Static Program Analysis

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

Summer Semester 2018

Thomas Noll
Software Modeling and Verification Group
RWTH Aachen University

https://moves.rwth-aachen.de/teaching/ss-18/spa/

Software Modeling and Verification Chair

RWTH AACHEN UNIVERSITY

Recap: Widening and Narrowing

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:

- The requirement $d_1 \sqcup d_2 \sqsubseteq d_1 \nabla d_2$ guarantees soundness of widening
- $(d_i^\nabla)_{i \in \mathbb{N}}$ 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 ($d \nabla d = d$)

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:

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

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

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

Recap: Widening and Narrowing

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

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

- **Soundness:** $\text{fix}^{\nabla}(\Phi_S) \sqsupseteq \text{fix}(\Phi_S)$ (cf. Algorithm 7.3)

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

(since Φ_S and thus Φ_S^k monotonic)

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 8.1

```
y := 0;
z := 0;
while [x > 0]^3 do
  if y < 17 then
    y := y + 1
  end;
  z := z + x;
  x := x - 1
end;
```

- Interval analysis (with widening) yields for label 3:

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

- Too pessimistic! In fact,

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

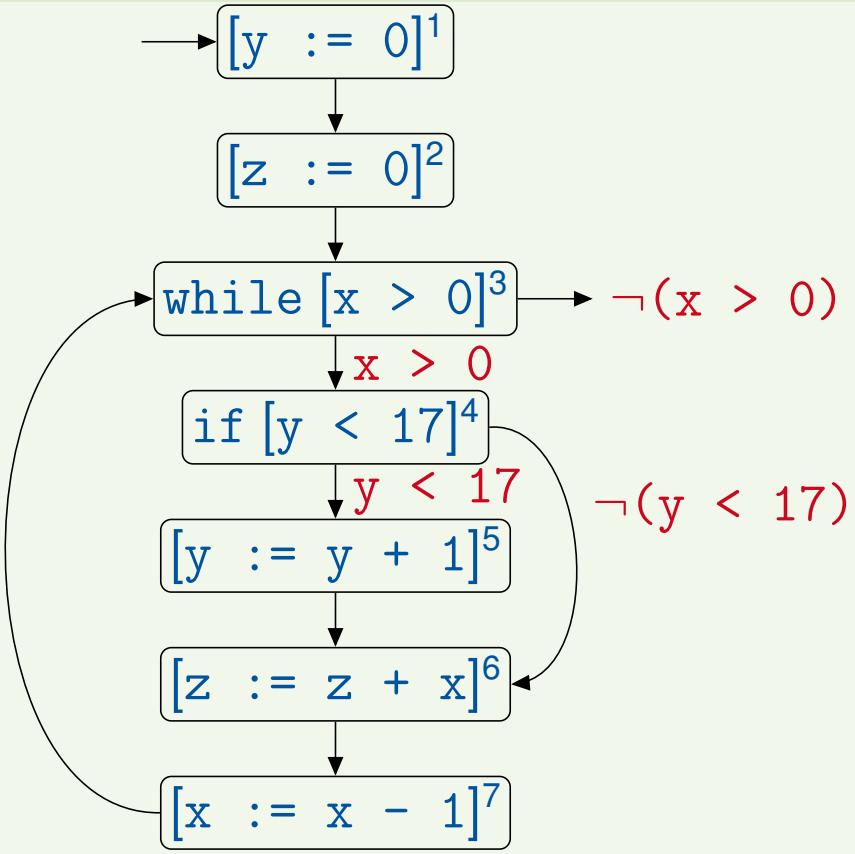
Taking Conditional Branches into Account II

- **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

Taking Conditional Branches into Account

Conditions as Edge Labels vs. Conditions as Assertions

Example 8.2 (cf. Example 8.1)




```
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);
```

Taking Conditional Branches into Account

Extending the Syntax of WHILE Programs

Definition 8.3 (Labelled WHILE programs with assertions)

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

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

To be done:

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

Original Constant Propagation Analysis

- 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

$$val_\delta(x) := \delta(x) \qquad val_\delta(z) := z$$

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

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

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

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

$$val_\sigma(a_1{=}a_2) := (val_\sigma(a_1) = val_\sigma(a_2)) \qquad val_\sigma(b_1 \wedge b_2) := \begin{cases} \text{true} & \text{if } val_\sigma(b_1) = val_\sigma(b_2) = \text{true} \\ \text{false} & \text{otherwise} \end{cases}$$

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

Conditional Constant Propagation Analysis

Transfer Functions of Assertions II

Example 8.4

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

   $\implies \Sigma_\delta = \{(1, 2, z) \mid z \in \mathbb{Z}\} \implies$
   $\varphi_{\mathtt{assert\ x=y}}(\delta) = (\bot, \bot, \bot)$
   $\varphi_{\mathtt{assert\ y=z}}(\delta) = (1, 2, 2)$
   $\varphi_{\mathtt{assert\ y<z}}(\delta) = (1, 2, \top)$
   $\varphi_{\mathtt{assert\ x<=z \wedge y>z}}(\delta) = (1, 2, 1)$

