Introduction to static analysis #5

Seminar @ Gondow Lab.

Overview

- Sparse Analysis
  - Spatial Sparsity
  - Temporal Sparsity
- Modular Analysis
- Backward Analysis

Sparse Analysis (1/2)

We can reduce the cost of the analysis by considering ***sparsity***.

- Spatial sparsity
- Temporal sparsity

By exploiting these, we can improve the scalability of the analysis (we call this ***sparse analysis***).

Sparse Analysis (2/2)

Sparse analysis is independent of its underlying analysis.

That is,

1. Design a sound analysis
2. Add sparse analysis to improve its scalability
    - its precision is preserved

Overview

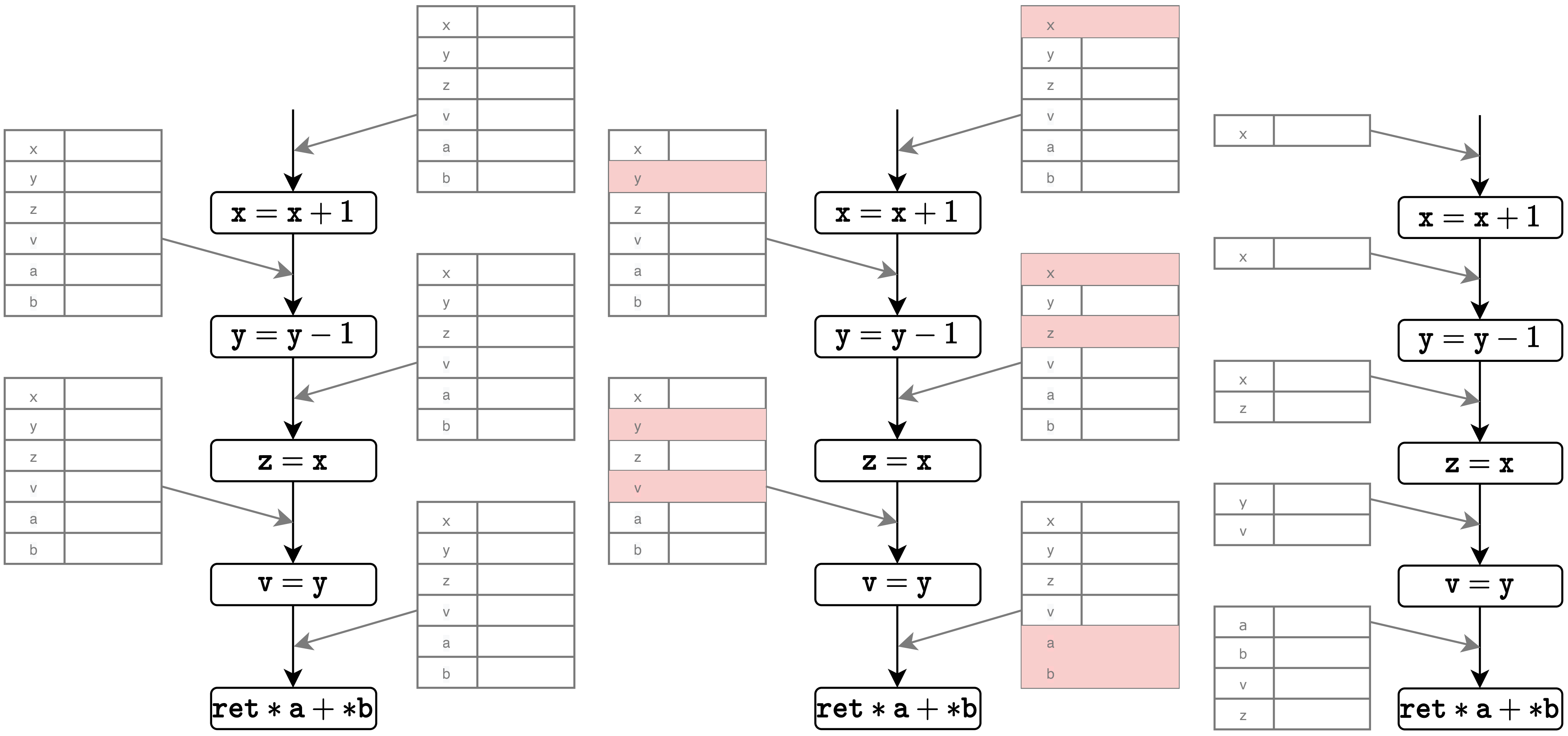
- Sparse Analysis
  - Spatial Sparsity
  - Temporal Sparsity
- Modular Analysis
- Backward Analysis

