Static Program Analysis

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

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

Undecidability of the MOP Solution I

Theorem (Undecidability of MOP solution)

The MOP solution for Constant Propagation is undecidable.

Proof.

Based on undecidability of Modified Post Correspondence Problem (MPCP):
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 := u1;y := v1;
while ... do
  if ... then
    x := x ++ u1;
    y := y ++ v1
  else if ... then
    ⋮
  else
    x := x ++ un;
    y := y ++ vn
  end ... end
end;
```

~~if x = y then z := 1 else z := 0 end;~~

```
z := (x = y);
[skip]^l
```

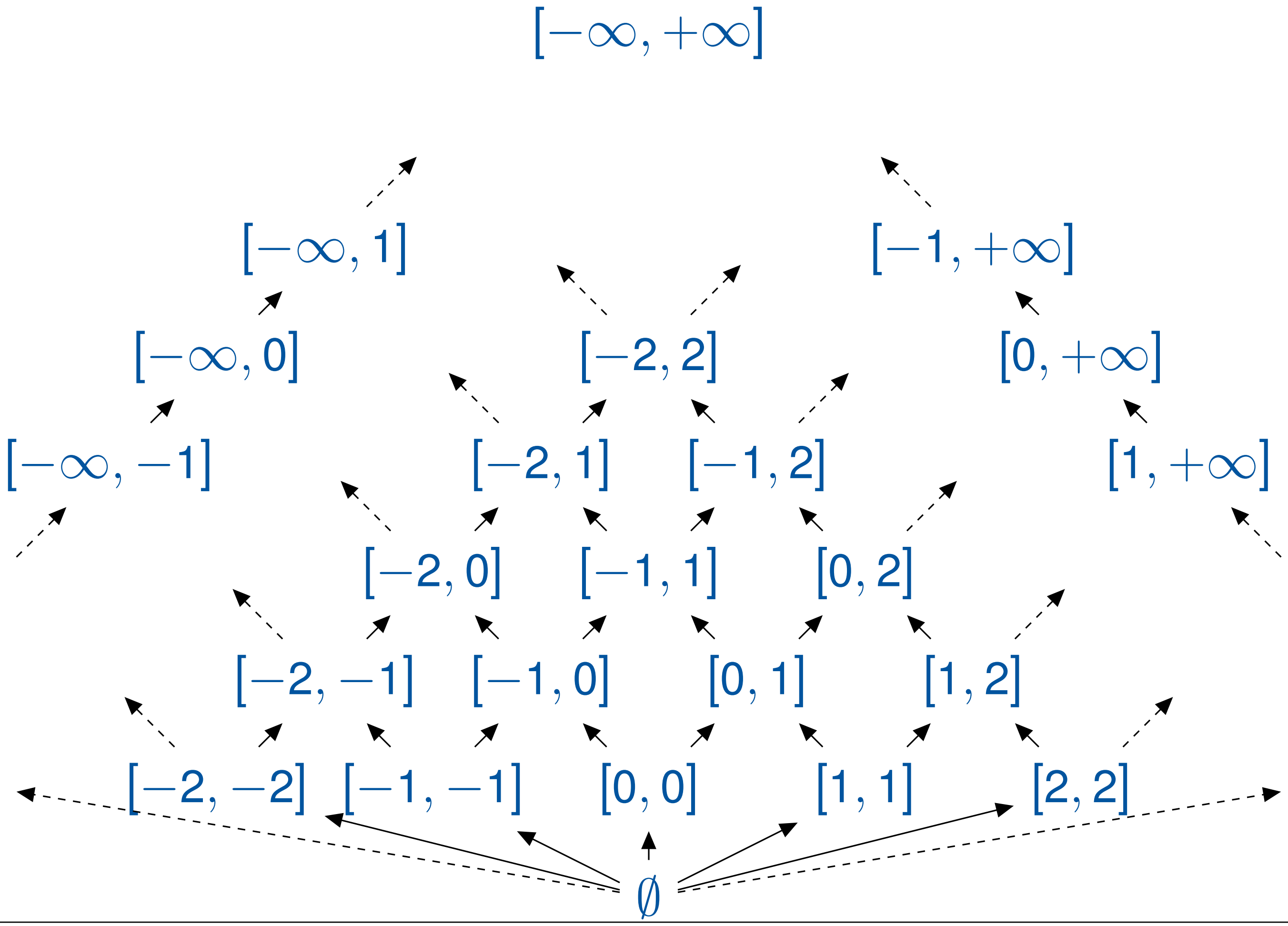
Then: $\mathrm{mop}(l)(\mathtt{z}) = \mathrm{false}$

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

$\iff$ the MPCP has no solution

□

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 $E := \{\text{init}(c)\}$ (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

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

with

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

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

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

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

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

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

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

- 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:

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

- 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 5.1 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** $W := W \cdot (l, l')$; *% Initialise W*

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

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

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

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

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

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

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

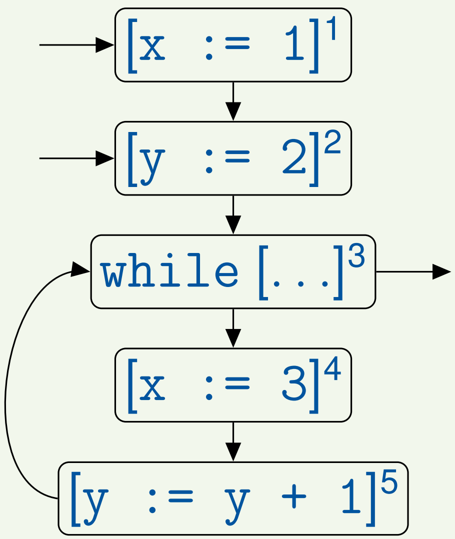
if (l', l'') *not in* W **then** $W := (l', l'') \cdot W$; *% Propagate modification*

Output: $\{\mathrm{AI}_l \mid l \in Lab\}$, *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. 5.4)