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

   $\implies \Sigma_\delta = \{(1, z_1, z_2) \mid z_1, z_2 \in \mathbb{Z}\} \implies$ $\varphi_{\mathtt{assert\ x=y}}(\delta) = (1, 1, \top)$
   $\varphi_{\mathtt{assert\ y=z}}(\delta) = (1, \top, \top)$

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 8.4(2)/(3))
- $\varphi_l(\delta)$ can be determined (or at least approximated) by Satisfiability Modulo Theories (SMT) techniques
- If $CP_l(x) = \bot$ for some $l \in Lab_c$ and $x \in Var_c$, then l is unreachable (and $CP_l(y) = \bot$ for all $y \in Var_c$)

Conditional Constant Propagation Analysis

An Example

Example 8.5

```
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
$CP_1 = (\top, \top)$	$CP_1 = (\top, \top)$
	$CP_2 = (\top, \top)$
$CP_3 = (\top, \top)$	$CP_3 = (1, \top)$
	$CP_4 = (\top, \top)$
$CP_5 = (\top, \top)$	$CP_5 = (\top, \top)$
$CP_6 = (\top, \top) \sqcup (\top, 2)$	$CP_6 = (1, 2) \sqcup (\top, 2)$
$= (\top, \top)$	$= (\top, 2)$

Original Interval Analysis

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

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

Transfer Functions of Assertions I

Additionally for $B^l = (\texttt{assert}\ b)$, $\delta : Var_c \to Int$ and $x \in Var_c$:

$$\varphi_l(\delta)(x) := \begin{cases} \emptyset & \text{if } Z = \emptyset \\ \left[\bigsqcap_{\mathbb{Z}\cup\{-\infty\}} Z, \bigsqcup_{\mathbb{Z}\cup\{+\infty\}} Z\right] & \text{otherwise} \end{cases}$$

where

- $Z := \{\sigma(x) \mid \sigma \in \Sigma_\delta, val_\sigma(b) = \text{true}\}$
- $\Sigma_\delta := \{\sigma : Var_c \to \mathbb{Z} \mid \forall y \in Var_c : \sigma(y) \in \delta(y)\}$
  (and thus $\Sigma_\delta = \emptyset$ iff $\delta(y) = \emptyset$ for some $y \in Var_c$)
- $val_\sigma : BExp \to \mathbb{B}$: see Slide 8.13

Conditional Interval Analysis

Transfer Functions of Assertions II

Example 8.6

$Var_c = \{\mathtt{x}, \mathtt{y}\}, \delta = (\underbrace{[-\infty, 2]}_{\mathtt{x}}, \underbrace{[0, +\infty]}_{\mathtt{y}})$

$$
\begin{aligned}
\Longrightarrow \quad & \varphi_{\texttt{assert x>0}}(\delta) = ([1, 2], [0, +\infty]) \\
& \varphi_{\texttt{assert x=y}}(\delta) = ([0, 2], [0, 2]) \\
& \varphi_{\texttt{assert x>y}}(\delta) = ([1, 2], [0, 1]) \\
& \varphi_{\texttt{assert x<y}}(\delta) = ([-\infty, 2], [0, +\infty])
\end{aligned}
$$

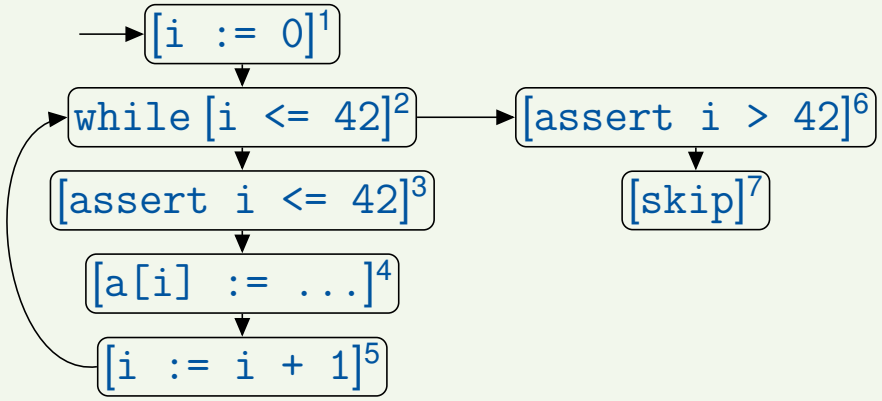
Remarks:

- Again for $B^l = (\texttt{assert}\ b)$ and $\delta : Var_c \to Int$, $\varphi_l(\delta) \sqsubseteq \delta$ and hence $\Sigma_{\varphi_l(\delta)} \subseteq \Sigma_\delta$ ("filter")
- Again if $\mathsf{AI}_l(x) = \emptyset$ for some $l \in Lab_c$ and $x \in Var_c$, then l is unreachable (and $\mathsf{AI}_l(y) = \emptyset$ for all $y \in Var_c$)

Conditional Interval Analysis

An Example

Example 8.7 (Interval analysis for array index; cf. Example 6.7)