What is spatial sparsity ?

We will consider this c-like program.

```
x = x + 1;
y = y - 1;
z = x;
v = y;
ret *a + *b;
```

What is spatial sparsity ?



We need only red part.

Spatial Sparsity (1/3)

Notation:

- $dom(M^\sharp) : \mathbb{M}^\sharp \to \wp(\mathbb{X})$
  - entries of $M^\sharp$
- $Access^\sharp(l) : \mathbb{L} \to \wp(\mathbb{X})$
  - set of abstract locations that may be accessed by the program in label l

Spatial Sparsity (2/3)

The abstract semantics function

$$F^\sharp : (\mathbb{L} \to \mathbb{M}^\sharp) \to (\mathbb{L} \to \mathbb{M}^\sharp)$$

becomes

$$F^\sharp_{sparse} : (\mathbb{L} \to \mathbb{M}^\sharp_{sparse}) \to (\mathbb{L} \to \mathbb{M}^\sharp_{sparse})$$

where

$$\mathbb{M}^\sharp_{sparse} = \{M^\sharp \in \mathbb{M}^\sharp \mid dom(M^\sharp) = Access^\sharp(l), l \in \mathbb{L}\} \cup \{\bot\}$$

- $\mathbb{M}^\sharp_{sparse}$: メモリ状態から、アクセスされ得ないものを削除したもの

Spatial Sparsity (3/3)

Then, when $Access^\sharp(\cdot)$ is computed?