Another Widening Example

Example 8.1



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

$$
\begin{aligned}
\varphi_1(J_x, J_y) &= ([1,1], J_y)\\
\varphi_2(J_x, J_y) &= (J_x, [2,2])\\
\varphi_3(J_x, J_y) &= (J_x, J_y)\\
\varphi_4(J_x, J_y) &= ([3,3], J_y)\\
\varphi_5(J_x, \emptyset) &= (J_x, \emptyset)\\
\varphi_5(J_x, [y_1, y_2]) &= (J_x, [y_1 + 1, y_2 + 1])
\end{aligned}
$$

Application of worklist algorithm

1. without widening (omitted): diverges (for $\mathtt{y}$) with expected result for $\mathtt{x}$: $\mathsf{AI}_3(\mathtt{x}) = [1, 3]$
2. with widening (on the board): terminates with unexpected result for $\mathtt{x}$: $\mathsf{AI}_3(\mathtt{x}) = [1, +\infty]$

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 $\mathrm{fix}^\nabla(\Phi_S)$)

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

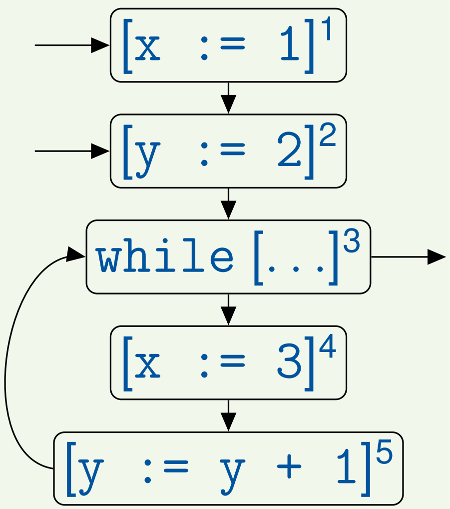
- **Soundness:** $\mathrm{fix}^\nabla(\Phi_S) \sqsupseteq \mathrm{fix}(\Phi_S)$ (cf. Alg. 7.5)

$$\Longrightarrow \Phi_S^k(\mathrm{fix}^\nabla(\Phi_S)) \sqsupseteq \Phi_S^k(\mathrm{fix}(\Phi_S)) = \mathrm{fix}(\Phi_S)$$

(since Φ_S and thus Φ_S^k monotonic)

Narrowing Example

Example 8.2 (cf. Example 8.1)



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

$$
\begin{aligned}
\varphi_1(J_x, J_y) &= ([1,1], J_y)\\
\varphi_2(J_x, J_y) &= (J_x, [2,2])\\
\varphi_3(J_x, J_y) &= (J_x, J_y)\\
\varphi_4(J_x, J_y) &= ([3,3], J_y)\\
\varphi_5(J_x, \emptyset) &= (J_x, \emptyset)\\
\varphi_5(J_x, [y_1, y_2]) &= (J_x, [y_1+1, y_2+1])
\end{aligned}
$$

Narrowing	AI_1	AI_2	AI_3	AI_4	AI_5
$\mathrm{fix}^{\nabla}(\Phi_S)$	$(\top,\top)$	$([1,1],\top)$	$([1,+\infty],[2,+\infty])$	$([1,+\infty],[2,+\infty])$	$([3,3],[2,+\infty])$
$\Phi_S(\mathrm{fix}^{\nabla}(\Phi_S))$	$(\top,\top)$	$([1,1],\top)$	$([1,3],[2,+\infty])$	$([1,+\infty],[2,+\infty])$	$([3,3],[2,+\infty])$
$\Phi_S^2(\mathrm{fix}^{\nabla}(\Phi_S))$	$(\top,\top)$	$([1,1],\top)$	$([1,3],[2,+\infty])$	$([1,3],[2,+\infty])$	$([3,3],[2,+\infty])$
$\Phi_S^3(\mathrm{fix}^{\nabla}(\Phi_S))$	$(\top,\top)$	$([1,1],\top)$	$([1,3],[2,+\infty])$	$([1,3],[2,+\infty])$	$([3,3],[2,+\infty])$

Narrowing in Practice

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

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
  end;
  z := z + x;
  x := x - 1
end;
```

