A *posteriori* soundness for nondeterministic abstract interpretations

Matthew Might (University of Utah)
Panagiotis Manolios (Northeastern University)
Questions you don’t want at your defense
Questions you don’t want at your defense

• “But, why did you prove it that way?”
Questions you don’t want at your defense

• “But, why did you prove it *that* way?”

• “But, why is that *necessary*?”
Questions you don’t want at your defense

- “But, why did you prove it *that* way?”
- “But, why is that *necessary*?”
- “So, why *did* the Cousots do it that way?”
Nondeterministic Abstract Interpretation

- Where did it come from?
- How do you prove it sound?
- Why would you want to use it?
Nondeterministic Abstract Interpretation

• Where did it come from?
  • Frustration with the standard recipe.
• How do you prove it sound?
• Why would you want to use it?
Nondeterministic Abstract Interpretation

- Where did it come from?
  - Frustration with the standard recipe.
- How do you prove it sound?
  - A *posteriori* proof technique.
- Why would you want to use it?
Nondeterministic Abstract Interpretation

- Where did it come from?
  - Frustration with the standard recipe.
- How do you prove it sound?
  - A posteriori proof technique.
- Why would you want to use it?
  - Better speed, better precision.
Outline

• Review standard recipe.
• Find annoyances.
• Get rid of them.
The Standard Recipe

Define concrete state-space: $L$

Define concrete semantics: $f : L \rightarrow L$

Define abstract state-space: $\hat{L}$

Define abstraction map: $\alpha : L \rightarrow \hat{L}$

Define abstract semantics: $\hat{f} : \hat{L} \rightarrow \hat{L}$

Prove $\hat{f}$ simulates $f$ under $\alpha$.
The A Posteriori Recipe

Define concrete state-space: $L$

Define concrete semantics: $f : L \rightarrow L$

Define abstract state-space: $\hat{L}$

Define abstract semantics: $\hat{f} : \hat{L} \rightarrow \hat{L}$

Execute abstract semantics to obtain $\hat{\ell}' = \hat{f}(\hat{\ell})$.

Define abstraction map: $\alpha : L \rightarrow \hat{L}$

Prove $\hat{f}$ simulates $f$ under $\alpha$. 
The A Posteriori Recipe

Define concrete state-space: \( L \)

Define concrete semantics: \( f : L \to L \)

Define abstract state-space: \( \hat{L} \)

Define abstract semantics: \( \hat{f} : \hat{L} \to \hat{L} \)

Execute abstract semantics to obtain \( \hat{\ell}' = \hat{f}(\hat{\ell}) \).

Define abstraction map: \( \alpha : L \to \hat{L} \)

Prove \( \hat{\ell}' \) simulates \( f \) under \( \alpha \).
The A Posteriori Recipe

Define concrete state-space: \( L \)

Define concrete semantics: \( f : L \rightarrow L \)

Define abstract state-space: \( \hat{L} \)

Define abstract semantics: \( \hat{f} : \hat{L} \rightarrow 2^{\hat{L}} \)

Execute abstract semantics to obtain \( \hat{\ell}' = \hat{f}(\hat{\ell}). \)

Define abstraction map: \( \alpha : L \rightarrow \hat{L} \)

Prove \( \hat{\ell}' \) simulates \( f \) under \( \alpha \).
Illustrating the Standard Recipe
Malloc: The Language

\[ \nu := \text{malloc()} \]
Malloc: The Language

\[ lab : v := \text{malloc}() \]
Concrete Semantics

\[ State = Instruction \times Store \]
Concrete Semantics

State = Instruction × Store

\[ f(\varsigma) = \varsigma' \]
Concrete Semantics

\[ State = Instruction \times Store \]

\[ f([v := \text{malloc}()] : \vec{i}, \sigma) = (\vec{i}, \sigma[v \mapsto a']) \]
Concrete Semantics

State = Instruction \times Store

f(\{v := malloc()\}: \vec{i}, \sigma) = (\vec{i}, \sigma[v \mapsto a'])

a' = alloc(\varsigma)
Concrete Semantics

\[ State = Instruction \times Store \]

\[ f([v := malloc()] : i, \sigma) = (i, \sigma[v \mapsto a']) \]

\[ a' = alloc(\varsigma) = \max(\text{range}(\sigma)) + 1 \]
Abstract Semantics

\[ \hat{\text{State}} = \hat{\text{Instruction}} \times \hat{\text{Store}} \]

\[ \hat{f}(v := \text{malloc}()) : \vec{i}, \hat{\sigma} = (\vec{i}, \hat{\sigma}[v \mapsto \hat{a}]) \]

\[ \hat{a} = \text{alloc}(\hat{\xi}) \quad \text{(from some finite set)} \]
What to allocate?

