Abstract-Value Slicing

(Goal-Directed Weakening of Abstract Interpretation Results)

Sunae Seo

2008. 1. 31

SIGPL winter school

Contents

Problem

- AI results are often unnecessarily informative
  - AI computes program invariants as strong as possible.
  - Verification of a program usually does not require the whole AI results.
  - Our experiments show that 63%-84% of the results were not needed for the intended verification.
  - Constructing a compact program proof is tackled by those big AI results.

Simple Example:
Parity Analysis

- code: `x:=1; y:=2x`
- AI results:

$$\boxed{x{\to}\top,\ y{\to}\top}\ \texttt{x:=1;}\ \boxed{x{\to}\text{odd},\ y{\to}\top}\ \texttt{y:=2x}\ \boxed{x{\to}\text{odd},\ y{\to}\text{even}}$$

- Verification goal: `y` is even at the end
- Proof goal: $\dfrac{\vdots}{\{true\}\texttt{x:=1; y:=2x}\{\exists n.\, y = 2n\}}$
- Proof candidates:

$$\dfrac{\dfrac{\nabla_1}{\{true\}\texttt{x:=1}\{\exists m.x{=}2m{+}1\}} \quad \dfrac{\nabla_2}{\{\exists m.x{=}2m{+}1\}\texttt{y:=2x}\{\exists n.y{=}2n\}}}{\{true\}\texttt{x:=1; y:=2x}\{\exists n.y = 2n\}}$$

larger proof

$$\dfrac{\{true\}\texttt{x:=1}\{true\} \quad \dfrac{\nabla_3}{\{true\}\texttt{y:=2x}\{\exists n.y{=}2n\}}}{\{true\}\texttt{x:=1; y:=2x}\{\exists n.y = 2n\}}$$

smaller proof

- Sufficient AI results:

$$\boxed{x{\to}\top, y{\to}\top}\ \texttt{x:=1;}\ \boxed{x{\to}\top, y{\to}\top}\ \texttt{y:=2x;}\ \boxed{x{\to}\top, y{\to}\text{even}}$$

Simple Example

Can program slicing, dependency analysis or any other techniques find this?

No, only abstract-value slicing can do.

Solution

- We propose an algorithm called Abstract-value Slicing (in short, AVS).
  - AVS filters out unnecessary invariants from AI results.
  - AVS works as a postprocessor to AI.

Example:

Insertion Sort with Zone Analysis

- Insertion sort
- Property to verify: safe array access
- Analysis technique: AI with zone domain

AI results

```
insertion_sort(n, A[1..n])
int i,j,pivot;
```

$true$

```
i:=2; j:=0;
```

$(2 \le i) \wedge (0 \le j \le i-2)$

```
while (i<=n) do
```

$(2 \le i \le n) \wedge (0 \le j \le i-2)$

```
  pivot:=A[i]; j:=i-1;
```

$(2 \le i \le n) \wedge (0 \le j \le n-1)$
$\wedge (2 \le n) \wedge (j \le i-1)$

```
  while (j>=1 and A[j]>pivot) do
```

$(2 \le i \le n) \wedge (1 \le j \le n-1)$
$\wedge (2 \le n) \wedge (j \le i-1)$

```
    A[j+1]:=A[j]; j:=j-1;
  od;
```

$(2 \le i \le n) \wedge (0 \le j \le n-1)$
$\wedge (2 \le n) \wedge (j \le i-1)$

```
  A[j+1]:=pivot; i:=i+1;
od
```

AI results

```
insertion_sort(n, A[1..n])
int i,j,pivot;
```

true

```
i:=2; j:=0;
```

$(2 \le i) \wedge (0 \le j \le i-2)$

```
while (i<=n) do
```

$\underline{(2 \le i \le n)} \wedge (0 \le j \le i-2)$

```
  pivot:=A[i]; j:=i-1;
```

$(2 \le i \le n) \wedge \underline{(0 \le j \le n-1)} \wedge (2 \le n) \wedge (j \le i-1)$

```
  while (j>=1 and A[j]>pivot) do
```

$(2 \le i \le n) \wedge \underline{(1 \le j \le n-1)} \wedge (2 \le n) \wedge (j \le i-1)$

```
    A[j+1]:=A[j]; j:=j-1;
  od;
```

$(2 \le i \le n) \wedge \underline{(0 \le j \le n-1)} \wedge (2 \le n) \wedge (j \le i-1)$

```
  A[j+1]:=pivot; i:=i+1;
od
```

Property to verify

```
insertion_sort(n, A[1..n])
int i,j,pivot;

i:=2; j:=0;

while (i<=n) do

  pivot:=A[i]; j:=i-1;

  while (j>=1 and A[j]>pivot) do

    A[j+1]:=A[j]; j:=j-1;
  od;

  A[j+1]:=pivot; i:=i+1;
od
```