- Interval analysis (with widening) yields for l:

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

- Too pessimistic! In fact,

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

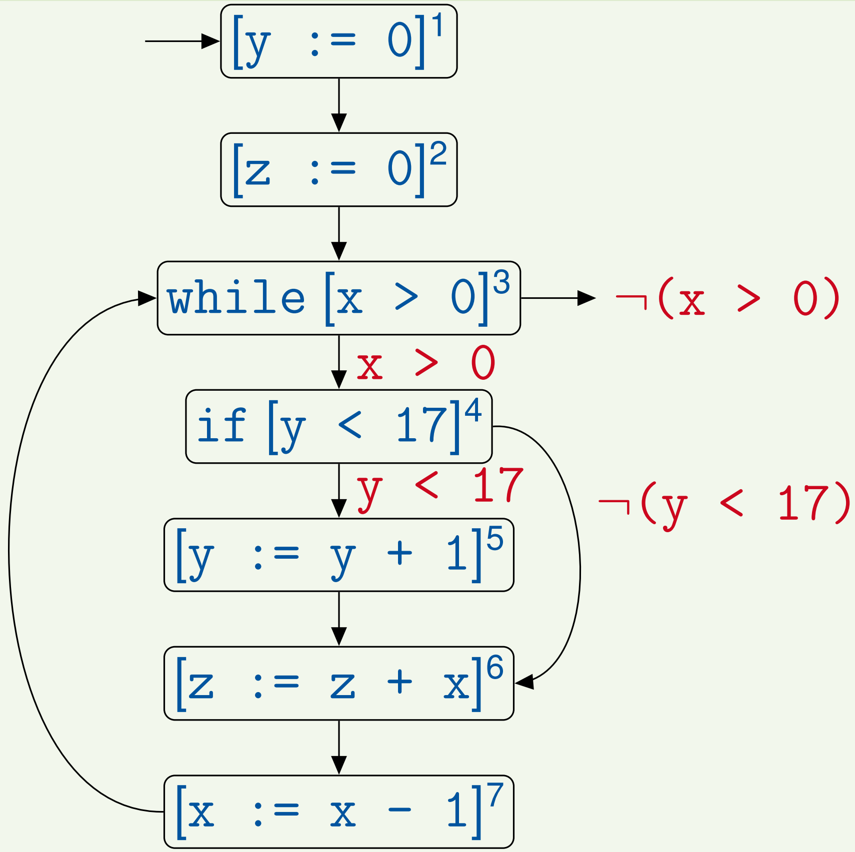
Taking Conditional Branches into Account II