- Abstract addresses = Scarce resource
- Avoid over-allocation: Good for speed
- Avoid under-allocation: Good for precision
Example: Over-allocation

\[ \hat{a}_1 \rightarrow 3 \rightarrow \hat{a}_2 \]
Example: Over-allocation

\[ \hat{a}_{1,2} \rightarrow 3 \]
Example: Under-allocation

\[ \hat{a}' \]

\[ 3 \]

\[ 4 \]
Example: Under-allocation

\[ \hat{a}_1 \rightarrow 3 \]

\[ \hat{a}_2 \rightarrow 4 \]
Allocation heuristics

Observation: Objects from like contexts act alike.
Allocation heuristics

Observation: Objects from like contexts act alike.

Example: \( \widehat{\text{alloc}}([\text{lab} : \ldots ] : \vec{i}, \_ ) = \text{lab} \)
Annoyance: Soundness

If

\[ \alpha(\varsigma) \sqsubseteq \hat{\varsigma} \]

then

\[ \alpha(f(\varsigma)) \sqsubseteq \hat{f}(\hat{\varsigma}) \]
Annoyance: Soundness

If

\[ \alpha(\varsigma) \subseteq \hat{\varsigma} \]

then

\[ \alpha_{Addr}(\text{alloc}(\varsigma)) \subseteq \text{alloc}(\hat{\varsigma}) \]
The Issue

\[ \text{alloc}(\_, \sigma) = \max(\text{range}(\sigma)) + 1 \]

\[ \text{alloc}([\text{lab} : \ldots] : \vec{i}, \_) = \text{lab} \]

What abstraction map will work here?
Example

\[ A : x := \text{malloc}() \]
\[ B : y := \text{malloc}() \]

\[ \sigma = [x \rightarrow 1, y \rightarrow 2] \]

\[ \alpha_{\text{Addr}} = [1 \rightarrow A, 2 \rightarrow B] \]
Example

\[
\begin{align*}
B & : y := \text{malloc}() \\
A & : x := \text{malloc}() \\
\sigma & = [x \mapsto 1, y \mapsto 2] \\
\alpha_{\text{Addr}} & = [1 \mapsto A, 2 \mapsto B]
\end{align*}
\]
Example

\[ B : y := \text{malloc()}, \quad A : x := \text{malloc()} \]

\[ \sigma = [x \rightarrow 2, y \rightarrow 1] \]

\[ \alpha_{Addr} = [1 \rightarrow A, 2 \rightarrow B] \]
Example

\[ B : y := \text{malloc()} \]
\[ A : x := \text{malloc()} \]

\[ \sigma = [x \rightarrow 2, y \rightarrow 1] \]

\[ \alpha_{\text{Addr}} = [2 \rightarrow A, 1 \rightarrow B] \]
Standard Solution

Change the concrete semantics!
Standard Solution

Change the concrete semantics!

\[ Addr = \mathbb{N} \]

\[ alloc(-, \sigma) = \max(range(\sigma)) + 1 \]
Standard Solution

Change the concrete semantics!

\[ Addr = \mathbb{N} \times Lab \]

\[ alloc([[lab : \ldots]], \sigma) = (\max(\text{range}(\sigma)_1) + 1, lab) \]
Standard Solution

Change the concrete semantics!

\[
\text{Addr} = \mathbb{N} \times \text{Lab}
\]

\[
\text{alloc}([\text{lab} : \ldots], \sigma) = (\max(\text{range}(\sigma)_1) + 1, \text{lab})
\]

\[
\alpha(\_, \text{lab}) = \text{lab}
\]
Another problem: Heuristics sometimes make stupid decisions
Another problem: Heuristics sometimes make stupid decisions

Why not adapt on the fly?
Example: Greedy Strategy

Heuristic says, “Allocate $\hat{a}_1$, and bind 4.”
Example: Greedy Strategy

Heuristic says, “Allocate $\alpha_1$, and bind 4.”
Example: Greedy Strategy

Heuristic says, “Allocate $\hat{a}_1$, and bind 4.”

Adaptive allocator says, “Try $r(\hat{a}_1)$ first.”
Example: Greedy Strategy

Heuristic says, “Allocate $\hat{a}_1$, and bind 4.”