AI results

```
insertion_sort(n, A[1..n])
int i,j,pivot;
```

$$true$$

```
i:=2; j:=0;
```

$$(2 \le i) \wedge (0 \le j \le i+2)$$

```
while (i<=n) do
```

$$\underline{(2 \le i \le n)} \wedge (0 \le j \le i-2)$$

```
  pivot:=A[i]; j:=i-1;
```

$$(2 \le i \le n) \wedge \underline{(0 \le j \le n-1)} \wedge (2 \le n) \wedge (j \le i-1)$$

```
  while (j>=1 and A[j]>pivot) do
```

$$(2 \le i \le n) \wedge \underline{(1 \le j \le n-1)} \wedge (2 \le n) \wedge (j \le i-1)$$

```
    A[j+1]:=A[j]; j:=j-1;
  od;
```

$$(2 \le i \le n) \wedge \underline{(0 \le j \le n-1)} \wedge (2 \le n) \wedge (j \le i-1)$$

```
  A[j+1]:=pivot; i:=i+1;
od
```

Abstract-value slicing

```
insertion_sort(n, A[1..n])
int i,j,pivot;
```

$$true$$

```
i:=2; j:=0;
```

$$(2 \le i)$$

```
while (i<=n) do
```

$$\underline{(2 \le i \le n)}$$

```
  pivot:=A[i]; j:=i-1;
```

$$(2 \le i \quad) \wedge \underline{(0 \le j \le n-1)}$$

```
  while (j>=1 and A[j]>pivot) do
```

$$(2 \le i \quad) \wedge \underline{(1 \le j \le n-1)}$$

```
    A[j+1]:=A[j]; j:=j-1;
  od;
```

$$(2 \le i \quad) \wedge \underline{(0 \le j \le n-1)}$$

```
  A[j+1]:=pivot; i:=i+1;
od
```

Abstract-value Slicing (1/2)

- Abstract-value slicing
  - AVS filters out unnecessary information from AI results
  - Technically, AVS weakens AI result $fixF$, such that
    - sliced AI result f is a conservative solution of AI

$$fixF \sqsubseteq f$$

    - sliced AI result is still enough to prove the property to verify ϕ

$$f \sqsubseteq \phi$$

Abstract-value Slicing (2/2)

- Two components of AVS
  - Extractor domain with extractor application:
    - is a working space of AVS indicating which information in AI results is necessary
  - Back-tracers for atomic terms
    - specify how AVS treats atomic terms

Example: Evenness

- Before AVS
  - AI results

$\boxed{x\to\top_e, y\to\top_e}$ y:=2y; $\boxed{x\to\top_e, y\to\text{even}}$ x:=2y; $\boxed{x\to\text{even},\ y\to\text{even}}$ y:=x $\boxed{x\to\text{even},\ y\to\text{even}}$

  - Verification goal (initial extractor annotation)

$\boxed{\{\}}$ y:=2y; $\boxed{\{\}}$ x:=2y; $\boxed{\{\}}$ y:=x $\boxed{\{y\}}$

Example: Evenness

- Before AVS
  - AI results

$$\boxed{x\to\top_e, y\to\top_e}\ \text{y:=2y;}\ \boxed{x\to\top_e, y\to\text{even}}\ \text{x:=2y;}\ \boxed{x\to\text{even},\ y\to\text{even}}\ \text{y:=x}\ \boxed{x\to\text{even},\ y\to\text{even}}$$

  - Verification goal (initial extractor annotation)

$$\boxed{\{\}}\ \text{y:=2y;}\ \boxed{\{\}}\ \text{x:=2y;}\ \boxed{\{\}}\ \text{y:=x}\ \boxed{\{y\}}$$

- After AVS
  - AVS results

$$\boxed{\{\}}\ \text{y:=2y;}\ \boxed{\{\}}\ \text{x:=2y;}\ \boxed{\{x\}}\ \text{y:=x}\ \boxed{\{y\}}$$

  - Sliced AI results

$$\boxed{x\to\top_e, y\to\top_e}\ \text{y:=2y;}\ \boxed{x\to\top_e, y\to\top_e}\ \text{x:=2y;}\ \boxed{x\to\text{even},\ y\to\top_e}\ \text{y:=x}\ \boxed{x\to\top_e,\ y\to\text{even}}$$

AVS: Extractor Domain $\mathcal{E}$

- An extractor domain $\mathcal{E}$ is a finite lattice
- An extractor application $\mathsf{ex} : \mathcal{E} \to \mathcal{A} \to \mathcal{A}$ is a function, such that

$$\forall e \in \mathcal{E}.\ a \sqsubseteq \mathsf{ex}(e)(a)$$

  - Top of extractor domain means that nothing is necessary among the given AI result.
  - Bottom of extractor domain means that all of AI result is necessary.