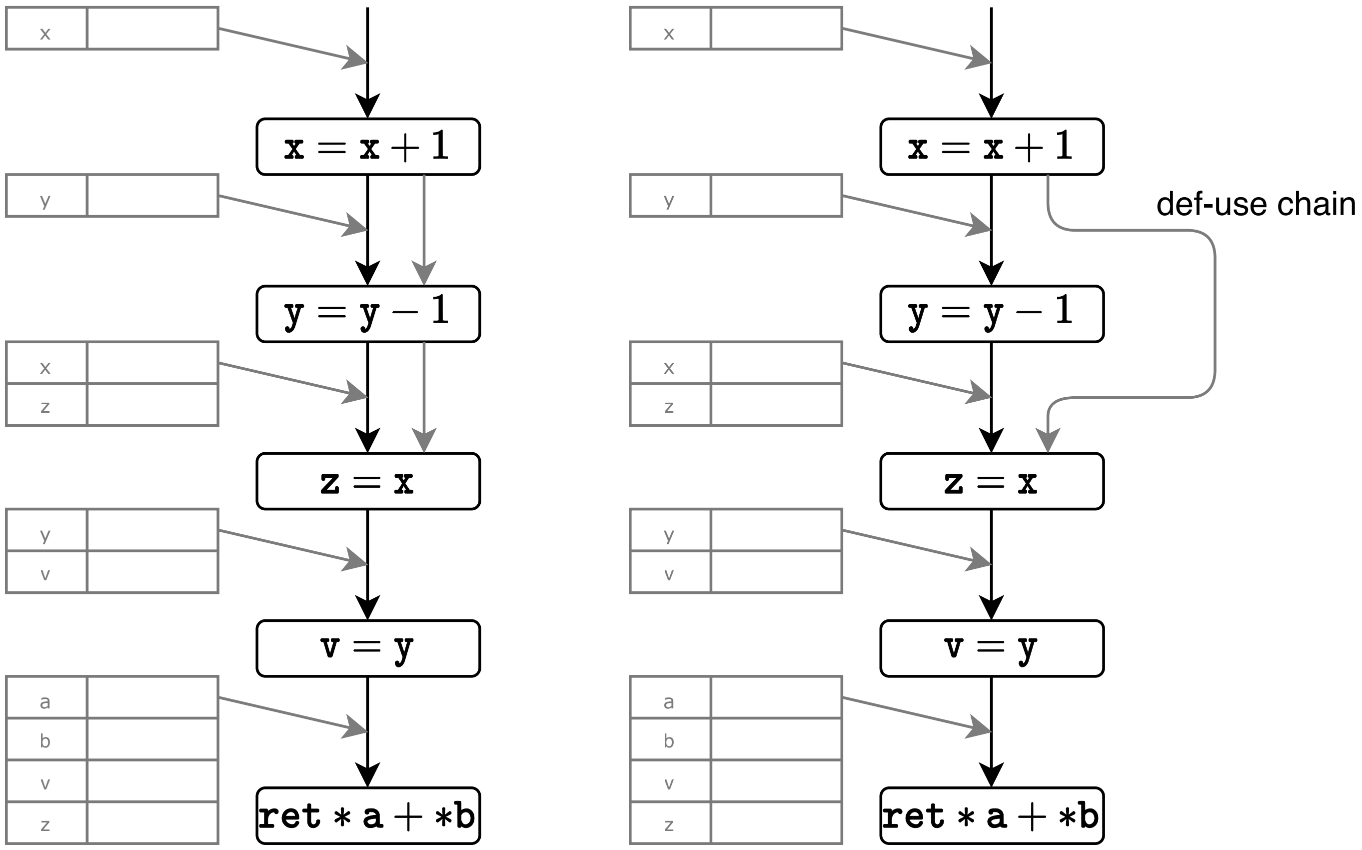
- $\rightarrow$ before the main analysis starts (so called *pre-analysis*)
  - pre-analysis : typically coarser, hence quicker yet sound analysis

Overview

- Sparse Analysis
  - Spatial Sparsity
  - Temporal Sparsity
- Modular Analysis
- Backward Analysis

Temporal Sparsity

- I show only the case of $\mathtt{x}$.
- Statements where defined variables are not used can be skipped.
  - in this case the second statement
- Such flow is called ***def-use chain***



(a) Blindly following the semantic control flow

(b) Directly following the def-use chain

Temporal Sparsity

Once the def-use chain is available, temporal sparsity analysis is defined as follow:

$$(l, M^\sharp) \hookrightarrow^\sharp (l', M'^\sharp) \ \text{ for } \ l' \in \mathtt{\textbf{next}}^\sharp(l, M^\sharp)$$

Then, $\hookrightarrow^\sharp$ become sparse.

$\mathtt{\textbf{next}}^\sharp(l, M^\sharp)$ determines the def-use relation from where point l to its use point l'.

Precision-Preserving Def-Use Chain

Definition 5.4 (Safe def and use sets from pre-analysis)

- $D^\sharp(l)$: sets of abstract locations
- $U^\sharp(l)$: sets of abstract locations

$D^\sharp_{pre}$ and $U^\sharp_{pre}$ are those that are computed by the pre-analysis.

