead to unnecessarily imprecise results
- **Solution:** improvement by iterating again from the result obtained by widening (i.e., from $\mathrm{fix}^\nabla(\Phi_S)$)
  $\Longrightarrow$ compute $\Phi_S^k(\mathrm{fix}^\nabla(\Phi_S))$ for $k = 1, 2, \ldots$

Idea of Narrowing

- **Observation:** widening can lead to unnecessarily imprecise results
- **Solution:** improvement by iterating again from the result obtained by widening (i.e., from $\mathrm{fix}^\nabla(\Phi_S)$)
  $\Longrightarrow$ compute $\Phi_S^k(\mathrm{fix}^\nabla(\Phi_S))$ for $k = 1, 2, \ldots$
- **Soundness:** $\mathrm{fix}^\nabla(\Phi_S) \sqsupseteq \mathrm{fix}(\Phi_S)$ (cf. Alg. 7.7)
  $\Longrightarrow \Phi_S^k(\mathrm{fix}^\nabla(\Phi_S)) \sqsupseteq \Phi_S^k(\mathrm{fix}(\Phi_S)) = \mathrm{fix}(\Phi_S)$
  (since Φ_S and thus Φ_S^k monotonic)

Narrowing Example

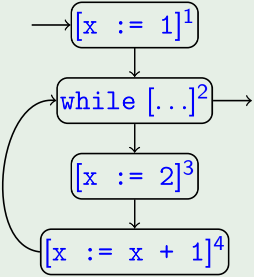
Example 8.2 (cf. Example 8.1)



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

$$\begin{aligned}
\varphi_1(J) &= [1,1] \\
\varphi_2(J) &= J \\
\varphi_3(J) &= [2,2] \\
\varphi_4(\emptyset) &= \emptyset \\
\varphi_4([x_1,x_2]) &= [x_1+1, x_2+1]
\end{aligned}$$

Example 8.2 (cf. Example 8.1)



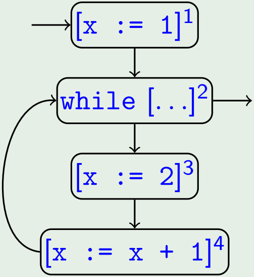
Transfer functions (for $\delta(\mathtt{x}) = J$):

$$
\begin{aligned}
\varphi_1(J) &= [1,1] \\
\varphi_2(J) &= J \\
\varphi_3(J) &= [2,2] \\
\varphi_4(\emptyset) &= \emptyset \\
\varphi_4([x_1,x_2]) &= [x_1+1, x_2+1]
\end{aligned}
$$

Narrowing:

	AI_1	AI_2	AI_3	AI_4
$\mathrm{fix}^{\nabla}(\Phi_S)$	$[-\infty,+\infty]$	$[1,+\infty]$	$[1,+\infty]$	$[2,2]$

Example 8.2 (cf. Example 8.1)



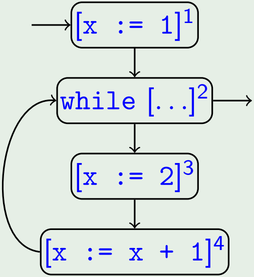
Transfer functions (for $\delta(\mathtt{x}) = J$):

$$
\begin{aligned}
\varphi_1(J) &= [1,1] \\
\varphi_2(J) &= J \\
\varphi_3(J) &= [2,2] \\
\varphi_4(\emptyset) &= \emptyset \\
\varphi_4([x_1,x_2]) &= [x_1+1, x_2+1]
\end{aligned}
$$

Narrowing:

	AI_1	AI_2	AI_3	AI_4
$\mathrm{fix}^\nabla(\Phi_S)$	$[-\infty,+\infty]$	$[1,+\infty]$	$[1,+\infty]$	$[2,2]$
$\Phi_S(\mathrm{fix}^\nabla(\Phi_S))$	$[-\infty,+\infty]$	$[1,3]$	$[1,+\infty]$	$[2,2]$

Example 8.2 (cf. Example 8.1)