Example: Evenness

- Before AVS
  - AI results

$\boxed{x\to\top_e, y\to\top_e}$ y:=2y; $\boxed{x\to\top_e, y\to\text{even}}$ x:=2y; $\boxed{x\to\text{even}, y\to\text{even}}$ y:=x $\boxed{x\to\text{even}, y\to\text{even}}$

$$\mathcal{E} \stackrel{\text{def}}{=} \wp(\mathsf{Vars}) \qquad \mathsf{ex}(e)(a) \stackrel{\text{def}}{=} \lambda x.\, \textbf{if } x \in e \textbf{ then } a(x) \textbf{ else } \top_e$$

- After AVS
  - AVS results

$\boxed{\{\}}$ y:=2y; $\boxed{\{\}}$ x:=2y; $\boxed{\{x\}}$ y:=x $\boxed{\{y\}}$

  - Sliced AI results

$\boxed{x\to\top_e, y\to\top_e}$ y:=2y; $\boxed{x\to\top_e, y\to\top_e}$ x:=2y; $\boxed{x\to\text{even}, y\to\top_e}$ y:=x $\boxed{x\to\top_e, y\to\text{even}}$

AVS: Back-tracers $(\!|t|\!)_{ab}$

- An extractor transformer for atomic terms
  - Given an atomic term t and two abstract values $a, b \in \mathcal{A}$ satisfying

$$[\![t]\!]a \sqsubseteq b,$$

  - Back-tracer $(\!|t|\!)_{ab}$ is a function satisfying

$$(\!|t|\!)_{ab} \ : \ \mathcal{E} \to \mathcal{E}$$
$$\forall e \in \mathcal{E}.\ [\![t]\!]\Big(\mathsf{ex}\big((\!|t|\!)_{ab}(e)\big)(a)\Big) \sqsubseteq \mathsf{ex}(e)(b)$$

Example: Evenness

- Before AVS
  - AI results

$\boxed{x\to\top_e,\ y\to\top_e}$ y:=2y; $\boxed{x\to\top_e,\ y\to\text{even}}$ x:=2y; $\boxed{x\to\text{even},\ y\to\text{even}}$ y:=x $\boxed{x\to\text{even},\ y\to\text{even}}$

$\{\}$ $\{\}$ $\{x\}$ $\{y\}$

$$
\begin{aligned}
(\!|y := x|\!)_{ab}(e) &\stackrel{\text{def}}{=} \text{if } y \in e \text{ then } (e - \{y\}) \cup \{x\} \text{ else } e \\
(\!|x := 2E|\!)_{ab}(e) &\stackrel{\text{def}}{=} e - \{x\}
\end{aligned}
$$

Correctness

- Proposition

  For all $f \in \mathsf{postfix}(F)$ and all $\epsilon \in \Pi_{n \in V} \mathcal{E}$ the slicer $\mathsf{SL}(f, \epsilon)$ terminates, and it outputs ϵ' such that $\epsilon' \sqsubseteq \epsilon$

  $$F\Big(\mathsf{exall}(\epsilon')(f)\Big) \sqsubseteq \mathsf{exall}(\epsilon')(f).$$

  $$f \sqsubseteq \mathsf{exall}(\epsilon')(f) \sqsubseteq \mathsf{exall}(\epsilon)(f)$$

$\mathsf{exall}(\epsilon')(f)$ (1) is a correct AI solution;
(2) slices AI results; and
(3) proves the property of interest.

Contents

- A quick lookup
  - Problem and approach
  - Outline using insertion sort example
  - Abstract-value slicing : extractor domain & back-tracer
  - Correctness argument
- A little bit deeper
  - Abstract interpretation
  - Abstract-value slicer
  - Designing back-tracers
- Experiments
- Conclusion and discussion

Abstract Interpretation:

Syntax

Control-flow graph of a program, (V, E, n_i, n_f, L):

- V: a node represents a program point.
- E: an edge represents a flow of control.
- n_i, n_f: an initial and a final node.
- L of type $E \to \mathsf{ATerm}$: a labeling function.

Atomic terms ATerm are either assignments, boolean tests or `skip`.

Abstract Interpretation:
Semantics

AI components:

- Abstract domain, $\mathcal{A} = (A, \sqsubseteq, \bot, \sqcup)$.
- Abstract transfer function for ATerm, $[\![-]\!] : \mathsf{ATerm} \rightarrow (\mathcal{A} \rightarrow_m \mathcal{A})$.
- A strategy for fixpoint computation.

Abstract interpretation

- Step function F

$$
\begin{array}{lcll}
F & : & \prod_{n \in V} \mathcal{A} \rightarrow_m \prod_{n \in V} \mathcal{A} & \\
F(f)(n) & \stackrel{\text{def}}{=} & \begin{cases} a_0 & \text{if } n = n_i \\ \bigsqcup \{ [\![L(mn)]\!](f(m)) \mid mn \in E \} & \text{otherwise} \end{cases}
\end{array}
$$

- AI solution $fixF$: $F(fixF) \sqsubseteq fixF$

Example: Zone Analysis

Abstract domain $\mathcal{M} = (M, \sqsubseteq, \bot, \sqcup)$

$$M \stackrel{\text{def}}{=} \{a \mid a \text{ is a DBM}\} \qquad a \sqsubseteq a' \stackrel{\text{def}}{\Leftrightarrow} \forall ij.\ a_{ij} \leq a'_{ij}$$

$$\bot_{ij} \stackrel{\text{def}}{=} -\infty \qquad [a \sqcup a']_{ij} \stackrel{\text{def}}{=} \max(a_{ij}, a'_{ij})$$

$$\text{States} \stackrel{\text{def}}{=} [\{x_1, \ldots, x_N\} \to \text{Ints}]$$

$$\gamma(a) \stackrel{\text{def}}{=} \{\sigma \in \text{States} \mid \forall ij.\ \sigma[x_0 \to 0](x_j) - \sigma[x_0 \to 0](x_i) \leq a_{ij}\}$$