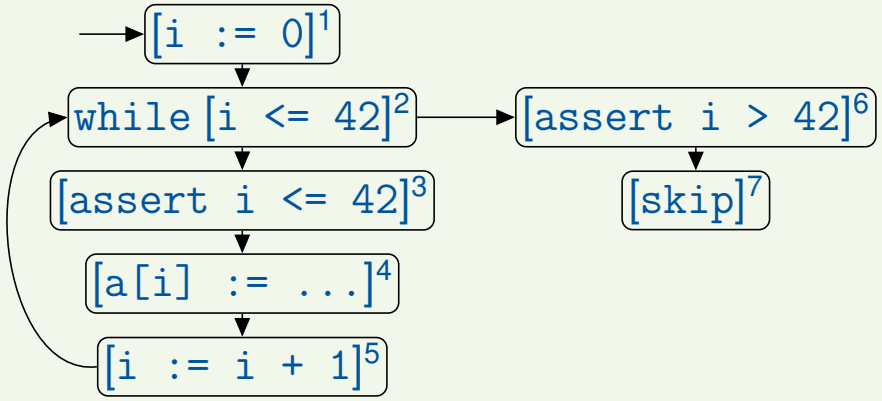
$$\varphi_1(J) = [0,0]$$
$$\varphi_2(J) = J$$
$$\varphi_3(J) = J \cap [-\infty, 42]$$
$$\varphi_4(J) = J$$
$$\varphi_5(\emptyset) = \emptyset$$
$$\varphi_5([i_1, i_2]) = [i_1 + 1, i_2 + 1]$$
$$\varphi_6(J) = J \cap [43, +\infty]$$

Worklist w. widening	AI_1	AI_2	AI_3	AI_4	AI_5	AI_6	AI_7
12, 23, 34, 45, 52, 26, 67	$[-\infty, +\infty]$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
23, 34, 45, 52, 26, 67	$[-\infty, +\infty]$	$[0,0]$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
34, 45, 52, 26, 67	$[-\infty, +\infty]$	$[0,0]$	$[0,0]$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
45, 52, 26, 67	$[-\infty, +\infty]$	$[0,0]$	$[0,0]$	$[0,0]$	$\emptyset$	$\emptyset$	$\emptyset$
52, 26, 67	$[-\infty, +\infty]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$\emptyset$	$\emptyset$
23, 26, 67	$[-\infty, +\infty]$	$[0, +\infty]$	$[0,0]$	$[0,0]$	$[0,0]$	$\emptyset$	$\emptyset$
34, 26, 67	$[-\infty, +\infty]$	$[0, +\infty]$	$[0, +\infty]$	$[0,0]$	$[0,0]$	$\emptyset$	$\emptyset$
45, 26, 67	$[-\infty, +\infty]$	$[0, +\infty]$	$[0, +\infty]$	$[0, +\infty]$	$[0,0]$	$\emptyset$	$\emptyset$
52, 26, 67	$[-\infty, +\infty]$	$[0, +\infty]$	$[0, +\infty]$	$[0, +\infty]$	$[0, +\infty]$	$\emptyset$	$\emptyset$
26, 67	$[-\infty, +\infty]$	$[0, +\infty]$	$[0, +\infty]$	$[0, +\infty]$	$[0, +\infty]$	$\emptyset$	$\emptyset$
67	$[-\infty, +\infty]$	$[0, +\infty]$	$[0, +\infty]$	$[0, +\infty]$	$[0, +\infty]$	$[0, +\infty]$	$\emptyset$

Conditional Interval Analysis

An Example (continued)

Example 8.7 (Interval analysis for array index; continued)



$$\varphi_1(J) = [0,0]$$
$$\varphi_2(J) = J$$
$$\varphi_3(J) = J \cap [-\infty, 42]$$
$$\varphi_4(J) = J$$
$$\varphi_5(\emptyset) = \emptyset$$
$$\varphi_5([i_1, i_2]) = [i_1 + 1, i_2 + 1]$$
$$\varphi_6(J) = J \cap [43, +\infty]$$

Narrowing	AI_1	AI_2	AI_3	AI_4	AI_5	AI_6	AI_7
$\text{fix}^{\nabla}(\Phi_S)$	$[-\infty,+\infty]$	$[0,+\infty]$	$[0,+\infty]$	$[0,+\infty]$	$[0,+\infty]$	$[0,+\infty]$	$[43,+\infty]$
$\Phi_S(\text{fix}^{\nabla}(\Phi_S))$	$[-\infty,+\infty]$	$[0,+\infty]$	$[0,+\infty]$	$[0,42]$	$[0,+\infty]$	$[0,+\infty]$	$[43,+\infty]$
$\Phi_S^2(\text{fix}^{\nabla}(\Phi_S))$	$[-\infty,+\infty]$	$[0,+\infty]$	$[0,+\infty]$	$[0,42]$	$[0,42]$	$[0,+\infty]$	$[43,+\infty]$
$\Phi_S^3(\text{fix}^{\nabla}(\Phi_S))$	$[-\infty,+\infty]$	$[0,43]$	$[0,+\infty]$	$[0,42]$	$[0,42]$	$[0,+\infty]$	$[43,+\infty]$
$\Phi_S^4(\text{fix}^{\nabla}(\Phi_S))$	$[-\infty,+\infty]$	$[0,43]$	$[0,43]$	$[0,42]$	$[0,42]$	$[0,43]$	$[43,+\infty]$
$\Phi_S^5(\text{fix}^{\nabla}(\Phi_S))$	$[-\infty,+\infty]$	$[0,43]$	$[0,43]$	$[0,42]$	$[0,42]$	$[0,43]$	$[43,43]$
$\Phi_S^6(\text{fix}^{\nabla}(\Phi_S))$	$[-\infty,+\infty]$	$[0,43]$	$[0,43]$	$[0,42]$	$[0,42]$	$[0,43]$	$[43,43]$

Optimisation in the GNU C Compiler

Live Variables

Source:

```
int f() {
  int a;
  a = 1;
  a = 2;
  return a;
}
```

W/o optimisation:

```
        ...
_f:
## @f
        .cfi_startproc
## BB#0:
        pushq   %rbp
Ltmp0:
        .cfi_def_cfa_offset 16
Ltmp1:
        .cfi_offset %rbp, 16
        movq    %rsp, %rbp
Ltmp2:
        .cfi_def_cfa_register %rbp
        movl    $1, 4(%rbp)
        movl    $2, 4(%rbp)
        movl    4(%rbp), %eax
        popq    %rbp
        retq
        .cfi_endproc
        ...
```

With -O3:

```
        ...
_f:
## @f
        .cfi_startproc
## BB#0:
        pushq   %rbp
Ltmp0:
        .cfi_def_cfa_offset 16
Ltmp1:
        .cfi_offset %rbp, 16
        movq    %rsp, %rbp
Ltmp2:
        .cfi_def_cfa_register %rbp

        movl    $2, %eax

        popq    %rbp
        retq
        .cfi_endproc
        ...
```

Available Expressions

Source:

```
int f(int a,int b){
  int c;
  c = a*b;
  if (c > 0)
    {return a*b+1;}
  else
    {return a*b+2;}
}
```

W/o optimisation:

```
        ...
_f:                                     ## @f
        .cfi_startproc
## BB#0:
        pushq   %rbp
Ltmp0:
        .cfi_def_cfa_offset 16
Ltmp1:
        .cfi_offset %rbp, 16
        movq    %rsp, %rbp
Ltmp2:
        .cfi_def_cfa_register %rbp
        movl    %edi, 8(%rbp)
        movl    %esi, 12(%rbp)
        movl    8(%rbp), %esi
        imull   12(%rbp), %esi  ; a*b
        movl    %esi, 16(%rbp)  ; store c
        cmpl    $0, 16(%rbp)    ; c > 0?
        jle     LBB0_2
## BB#1:
        movl    8(%rbp), %eax
        imull   12(%rbp), %eax  ; a*b
        addl    $1, %eax        ; add 1
        movl    %eax, 4(%rbp)
        jmp     LBB0_3
LBB0_2:
        movl    8(%rbp), %eax
        imull   12(%rbp), %eax  ; a*b
        addl    $2, %eax        ; add 2
        movl    %eax, 4(%rbp)
LBB0_3:
        movl    4(%rbp), %eax
        popq    %rbp
        retq
        .cfi_endproc
        ...
```

With -O3:

```
        ...
_f:                                     ## @f
        .cfi_startproc
## BB#0:
        pushq   %rbp
Ltmp0:
        .cfi_def_cfa_offset 16
Ltmp1:
        .cfi_offset %rbp, 16
        movq    %rsp, %rbp
Ltmp2:
        .cfi_def_cfa_register %rbp
        imull   %esi, %edi          ; a*b
        testl   %edi, %edi          ; a*b = 0?
        setle   %al
        movzbl  %al, %eax
        leal    1(%rax,%rdi), %eax
        popq    %rbp
        retq
        .cfi_endproc
        ...
```