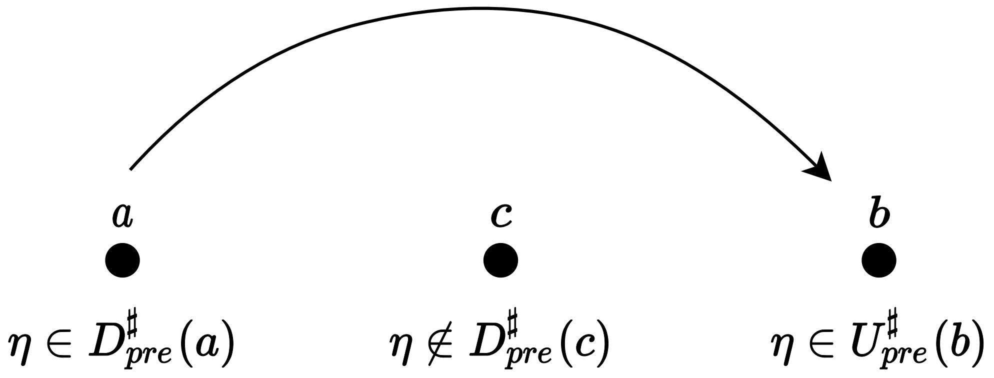
- $D^\sharp_{pre}$ and $U^\sharp_{pre}$ are *safe* whenever
  - $\forall l \in \mathbb{L} : D^\sharp_{pre}(l) \supseteq D^\sharp(l)$ and $\forall l \in \mathbb{L} : U^\sharp_{pre}(l) \supseteq U^\sharp(l)$
    - over-approximate non-sparse analysis
  - $\forall l \in \mathbb{L} : U^\sharp_{pre}(l) \supseteq D^\sharp_{pre}(l) \setminus D^\sharp(l)$
    - this will be explained later

Precision-Preserving Def-Use Chain

Definition 5.5 (Def-use chain information from pre-analysis)

We define $D^\sharp_{pre}$ and $U^\sharp_{pre}$ as in definition 5.4.

- label a and b have a *def-use chain* for abstract location η whenever
  - for every label c in the execution paths from a to b
    - $\eta \notin D^\sharp_{pre}(c)$

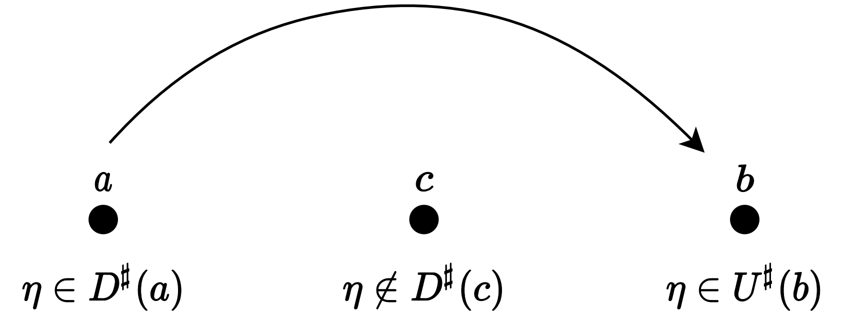


Precision-Preserving Def-Use Chain

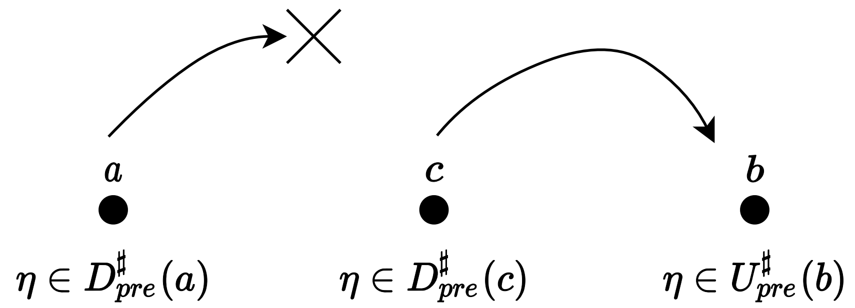
- Why is the second condition in def 5.4 needed to be safe?