Abstract semantics of atomic terms

$$[\![x_i \leq c]\!]a \stackrel{\text{def}}{=} a[0i \to \min(a_{0i}, c)]$$

$$[\![x_i \geq c]\!]a \stackrel{\text{def}}{=} a[i0 \to \min(a_{i0}, -c)]$$

$$[\![x_i - x_j \leq c]\!]a \stackrel{\text{def}}{=} a[ji \to \min(a_{ji}, c)]$$

$$[\![E \leq E']\!]a \stackrel{\text{def}}{=} a \qquad \text{(for all other inequalities)}$$

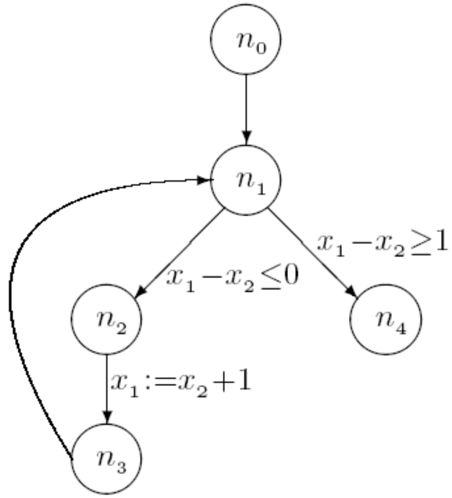
$$[\![x_i := x_i + c]\!]a \stackrel{\text{def}}{=} a\big[ki \to (a_{ki} + c),\ ik \to (a_{ik} + (-c))\big]_{0 \leq k (\neq i) \leq N}$$

$$[\![x_i := x_j + c]\!]a \stackrel{\text{def}}{=} a^*\big([ki \to \infty,\ ik \to \infty]_{0 \leq k(\neq i) \leq N}\big)[ji \to c,\ ij \to (-c)]$$

$$[\![x_i := c]\!]a \stackrel{\text{def}}{=} a^*\big([ki \to \infty,\ ik \to \infty]_{0 \leq k(\neq i) \leq N}\big)[0i \to c,\ i0 \to (-c)]$$

$$[\![x_i := E]\!]a \stackrel{\text{def}}{=} a^*\big([ki \to \infty,\ ik \to \infty]_{0 \leq k(\neq i) \leq N}\big) \qquad \text{(for all other assignments)}$$

$$[\![\texttt{skip}]\!]a \stackrel{\text{def}}{=} a$$



n_0

n_1

```
while (x1 - x2 <= 0) do
```

n_2

```
  x1 := x2 + 1
```

n_3

```
od
```

n_4

(a) Program

$$n_0 = n_i,\ n_4 = n_f,\ a_0 = \left[\begin{array}{c|ccc} & x_0 & x_1 & x_2 \\ \hline x_0 & \infty & 4 & 3 \\ x_1 & -1 & \infty & \infty \\ x_2 & -1 & 0 & \infty \end{array}\right]$$

(b) Entry, Exit Node and Initial Abstract State

	n_0	n_1	n_2	n_3	n_4
Analysis result as DBMs	$\begin{array}{c\|ccc} & x_0 & x_1 & x_2 \\ \hline x_0 & \infty & 4 & 3 \\ x_1 & -1 & \infty & \infty \\ x_2 & -1 & 0 & \infty \end{array}$	$\begin{array}{c\|ccc} & x_0 & x_1 & x_2 \\ \hline x_0 & \infty & \infty & 3 \\ x_1 & \infty & \infty & \infty \\ x_2 & -1 & 1 & \infty \end{array}$	$\begin{array}{c\|ccc} & x_0 & x_1 & x_2 \\ \hline x_0 & \infty & \infty & 3 \\ x_1 & \infty & \infty & \infty \\ x_2 & -1 & 0 & \infty \end{array}$	$\begin{array}{c\|ccc} & x_0 & x_1 & x_2 \\ \hline x_0 & \infty & \infty & 3 \\ x_1 & \infty & \infty & -1 \\ x_2 & -1 & 1 & \infty \end{array}$	$\begin{array}{c\|ccc} & x_0 & x_1 & x_2 \\ \hline x_0 & \infty & \infty & 3 \\ x_1 & \infty & \infty & -1 \\ x_2 & -1 & 1 & \infty \end{array}$
Analysis result as constraints	$1 \leq x_1 \leq 4$ $\wedge\ 1 \leq x_2 \leq 3$ $\wedge\ x_1 \leq x_2$	$1 \leq x_2 \leq 3$ $\wedge\ x_1 \leq x_2 + 1$	$1 \leq x_2 \leq 3$ $\wedge\ x_1 \leq x_2$	$1 \leq x_2 \leq 3$ $\wedge\ x_1 = x_2 + 1$	$1 \leq x_2 \leq 3$ $\wedge\ x_1 = x_2 + 1$