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

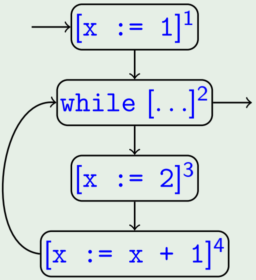
$$
\begin{aligned}
\varphi_1(J) &= [1,1] \\
\varphi_2(J) &= J \\
\varphi_3(J) &= [2,2] \\
\varphi_4(\emptyset) &= \emptyset \\
\varphi_4([x_1,x_2]) &= [x_1+1, x_2+1]
\end{aligned}
$$

Narrowing:

	AI_1	AI_2	AI_3	AI_4
$\mathsf{fix}^\nabla(\Phi_S)$	$[-\infty,+\infty]$	$[1,+\infty]$	$[1,+\infty]$	$[2,2]$
$\Phi_S(\mathsf{fix}^\nabla(\Phi_S))$	$[-\infty,+\infty]$	$[1,3]$	$[1,+\infty]$	$[2,2]$
$\Phi_S^2(\mathsf{fix}^\nabla(\Phi_S))$	$[-\infty,+\infty]$	$[1,3]$	$[1,3]$	$[2,2]$

Narrowing Example

Example 8.2 (cf. Example 8.1)



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

$$
\begin{aligned}
\varphi_1(J) &= [1,1] \\
\varphi_2(J) &= J \\
\varphi_3(J) &= [2,2] \\
\varphi_4(\emptyset) &= \emptyset \\
\varphi_4([x_1,x_2]) &= [x_1+1, x_2+1]
\end{aligned}
$$

Narrowing:

	AI_1	AI_2	AI_3	AI_4
$\mathrm{fix}^{\nabla}(\Phi_S)$	$[-\infty,+\infty]$	$[1,+\infty]$	$[1,+\infty]$	$[2,2]$
$\Phi_S(\mathrm{fix}^{\nabla}(\Phi_S))$	$[-\infty,+\infty]$	$[1,3]$	$[1,+\infty]$	$[2,2]$
$\Phi_S^2(\mathrm{fix}^{\nabla}(\Phi_S))$	$[-\infty,+\infty]$	$[1,3]$	$[1,3]$	$[2,2]$
$\Phi_S^3(\mathrm{fix}^{\nabla}(\Phi_S))$	$[-\infty,+\infty]$	$[1,3]$	$[1,3]$	$[2,2]$

Narrowing in Practice

- **Problem:** narrowing may not terminate
  (due to infinite descending chains)

Narrowing in Practice

- **Problem:** narrowing may not terminate
  (due to infinite descending chains)
- **But:** possible to stop after every step without losing soundness

Narrowing in Practice

- **Problem:** narrowing may not terminate
  (due to infinite descending chains)
- **But:** possible to stop after every step without losing soundness
- **In practice:** termination often ensured by using narrowing operators
  ($\approx$ counterpart of widening operator; definition omitted)

Outline

1. Recap: Interval Analysis
2. Narrowing
3. Taking Conditional Branches into Account
4. Constant Propagation Analysis with Assertions

Taking Conditional Branches into Account I

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

Taking Conditional Branches into Account I

- **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

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

Taking Conditional Branches into Account I

- **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

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

- Interval analysis (with widening) yields for l:

$$x \in [-\infty, +\infty]$$
$$y \in [0, +\infty]$$
$$z \in [-\infty, +\infty]$$

Taking Conditional Branches into Account I

- **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

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

- Interval analysis (with widening) yields for l:

$$x \in [-\infty, +\infty]$$
$$y \in [0, +\infty]$$
$$z \in [-\infty, +\infty]$$

- Too pessimistic! In fact,

$$x \in [-\infty, +\infty]$$
$$y \in [0, 17]$$
$$z \in [0, +\infty]$$

Taking Conditional Branches into Account II

- **Solution:** introduce transfer functions for branches

Taking Conditional Branches into Account II

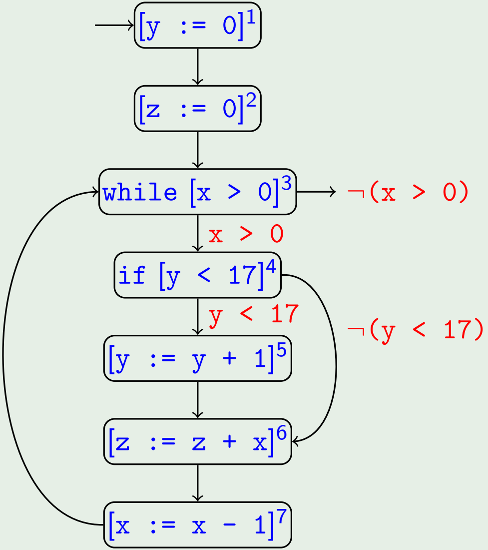
- **Solution:** introduce transfer functions for branches
- **First approach:** attach (negated) conditions as labels to control flow edges
  - advantage: no language modification required
  - disadvantage: entails extension of DFA framework
  - will not further be considered here

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 (statements)
  - advantage: DFA framework can be reused
  - disadvantage: entails extension of WHILE language
  - the way we will follow

First Approach: Conditions as Edge Labels

Example 8.4 (cf. Example 8.3)



Second Approach: Conditions as Assertions

Example 8.5 (cf. Example 8.3)

```
y := 0;
z := 0;
while x > 0 do
  assert x > 0;
  if y < 17 then
    assert y < 17;
    y := y + 1;
  z := z + x;
  x := x - 1;
assert ¬(x > 0);
```

Definition 8.6 (Labelled WHILE programs with assertions)

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

$$
\begin{aligned}
a &::= z \mid x \mid a_1\texttt{+}a_2 \mid a_1\texttt{-}a_2 \mid a_1\texttt{*}a_2 \in AExp\\
b &::= t \mid a_1\texttt{=}a_2 \mid a_1\texttt{>}a_2 \mid \neg b \mid b_1 \wedge b_2 \mid b_1 \vee b_2 \in BExp\\
c &::= [\texttt{skip}]^l \mid [x\ \texttt{:=}\ a]^l \mid c_1\texttt{;}c_2 \mid\\
&\qquad \texttt{if}\ [b]^l\ \texttt{then}\ c_1\ \texttt{else}\ c_2 \mid \texttt{while}\ [b]^l\ \texttt{do}\ c \mid [\texttt{assert}\ b]^l \in Cmd
\end{aligned}
$$

Definition 8.6 (Labelled WHILE programs with assertions)

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

$$
\begin{aligned}
a &::= z \mid x \mid a_1 \texttt{+} a_2 \mid a_1 \texttt{-} a_2 \mid a_1 \texttt{*} a_2 \in AExp \\
b &::= t \mid a_1 \texttt{=} a_2 \mid a_1 \texttt{>} a_2 \mid \neg b \mid b_1 \wedge b_2 \mid b_1 \vee b_2 \in BExp \\
c &::= [\texttt{skip}]^l \mid [x \texttt{ := } a]^l \mid c_1 \texttt{;} c_2 \mid \\
&\quad\ \texttt{if } [b]^l \texttt{ then } c_1 \texttt{ else } c_2 \mid \texttt{while } [b]^l \texttt{ do } c \mid [\texttt{assert } b]^l \in Cmd
\end{aligned}
$$

To be done:

- Definition of transfer functions for `assert` blocks (depending on analysis problem)
- Idea: assertions as filters that let only “valid” information pass

Outline

1 Recap: Interval Analysis

2 Narrowing

3 Taking Conditional Branches into Account

4 Constant Propagation Analysis with Assertions

Constant Propagation Analysis with Assertions I

So far:

- 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
    - $\delta(x) = \bot$: x undefined
    - $\delta(x) = \top$: x overdefined (i.e., different possible values)
  - $\sqsubseteq \subseteq D \times D$ defined by pointwise extension of $\bot \sqsubseteq z \sqsubseteq \top$ (for every $z \in \mathbb{Z}$)
- Transfer functions $\{\varphi_l \mid l \in Lab\}$ defined by