Adaptive allocator says, “Try $r(\hat{a}_1)$ first.”
Example: Greedy Strategy

Heuristic says, “Allocate \( \hat{a}_2 \), and bind 3.”
Example: Greedy Strategy

Heuristic says, “Allocate $\hat{a}_2$, and bind 3.”
Example: Greedy Strategy

Heuristic says, "Allocate $\hat{a}_2$, and bind 3."

Adaptive allocator says, "Just use $\hat{a}_1$."
Example: Greedy Strategy

Heuristic says, “Allocate $\hat{a}_2$, and bind 3.”

Adaptive allocator says, “Just use $\hat{a}_1$."

\[ \hat{a}_1 \rightarrow 3 \]
Dynamic Optimization

Given $m$ abstract addresses, how should they be allocated to maximize precision?
So, why not?

Can’t within confines of standard recipe.

(Counter-example in paper.)
Making it so
Making it so

• Factor allocation out of semantics.
• Make allocation nondeterministic.
• Prove nondeterministic allocation sound.
Locative = Address

(But also times, bindings, contours, etc.)
Factoring out allocation
\( f : State \rightarrow State \)
\( f : \text{State} \rightarrow \text{State} \)
\[ f : \text{State} \rightarrow \text{State} \]
\[ F : State \rightarrow Loc \rightarrow State \]
\[ F : State \rightarrow Loc \rightarrow State \]
\[ F : State \rightarrow Loc \rightarrow State \]
\( \hat{f} : \overline{\text{State}} \rightarrow 2^{\overline{\text{State}}} \)
\[ \hat{f} : \text{State} \rightarrow 2^{\text{State}} \]
$\hat{f} : \widehat{\text{State}} \rightarrow \widehat{2^{\text{State}}}$
\( \hat{F} : \widehat{\text{State}} \rightarrow 2^{\widehat{\text{Loc}}} \rightarrow \widehat{\text{State}} \)
$\hat{F} : \widehat{State} \rightarrow 2^{\widehat{Loc} \rightarrow \widehat{State}}$
\( \hat{F} : \hat{\text{State}} \rightarrow 2^{\hat{\text{Loc}} \rightarrow \hat{\text{State}}} \)
Nondeterministic Abstract Interpretation
Nondeterministic Abstract Interpretation

- Sealed abstract transition graphs.
- Factored abstraction maps.
- A posteriori soundness condition.
Transition Graphs

- Nodes = States
- Edge = Transition labeled by chosen locative
Sealed Graphs

Graph is **sealed** under factored semantics iff every state has an edge to cover every transition.
Example: Unsealed Graph
Example: Unsealed Graph
\[ \hat{F}(\hat{\varsigma}) = \{ \hat{h}_1, \hat{h}_2, \hat{h}_3 \} \]
\[ \hat{F}(\hat{\varsigma}) = \{\hat{h}_1, \hat{h}_2, \hat{h}_3\} \]
Proving Sealed Graphs Sound
Factoring Abstraction

\[ \alpha : State \rightarrow \hat{State} \]
Factoring Abstraction

\[ \alpha : \text{State} \rightarrow \hat{\text{State}} \]

\[ \beta : (\text{Loc} \rightarrow \hat{\text{Loc}}) \rightarrow (\text{State} \rightarrow \hat{\text{State}}) \]
Dependent Simulation
Dependent Simulation
Dependent Simulation

$s \xrightarrow{\ell} s'$
Dependent Simulation

\[ \begin{array}{c}
\varsigma \\
\Downarrow \beta(\alpha_{Loc}[\ell \mapsto \hat{\ell}])
\end{array} \Rightarrow \begin{array}{c}
\varsigma' \\
\downarrow \\
\hat{\varsigma}'
\end{array} \]
Dependent Simulation

\[ \beta(\alpha_{Loc}) \]

\[ \hat{s} \rightarrow \hat{s}' \]

\[ \ell \]

\[ \hat{\ell} \rightarrow \hat{\ell}' \]

\[ \beta(\alpha_{Loc}[\ell \mapsto \hat{\ell}]) \]
A Posteriori Theorem

Dependent simulation $\rightarrow$ Abstraction always exists
Proof Highlights

- Reduces to existence of locative abstractor.
- Construct abstractor as limit of sequence:

\[ \alpha_{Loc} = \lim_{i \to N} \alpha^i_{Loc} \]
More in the paper

- Nondeterministic CFA: $\exists CFA$.
- More on greedy adaptive allocation.
- Discussion of global precision sensitivity.
Ongoing Work

- Empirical trials: 1.5x - 3x space, time savings
- Genetic algorithms
- Probabilistic allocation
So...

- Stop changing concrete semantics.
- Look beyond context for allocation.
- Don’t allocate context if bad for precision.
Thanks, y'all