(c) Analysis Result

Abstract-value Slicing

- AVS components
  - Extractor domain with extractor application
  - Back-tracers for atomic terms
- Abstract-value slicer
  - Step function

$$(\!|P|\!)_{fg} \;:\; \left(\prod_{n\in V}\mathcal{E}\right) \rightarrow \left(\prod_{n\in V}\mathcal{E}\right)$$

$$(\!|P|\!)_{fg}(\epsilon)(n) \stackrel{\text{def}}{=} \bigsqcap \{(\!|L(nm)|\!)_{f(n)g(m)}(\epsilon(m)) \mid nm \in E\},$$

  - Abstract-value slicer

$$\mathsf{SL} \;:\; \left(\mathsf{postfix}(F) \times \prod_{n\in V}\mathcal{E}\right) \rightarrow \left(\prod_{n\in V}\mathcal{E}\right)$$

$$\begin{aligned}\mathsf{SL}(f,\epsilon) \stackrel{\text{def}}{=} \;& \mathbf{let}\; B_f = \lambda\epsilon'.(\epsilon' \sqcap (\!|P|\!)_{ff}\,\epsilon') \;\; \mathbf{and} \\ & \qquad k \;\; = \min\{n \mid n \geq 0 \wedge B_f^n(\epsilon) = B_f^{n+1}(\epsilon)\} \\ & \mathbf{in}\; B_f^k(\epsilon).\end{aligned}$$

Example:
Extractor Domain for Zone Analysis

Extractor domain (when $I = \{(i,j) \mid 0 \leq i,j \leq N\}$)

$$\mathcal{E} \stackrel{\text{def}}{=} \langle \wp(I), \supseteq, I, \emptyset, \cap, \cup \rangle$$

Extractor application

$$\left(\mathrm{ex}_a(e)\right)_{ij} \stackrel{\text{def}}{=} \begin{cases} a_{ij} & \text{if } ij \in e \\ \infty & \text{otherwise.} \end{cases}$$

Example:

Back-tracers for Zone Analysis

$$
\begin{array}{lcl}
(\!|x_i \le c|\!)_{ab}(e) & \stackrel{\text{def}}{=} & \textbf{if } (b_{0i} \ge c) \textbf{ then } (e - \{kl \mid b_{kl} = \infty\} - \{0i\}) \\
 & & \qquad\qquad\quad \textbf{else } (e - \{kl \mid b_{kl} = \infty\}) \\
(\!|x_i \ge c|\!)_{ab}(e) & \stackrel{\text{def}}{=} & \textbf{if } (b_{i0} \ge -c) \textbf{ then } (e - \{kl \mid b_{kl} = \infty\} - \{i0\}) \\
 & & \qquad\qquad\qquad \textbf{else } (e - \{kl \mid b_{kl} = \infty\}) \\
(\!|x_i - x_j \le c|\!)_{ab}(e) & \stackrel{\text{def}}{=} & \textbf{if } (b_{ji} \ge c) \textbf{ then } (e - \{kl \mid b_{kl} = \infty\} - \{ji\}) \\
 & & \qquad\qquad\quad \textbf{else } (e - \{kl \mid b_{kl} = \infty\}) \\
(\!|E \le E'|\!)_{ab}(e) & \stackrel{\text{def}}{=} & e - \{kl \mid b_{kl} = \infty\} \\
(\!|x_i := x_i + c|\!)_{ab}(e) & \stackrel{\text{def}}{=} & e - \{kl \mid b_{kl} = \infty\} \\
(\!|x_i := E|\!)_{ab}(e) & \stackrel{\text{def}}{=} & \textbf{if } (\mathsf{hasNegCycle}(a) = \mathit{true}) \\
 & & \textbf{then } \mathsf{edges}(\mathsf{pickNegCycle}(a)) \\
 & & \textbf{else let } e' = \left(e - \{kl \mid b_{kl} = \infty\} - \{ik, ki \mid 0 \le k \le N\}\right) \\
 & & \qquad \textbf{in } \bigcup_{kl \in e'} \left(\textbf{if } a_{kl} \le b_{kl} \textbf{ then } \{kl\} \textbf{ else } \mathsf{edges}(\mathsf{mPath}(a, k, l))\right) \\
 & & \quad (\text{where } E \text{ is either } x_j + c,\ c, \text{ or a general expression } E, \text{ and} \\
 & & \quad\ \mathsf{edges}(k_0 k_1 \ldots k_n) = \{k_0 k_1,\ k_1 k_2,\ \ldots,\ k_{n-1} k_n\}.\ ) \\
(\!|\mathtt{skip}|\!)_{ab}(e) & \stackrel{\text{def}}{=} & e - \{kl \mid b_{kl} = \infty\}
\end{array}
$$