> - $\forall l \in \mathbb{L} : U^\sharp_{pre} \supseteq D^\sharp_{pre}(l) D^\sharp(l)$

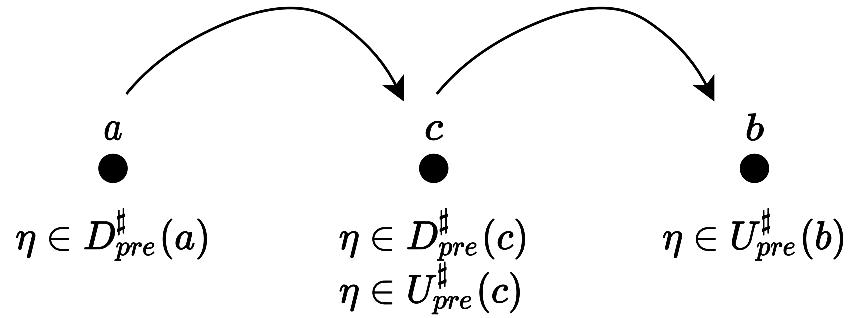
- $\rightarrow$ To preserve the original "flow".



(a) Original analysis def-use cahin for η



(b) Missing def-use edge (a to b)



(b) Recovered def-use edge (a to b)

Overview

- Sparse Analysis
- Modular Analysis
  - Introduction
  - Case study
- Backward Analysis

Modular Analysis

- Unit of modular analysis : procedure (function)
- analyze each of them, and then, links them together and get the whole-program analysis result.

merit:

- incremental analysis
- improvement of precision
- need to recompute only the analysis of a procedure when it is modified.

Parameterization

Consider a interval analysis $[l, h]$.

1. parameterize the calling context
    - in this case symbolize the lower and upper bound of interval
2. compute the post-state in terms of symbolic parameters
3. at link time, instantiate pre- and post-state
4. ex) check whether the no-buffer-overrun conditions are violated

Summary-Based

- Modular analysis compute what a procedure does (***summary***).

> When we resolve the symbolic safe conditions to be violated, an alarm is raised.

Scalability

- When a procedure is modified:
  - the whole-program analysis result is quickly obtained by updating only the result of modified condition.
- This analysis make the whole analysis scalable.

Overview

- Sparse Analysis
- Modular Analysis
  - Introduction
  - Case study
- Backward Analysis

Case Study (1/3)

Goal:

- estimate the sizes of buffers and the ranges of their indexing expressions

Fist example:

```c
void set_i(int *arr, int index) {
  arr[index] = 0;
}
```

Parametric context is:

$$\mathtt{arr} \mapsto (\text{offset} : [\mathtt{s}_0, \mathtt{s}_1], \text{size} : [\mathtt{s}_2, \mathtt{s}_3])$$

$$\mathtt{index} \mapsto [\mathtt{s}_4, \mathtt{s}_5]$$

Safe condition is:

$$[\mathtt{s}_0 + \mathtt{s}_4, \mathtt{s}_1 + \mathtt{s}_5] < [\mathtt{s}_2, \mathtt{s}_3]$$

Case Study (2/3)

Second example

```c
char * malloc_wrapper(int n) {
  return malloc(n);
}
```

Symbolic procedure summary would be:

$$\mathbf{n} \mapsto [\mathbf{s}_6, \mathbf{s}_7]$$

$$\mathbf{ret} \mapsto (\text{offset} : [0, 0], \text{size} : [\mathbf{s}_6, \mathbf{s}_7])$$

Case Study (3/3)

```c
void interprocedural() {
  int *arr = malloc_wrapper(9*sizeof(int));
  // arr -> (offset:[0,0], size:[9,9])
  int i;
  for ( i = 0; i < 9; i+=1 ) {
    // i -> [0,8]
    set_i(arr, i);        // safe
    set_i(arr, i + 1);  // alarm
  }
}
```

For the first $\mathbf{set\_i}(\mathbf{arr}, \mathbf{i})$ call, the safety condition is:

$$[0+0, 0+8] < [9, 9]$$

For the second $\mathbf{set\_i}(\mathbf{arr}, \mathbf{i}+1)$ call, the safety condition is:

$$[0+0, 0+9] < [9, 9]$$

This condition is false, hence alarm.