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

where

$$\begin{aligned} val_\delta(x) &:= \delta(x) \\ val_\delta(z) &:= z \end{aligned} \qquad val_\delta(a_1 \; op \; a_2) := \begin{cases} z_1 \; op \; z_2 & \text{if } z_1, z_2 \in \mathbb{Z} \\ \bot & \text{if } z_1 = \bot \text{ or } z_2 = \bot \\ \top & \text{otherwise} \end{cases}$$

for $z_1 := val_\delta(a_1)$ and $z_2 := val_\delta(a_2)$

Constant Propagation Analysis with Assertions II

Additionally for $B^l = (\texttt{assert}\ b)$, $\delta : Var_c \to \mathbb{Z} \cup \{\bot, \top\}$ and $x \in Var_c$:

$$\varphi_l(\delta)(x) := \begin{cases} \bot & \text{if } \nexists \sigma \in \Sigma_\delta : val_\sigma(b) = \text{true} \\ z & \text{if } \forall \sigma \in \Sigma_\delta : val_\sigma(b) = \text{true} \implies \sigma(x) = z \\ \top & \text{otherwise} \end{cases}$$

where

- the set of δ-assignments is given by

$$\Sigma_\delta := \left\{ \sigma : Var_c \to \mathbb{Z} \,\middle|\, \forall y \in Var_c : \sigma(y) \in \begin{cases} \emptyset & \text{if } \delta(y) = \bot \\ \{z\} & \text{if } \delta(y) = z \\ \mathbb{Z} & \text{if } \delta(y) = \top \end{cases} \right\}$$

  (and thus $\Sigma_\delta = \emptyset$ iff $\delta(y) = \bot$ for some $y \in Var_c$)
- the evaluation function $val_\sigma : BExp \to \mathbb{B}$ is defined by

$$val_\sigma(t) := t$$
$$val_\sigma(a_1 {=} a_2) := (val_\sigma(a_1) = val_\sigma(a_2))$$

$$val_\sigma(\neg b) := \begin{cases} \text{true} & \text{if } val_\sigma(b) = \text{false} \\ \text{false} & \text{otherwise} \end{cases}$$

$$val_\sigma(b_1 \wedge b_2) := \begin{cases} \text{true} & \text{if } val_\sigma(b_1) = val_\sigma(b_2) = \text{true} \\ \text{false} & \text{otherwise} \end{cases}$$

  etc.

Constant Propagation Analysis with Assertions III

Example 8.7

1. $\mathit{Var}_c = \{\mathtt{x}, \mathtt{y}, \mathtt{z}\}$, $\delta = (\underbrace{\bot}_{\mathtt{x}}, \underbrace{1}_{\mathtt{y}}, \underbrace{\top}_{\mathtt{z}})$
   $\implies \Sigma_\delta = \emptyset$
   $\implies \varphi_{\mathtt{assert}\ b}(\delta) = (\bot, \bot, \bot)$ for every $b \in \mathit{BExp}$

Constant Propagation Analysis with Assertions III

Example 8.7

1. $Var_c = \{\mathtt{x}, \mathtt{y}, \mathtt{z}\}$, $\delta = (\underbrace{\bot}_{\mathtt{x}}, \underbrace{1}_{\mathtt{y}}, \underbrace{\top}_{\mathtt{z}})$

   $\implies \Sigma_\delta = \emptyset$

   $\implies \varphi_{\mathtt{assert}\ b}(\delta) = (\bot, \bot, \bot)$ for every $b \in BExp$

2. $Var_c = \{\mathtt{x}, \mathtt{y}, \mathtt{z}\}$, $\delta = (\underbrace{1}_{\mathtt{x}}, \underbrace{2}_{\mathtt{y}}, \underbrace{\top}_{\mathtt{z}})$