	n_0	n_1	n_2	n_3	n_4
Initial extractor annotation	{}	{}	{}	{}	{(1,2), (2,1)}
Abstract interpretation result	$\begin{array}{cccc} & x_0 & x_1 & x_2 \\ \hline x_0 & \infty & 4 & 3 \\ x_1 & -1 & \infty & \infty \\ x_2 & -1 & 0 & \infty \end{array}$	$\begin{array}{cccc} & x_0 & x_1 & x_2 \\ \hline x_0 & \infty & \infty & 3 \\ x_1 & \infty & \infty & \infty \\ x_2 & -1 & 1 & \infty \end{array}$	$\begin{array}{cccc} & x_0 & x_1 & x_2 \\ \hline x_0 & \infty & \infty & 3 \\ x_1 & \infty & \infty & \infty \\ x_2 & -1 & 0 & \infty \end{array}$	$\begin{array}{cccc} & x_0 & x_1 & x_2 \\ \hline x_0 & \infty & \infty & 3 \\ x_1 & \infty & \infty & -1 \\ x_2 & -1 & 1 & \infty \end{array}$	$\begin{array}{cccc} & x_0 & x_1 & x_2 \\ \hline x_0 & \infty & \infty & 3 \\ x_1 & \infty & \infty & -1 \\ x_2 & -1 & 1 & \infty \end{array}$

(b) Input arguments for abstract-value slicer

	n_0	n_1	n_2	n_3	n_4
Result extractors from abstract-value slicer	{(2,1)}	{(2,1)}	{}	{(2,1)}	{(1,2), (2,1)}
Sliced abstract interpretation result	$\begin{array}{cccc} & x_0 & x_1 & x_2 \\ \hline x_0 & \infty & \infty & \infty \\ x_1 & \infty & \infty & \infty \\ x_2 & \infty & 0 & \infty \end{array}$	$\begin{array}{cccc} & x_0 & x_1 & x_2 \\ \hline x_0 & \infty & \infty & \infty \\ x_1 & \infty & \infty & \infty \\ x_2 & \infty & 1 & \infty \end{array}$	$\begin{array}{cccc} & x_0 & x_1 & x_2 \\ \hline x_0 & \infty & \infty & \infty \\ x_1 & \infty & \infty & \infty \\ x_2 & \infty & \infty & \infty \end{array}$	$\begin{array}{cccc} & x_0 & x_1 & x_2 \\ \hline x_0 & \infty & \infty & \infty \\ x_1 & \infty & \infty & \infty \\ x_2 & \infty & 1 & \infty \end{array}$	$\begin{array}{cccc} & x_0 & x_1 & x_2 \\ \hline x_0 & \infty & \infty & \infty \\ x_1 & \infty & \infty & -1 \\ x_2 & \infty & 1 & \infty \end{array}$

(c) Results from abstract-value slicer

	n_0	n_1	n_2	n_3	n_4
Before slicing	$1 \le x_1 \le 4$ $\wedge\ 1 \le x_2 \le 3$ $\wedge\ x_1 \le x_2$	$1 \le x_2 \le 3$ $\wedge\ x_1 \le x_2 + 1$	$1 \le x_2 \le 3$ $\wedge\ x_1 \le x_2$	$1 \le x_2 \le 3$ $\wedge\ x_1 = x_2 + 1$	$1 \le x_2 \le 3$ $\wedge\ x_1 = x_2 + 1$
After slicing	$x_1 \le x_2$	$x_1 \le x_2 + 1$		$x_1 \le x_2 + 1$	$x_1 = x_2 + 1$

(d) Results as constraints

Designing *Good* Back-tracers:
Best back-tracer construction

- Best back-tracer construction
  - When the abstract transfer function is join-preserving, the following is the best back-tracer

$$(\!|t|\!)_{ab}(e) \stackrel{\text{def}}{=} \bigsqcup\{e_0 \in \mathcal{E} \mid [\![t]\!](\mathsf{ex}(e_0)(a)) \sqsubseteq \mathsf{ex}(e)(b)\}.$$

Designing *Good* Back-tracers :

Back-tracers for Zone Analysis

$$(\!|x_i \le c|\!)_{ab}(e) \stackrel{\text{def}}{=} \textbf{if } (b_{0i} \ge c) \textbf{ then } (e - \{kl \mid b_{kl} = \infty\} - \{0i\}) \textbf{ else } (e - \{kl \mid b_{kl} = \infty\})$$

$$(\!|x_i \ge c|\!)_{ab}(e) \stackrel{\text{def}}{=} \textbf{if } (b_{i0} \ge -c) \textbf{ then } (e - \{kl \mid b_{kl} = \infty\} - \{i0\}) \textbf{ else } (e - \{kl \mid b_{kl} = \infty\})$$

$$(\!|x_i - x_j \le c|\!)_{ab}(e) \stackrel{\text{def}}{=} \textbf{if } (b_{ji} \ge c) \textbf{ then } (e - \{kl \mid b_{kl} = \infty\} - \{ji\}) \textbf{ else } (e - \{kl \mid b_{kl} = \infty\})$$

$$(\!|E \le E'|\!)_{ab}(e) \stackrel{\text{def}}{=} e - \{kl \mid b_{kl} = \infty\}$$

$$(\!|x_i := x_i + c|\!)_{ab}(e) \stackrel{\text{def}}{=} e - \{kl \mid b_{kl} = \infty\}$$

$$(\!|x_i := E|\!)_{ab}(e) \stackrel{\text{def}}{=} \textbf{if } (\mathsf{hasNegCycle}(a) = \mathit{true}) \textbf{ then } \mathsf{edges}(\mathsf{pickNegCycle}(a)) \textbf{ else let } e' = (e - \{kl \mid b_{kl} = \infty\} - \{ik, ki \mid 0 \le k \le N\}) \textbf{ in } \bigcup_{kl \in e'} \left(\textbf{if } a_{kl} \le b_{kl} \textbf{ then } \{kl\} \textbf{ else } \mathsf{edges}(\mathsf{mPath}(a,k,l))\right)$$