Overview

- Sparse Analysis
- Modular Analysis
- Backward Analysis
    - Forward vs Backward
    - Backward Analysis and Applications
    - Definition of Backward Analysis
    - Precision Refinement

Forward vs Backward

Let's over-approximate pre-condition from a post-condition.

Recall the filtering function $\mathscr{F}_{\mathtt{B}}$ from chapter 3,

$$\mathscr{F}_{\mathtt{B}}(M) = \{m \in M \mid [\![\mathtt{B}]\!](m) = \mathtt{true}\}$$

We can define $[\![B]\!]_{\mathbf{bwd}}$ and define $\mathscr{F}_{\mathtt{B}}$ from it:

$$[\![\mathtt{B}]\!]_{\mathbf{bwd}}(v) = \{m \in \mathbb{M} \mid [\![\mathtt{B}]\!](m) = v\}$$
$$\mathscr{F}_{\mathtt{B}}(M) = M \cap [\![B]\!]_{\mathbf{bwd}}(\mathtt{true})$$

- $[\![\mathtt{B}]\!]_{\mathbf{bwd}}$ is backward style.
  - input : value
  - output : set of states that lead to the input value

Forward vs Backward

We define backward semantics as follow:

$$\begin{aligned}\llbracket C\rrbracket_{\mathbf{bwd}}(M) &= \{m \in \mathbb{M} \mid \exists m' \in \llbracket C\rrbracket(\{m\}), m' \in M\} \\ &= \{m \in \mathbb{M} \mid \llbracket C\rrbracket(\{m\}) \cap M \neq \emptyset\}\end{aligned}$$

Intuitive explanation

- input : a set of states M
- output : a set of states that may lead to some of M by executing C

Overview

- Sparse Analysis
- Modular Analysis
- Backward Analysis
  - Forward vs Backward
  - Backward Analysis and Applications
  - Definition of Backward Analysis
  - Precision Refinement

Backward analysis and Applications (1/4)

```
int x0, x1;
input(x0);
if (x0 > 0) {
  x1 := x0;
} else {
  x1 := -x0;
}
```

-

Q. The result of the analysis of chapter 3 is: $(\{x_0 \mapsto ??, x_1 \mapsto ??\})$

Backward analysis and Applications (2/4)

```
int x0, x1;
input(x0);
if (x0 > 0) {
  x1 := x0;
} else {
  x1 := -x0;
}
```

-

Q. $[\![C]\!]_{\mathbf{bwd}}$ maps $2 \leq \mathbf{x}_1 \leq 5$ to ...

Backward analysis and Applications (3/4)

```
int x0, x1;
input(x0);
if (x0 > 0) {
  x1 := x0;
} else {
  x1 := -x0;
}
```

-

Q. $[\![C]\!]_{\mathbf{bwd}}$ maps $\mathbf{x_1 \leq -3}$ to ...

Backward analysis and Applications (4/4)

Use case:

- Provide *necessary condition* for a specific behavior to occur
  - = provide *sufficient condition* for a specific behavior not to occur
- Program understanding
- Precision Refinement

Overview

- Sparse Analysis
- Modular Analysis
- Backward Analysis
    - Forward vs Backward
    - Backward Analysis and Applications
    - Definition of Backward Analysis
    - Precision Refinement

Definition of Backward Analysis (1/2)