   $\implies \Sigma_\delta = \{(1, 2, z) \mid z \in \mathbb{Z}\}$

   $$\implies \begin{aligned} \varphi_{\mathtt{assert\ x=y}}(\delta) &= (\bot, \bot, \bot) \\ \varphi_{\mathtt{assert\ y=z}}(\delta) &= (1, 2, 2) \\ \varphi_{\mathtt{assert\ y<z}}(\delta) &= (1, 2, \top) \\ \varphi_{\mathtt{assert\ x<=z\wedge y>z}}(\delta) &= (1, 2, 1) \end{aligned}$$

Constant Propagation Analysis with Assertions III

Example 8.7

1. $Var_c = \{\mathrm{x}, \mathrm{y}, \mathrm{z}\}$, $\delta = (\underbrace{\bot}_{\mathtt{x}}, \underbrace{1}_{\mathtt{y}}, \underbrace{\top}_{\mathtt{z}})$
   $\Longrightarrow \Sigma_\delta = \emptyset$
   $\Longrightarrow \varphi_{\mathtt{assert}\ b}(\delta) = (\bot, \bot, \bot)$ for every $b \in BExp$
2. $Var_c = \{\mathrm{x}, \mathrm{y}, \mathrm{z}\}$, $\delta = (\underbrace{1}_{\mathtt{x}}, \underbrace{2}_{\mathtt{y}}, \underbrace{\top}_{\mathtt{z}})$
   $\Longrightarrow \Sigma_\delta = \{(1, 2, z) \mid z \in \mathbb{Z}\}$
   $$\Longrightarrow \begin{aligned} \varphi_{\mathtt{assert\ x=y}}(\delta) &= (\bot, \bot, \bot) \\ \varphi_{\mathtt{assert\ y=z}}(\delta) &= (1, 2, 2) \\ \varphi_{\mathtt{assert\ y<z}}(\delta) &= (1, 2, \top) \\ \varphi_{\mathtt{assert\ x<=z\wedge y>z}}(\delta) &= (1, 2, 1) \end{aligned}$$
3. $Var_c = \{\mathrm{x}, \mathrm{y}, \mathrm{z}\}$, $\delta = (\underbrace{1}_{\mathtt{x}}, \underbrace{\top}_{\mathtt{y}}, \underbrace{\top}_{\mathtt{z}})$
   $\Longrightarrow \Sigma_\delta = \{(1, z_1, z_2) \mid z_1, z_2 \in \mathbb{Z}\}$
   $$\Longrightarrow \begin{aligned} \varphi_{\mathtt{assert\ x=y}}(\delta) &= (1, 1, \top) \\ \varphi_{\mathtt{assert\ y=z}}(\delta) &= (1, \top, \top) \end{aligned}$$

Remarks:

- For $B^l = (\texttt{assert}\ b)$ and $\delta : Var_c \to \mathbb{Z} \cup \{\bot, \top\}$, $\varphi_l(\delta) \sqsubseteq \delta$ and hence $\Sigma_{\varphi_l(\delta)} \subseteq \Sigma_\delta$ ("filter")

Constant Propagation Analysis with Assertions IV

Remarks:

- For $B^l = (\texttt{assert } b)$ and $\delta : Var_c \to \mathbb{Z} \cup \{\bot, \top\}$, $\varphi_l(\delta) \sqsubseteq \delta$ 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 8.7)

Remarks:

- For $B^l = (\texttt{assert } b)$ and $\delta : Var_c \to \mathbb{Z} \cup \{\bot, \top\}$, $\varphi_l(\delta) \sqsubseteq \delta$ 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 8.7)
- $\varphi_l(\delta)$ can be determined (or at least approximated) by Satisfiability Modulo Theories (SMT) techniques

Remarks:

- For $B^l = (\texttt{assert}\ b)$ and $\delta : Var_c \to \mathbb{Z} \cup \{\bot, \top\}$, $\varphi_l(\delta) \sqsubseteq \delta$ 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 8.7)
- $\varphi_l(\delta)$ can be determined (or at least approximated) by Satisfiability Modulo Theories (SMT) techniques
- If $\mathrm{CP}_l(x) = \bot$ for some $l \in Lab_c$ and $x \in Var_c$, then l is unreachable (and $\mathrm{CP}_l(y) = \bot$ for all $y \in Var_c$)