(where E is either $x_j + c$, c, or a general expression E, and $\mathsf{edges}(k_0 k_1 \ldots k_n) = \{k_0 k_1, k_1 k_2, \ldots, k_{n-1} k_n\}$.)

$$(\!|\mathtt{skip}|\!)_{ab}(e) \stackrel{\text{def}}{=} e - \{kl \mid b_{kl} = \infty\}$$

Designing *Good* Back-tracers :
Extension Method

- Dual atomic domain
  - An element x in a lattice L is a dual atom iff

$$x \neq \top \wedge (\forall x' \in L.\, (x \sqsubseteq x' \wedge x' \neq x) \Rightarrow x' = \top),$$

  - L is dual atomic iff

$$\forall x \in L.\, x = \bigsqcap \{x' \mid x \sqsubseteq x' \text{ and } x' \text{ is a dual atom}\}.$$

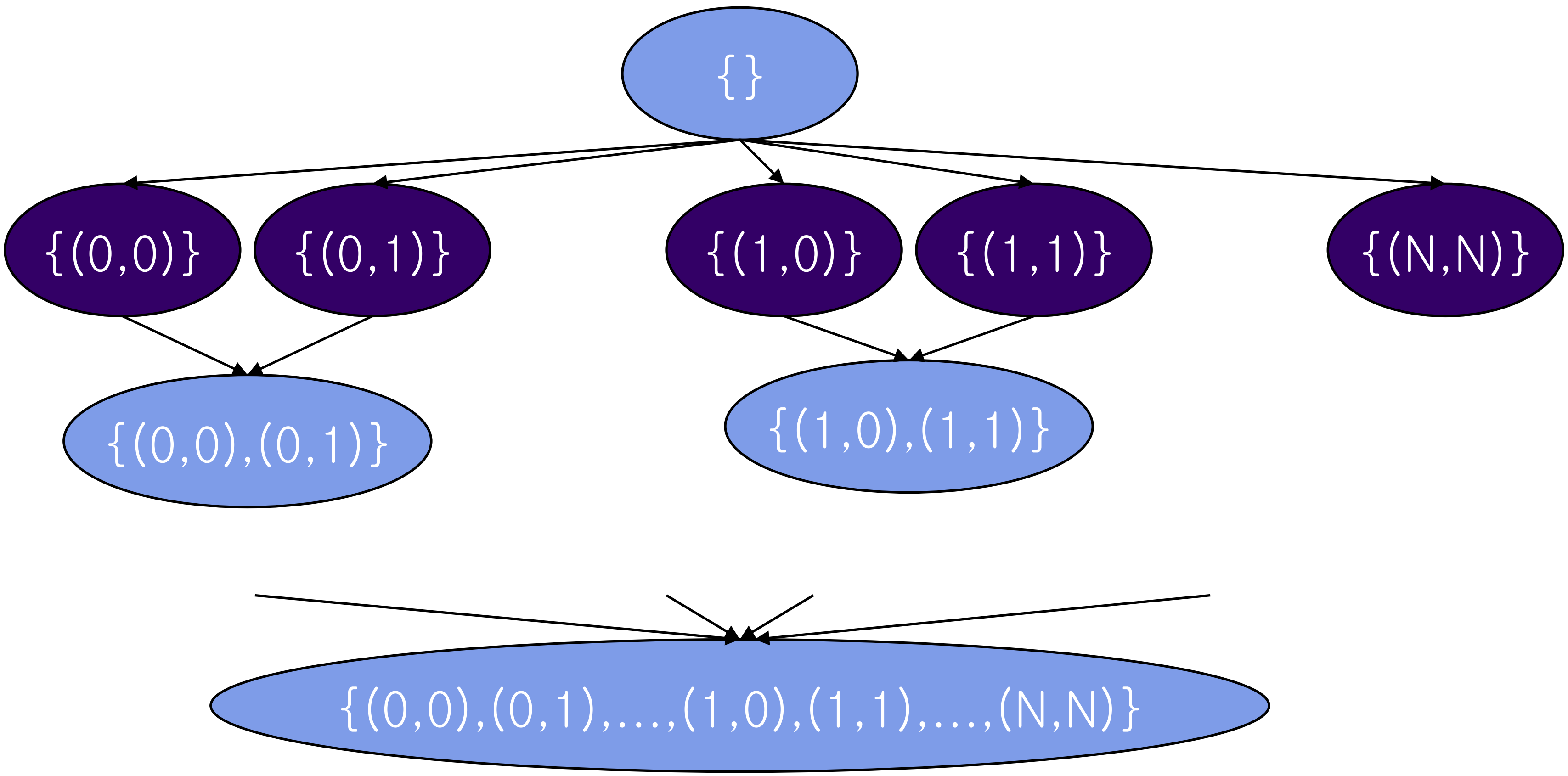
- When the extractor domain is dual atomic, the following is the back-tracer

$$(\!| t |\!)_{ab}(e) \stackrel{\text{def}}{=} \bigsqcap \{g(e_1) \mid e \sqsubseteq e_1 \quad \text{and} \quad e_1 \text{ is a dual atom}\}.$$