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

Software Modeling and Verification Chair
RWTH AACHEN UNIVERSITY

Conditions as Edge Labels vs. Conditions as Assertions

Example 8.4 (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
  end;
  z := z + x;
  x := x - 1
end;
assert ¬(x > 0);
```

Extending the Syntax of WHILE Programs

Definition 8.5 (Labelled WHILE programs with assertions)

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

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

To be done:

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

Constant Propagation Analysis with Assertions

Original Constant Propagation Analysis

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 (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 $z \in \mathbb{Z}$)
- transfer functions $\{\varphi_l \mid l \in Lab\}$ defined by

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

where

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

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

Transfer Functions of Assertions I

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-states 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 $val_\sigma : BExp \to \mathbb{B}$ for $\sigma : Var_c \to \mathbb{Z}$ is defined by

$$val_\sigma(t) := t$$

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

($val_\sigma : AExp \to \mathbb{Z}$ on previous slide)

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

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

Constant Propagation Analysis with Assertions

Transfer Functions of Assertions II

Example 8.6

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 $b \in \mathit{BExp}$

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

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

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

3. $\mathit{Var}_c = \{\mathtt{x}, \mathtt{y}, \mathtt{z}\}$, $\delta = (\underbrace{1}_{\mathtt{x}}, \underbrace{\top}_{\mathtt{y}}, \underbrace{\top}_{\mathtt{z}})$

   $\implies \Sigma_\delta = \{(1, z_1, z_2) \mid z_1, z_2 \in \mathbb{Z}\} \implies$

   $$\begin{aligned} \varphi_{\mathtt{assert\ x=y}}(\delta) &= (1, 1, \top) \\ \varphi_{\mathtt{assert\ y=z}}(\delta) &= (1, \top, \top) \end{aligned}$$

Constant Propagation Analysis with Assertions

Transfer Functions of Assertions III

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 8.6(3))
- $\varphi_l(\delta)$ 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 $y \in Var_c$)

An Example

Example 8.7

```
if [x = 1]^1 then
  [assert x = 1]^2;
  [y := x + 1]^3
else
  [assert ¬(x = 1)]^4;
  [y := 2]^5
end;
[skip]^6
```

Without assertions	With assertions
$\mathrm{CP}_1 = (\top, \top)$	$\mathrm{CP}_1 = (\top, \top)$
	$\mathrm{CP}_2 = (\top, \top)$
$\mathrm{CP}_3 = (\top, \top)$	$\mathrm{CP}_3 = (1, \top)$
	$\mathrm{CP}_4 = (\top, \top)$
$\mathrm{CP}_5 = (\top, \top)$	$\mathrm{CP}_5 = (\top, \top)$
$\mathrm{CP}_6 = (\top, \top) \sqcup (\top, 2)$	$\mathrm{CP}_6 = (1, 2) \sqcup (\top, 2)$
$= (\top, \top)$	$= (\top, 2)$