- $[\![\mathbf{skip}]\!]^\sharp_{\mathbf{bwd}}(M^\sharp) = M^\sharp$
- $[\![C_0; C_1]\!]^\sharp_{\mathbf{bwd}}(M^\sharp) = M^\sharp$
- $[\![\mathbf{if}(B)\{C_0\}\mathbf{else}\{C_1\}]\!]^\sharp_{\mathbf{bwd}}(M^\sharp) = \mathscr{F}^\sharp_B([\![C_0]\!]^\sharp_{\mathbf{bwd}}) \sqcup^\sharp \mathscr{F}^\sharp_{\neg B}([\![C_1]\!]^\sharp_{\mathbf{bwd}})$
- $[\![\mathbf{while}(B)\{C\}]\!]^\sharp_{\mathbf{bwd}}(M^\sharp) = \texttt{abs\_iter}(\mathscr{F}^\sharp_B \circ [\![C]\!]^\sharp_{\mathbf{bwd}}) \circ \mathscr{F}_{\neg B}(M^\sharp)$

Definition of Backward Analysis (2/2)

- expression $\mathbf{x} := \mathbf{E}$
  - if $\mathbf{x}$ appears in $\mathbf{E}$ ("non-invertible")
    - apply $\mathscr{F}^{\sharp}_{\mathtt{x}=\mathtt{E}}$ and then forget all the constraints over $\mathbf{x}$
  - if $\mathbf{x}$ does not appears in $\mathbf{E}$ ("invertible")
    - such as $\mathbf{x} = \mathbf{x} + 1$
    - derive pre-condition from post-condition
      - post-condition : $\{\mathbf{x} \mapsto [3, 9]\}$
      - pre-condition : $\{\mathbf{x} \mapsto [2, 8]\}$

Overview

- Sparse Analysis
- Modular Analysis
- Backward Analysis
  - Forward vs Backward
  - Backward Analysis and Applications
  - Definition of Backward Analysis
  - Precision Refinement

Precision Refinement

```
..  // --------------------- 1
if (y <= x) {
  ... // ------------------- 2
    if (x <= 4) {
      ... // --------------- 3
        if (5 <= y) {
          ... // ----------- 4
        }
    }
}
```

Note:

- third true branch is not *feasible*

Precision Refinement (1/3)

Forward analysis using intervals or polyhedra.

Intervals

1. $\{\mathtt{x} \mapsto \top, \mathtt{y} \mapsto \top\}$
2. $\{\mathtt{x} \mapsto \top, \mathtt{y} \mapsto \top\}$
3. $\{\mathtt{x} \mapsto (-\infty, 4], \mathtt{y} \mapsto \top\}$
4. $\{\mathtt{x} \mapsto (-\infty, 4], \mathtt{y} \mapsto [5, +\infty]\}$

Polyhedra

1. $\top$
2. $\mathtt{y} \leq \mathtt{x}$
3. $\mathtt{y} \leq \mathtt{x} \wedge \mathtt{x} \leq 4$
4. $\bot$

Precision Refinement (2/3)

Backward analysis using the result of fist forward analysis:

1. $\perp$ $\uparrow$ end
2. $\{\mathbf{x} \mapsto (-\infty, 4], \mathbf{y} \mapsto [5, +\infty)\}$ $\uparrow$
3. $\{\mathbf{x} \mapsto (-\infty, 4], \mathbf{y} \mapsto [5, +\infty)\}$ $\uparrow$
4. $\{\mathbf{x} \mapsto (-\infty, 4], \mathbf{y} \mapsto [5, +\infty)\}$ $\uparrow$ start

Again, forward analysis

1. $\perp$ $\downarrow$ start
2. $\perp$ $\downarrow$
3. $\perp$ $\downarrow$
4. $\perp$ $\downarrow$ end, not feasible

Precision Refinement (3/3)

- This forward-backward iteration can be iterated as many times as required.
- Backward analysis itself might be imprecise, however,
    - it can improve preciseness if used along with forward analysis.

Summary

- Sparse Analysis
  - addresses scalability
- Modular Analysis
  - addresses scalability
- Backward Analysis
  - addresses preciseness