Designing *Good* Back-tracers : Extractor domain for Zone analysis

Extractor domain (when $I = \{(i,j) \mid 0 \le i, j \le N\}$)

$$\mathcal{E} \stackrel{\text{def}}{=} \langle \wp(I), \supseteq, I, \emptyset, \cap, \cup \rangle$$



Designing *Good* Back-tracers :

Back-tracers for Zone Analysis

$$
\begin{aligned}
\llparenthesis x_i \le c \rrparenthesis_{ab}(e) &\stackrel{\text{def}}{=} \textbf{if } (b_{0i} \ge c) \textbf{ then } (e - \{kl \mid b_{kl} = \infty\} - \{0i\}) \\
&\qquad\qquad \textbf{else } (e - \{kl \mid b_{kl} = \infty\}) \\
\llparenthesis x_i \ge c \rrparenthesis_{ab}(e) &\stackrel{\text{def}}{=} \textbf{if } (b_{i0} \ge -c) \textbf{ then } (e - \{kl \mid b_{kl} = \infty\} - \{i0\}) \\
&\qquad\qquad \textbf{else } (e - \{kl \mid b_{kl} = \infty\}) \\
\llparenthesis x_i - x_j \le c \rrparenthesis_{ab}(e) &\stackrel{\text{def}}{=} \textbf{if } (b_{ji} \ge c) \textbf{ then } (e - \{kl \mid b_{kl} = \infty\} - \{ji\}) \\
&\qquad\qquad \textbf{else } (e - \{kl \mid b_{kl} = \infty\}) \\
\llparenthesis E \le E' \rrparenthesis_{ab}(e) &\stackrel{\text{def}}{=} e - \{kl \mid b_{kl} = \infty\} \\
\llparenthesis x_i := x_i + c \rrparenthesis_{ab}(e) &\stackrel{\text{def}}{=} e - \{kl \mid b_{kl} = \infty\} \\
\llparenthesis x_i := E \rrparenthesis_{ab}(e) &\stackrel{\text{def}}{=} \textbf{if } (\mathsf{hasNegCycle}(a) = \mathit{true}) \\
&\qquad \textbf{then } \mathsf{edges}(\mathsf{pickNegCycle}(a)) \\
&\qquad \textbf{else let } e' = \left(e - \{kl \mid b_{kl} = \infty\} - \{ik, ki \mid 0 \le k \le N\}\right) \\
&\qquad\quad \textbf{in } \bigcup_{kl \in e'} \left(\textbf{if } a_{kl} \le b_{kl} \textbf{ then } \{kl\} \textbf{ else } \mathsf{edges}(\mathsf{mPath}(a, k, l))\right) \\
&\qquad \text{(where } E \text{ is either } x_j + c,\ c, \text{ or a general expression } E, \text{ and} \\
&\qquad\ \mathsf{edges}(k_0 k_1 \ldots k_n) = \{k_0 k_1, k_1 k_2, \ldots, k_{n-1} k_n\}.\ ) \\
\llparenthesis \mathtt{skip} \rrparenthesis_{ab}(e) &\stackrel{\text{def}}{=} e - \{kl \mid b_{kl} = \infty\}
\end{aligned}
$$

Experiments (1/3)

- We implement
  - Abstract interpreter for zone analysis
  - Abstract-value slicer for zone analysis
  - Hoare proof construction algorithm

- We apply our algorithms to small array-accessing programs

Experiments (2/3)

- Abstract interpretation results

programs	number of invariants in AI results			removed /total	slicing time (sec)
	total	selected	removed		
Insertion sort	92	22	70	76%	0.07
Partition	120	46	74	62%	0.03
Bubble sort	217	42	175	81%	0.11
KMP	463	133	330	72%	0.28
Heap sort	817	181	636	78%	0.29

Experiments (3/3)

- Hoare proof size

	before slicing		after slicing		(1)-(2) /(1)	reduction in proof size
	(1)FOL	formula	(2)FOL	formula		
Insertion sort	248	2530	166	1122	33%	53%
Partition	398	3866	201	1847	49%	52%
Bubble sort	894	12230	389	2677	56%	76%
KMP	1364	26898	653	7683	52%	70%
Heap sort	2542	52370	1028	7936	60%	84%

Conclusion

- Our contribution
  - Abstract-value slicing
    - AVS eliminates unnecessary invariants from AI results;
    - General framework for designing AVS is proposed; and
    - Constructing correct parameters for AVS and designing AVS for various AI frameworks are suggested.
  - We show applicability of our works by experiments.

(All details can be found in our TOPLAS paper and related technical memo)

Discussion

- Points to consider
  - Back-tracers are no need to be monotone.
  - Under-approximation vs. over-approximation
  - Forward vs. backward analysis

Thanks.