# Modular Analysis

Kwangkeun Yi
(co-work with Joonhyup Lee)

Seoul National University, Korea

10/20/2024 @ SAS'24, Pasadena

# Towards a Modular Static Analysis Framework

Existing modular analysis instances: function-summary-based ones

- Wanted: a more "general" design framework
  - leverageing the AI framework
- Looking for: a modular semantics $[\![\cdot]\!]_M$ such that
  - modular analysis design = a finite, sound abstraction

$$[\![\text{pgm}]\!]_M^{\#}$$

  - modular analysis implementation = immediate from

$$[\![\text{pgm}]\!]_M^{\#}$$

# Towards a Modular Static Analysis Framework

For program

$$e_1 \rtimes e_2 \quad \text{of two modules } e_1 \text{ and } e_2,$$

we want its modular semantics $[\![\cdot]\!]_M$ and linking $\propto$ to satisfy

$$[\![e_1 \rtimes e_2]\!] = [\![e_1]\!]_M \propto [\![e_2]\!]_M$$

so that a modular analysis be

$$[\![e_1]\!]^{\#}_M \propto^{\#} [\![e_2]\!]^{\#}_M$$

with

$$[\![e_1 \rtimes e_2]\!] \subseteq \gamma([\![e_1]\!]^{\#}_M \propto^{\#} [\![e_2]\!]^{\#}_M).$$

# Towards a Modular Static Analysis Framework

For program

$$e_1 \rtimes e_2 \quad \text{of two modules } e_1 \text{ and } e_2,$$

we want its modular semantics $[\![\cdot]\!]_M$ and linking $\propto$ to satisfy

$$[\![e_1 \rtimes e_2]\!] = [\![e_1]\!]_M \propto [\![e_2]\!]_M$$

so that a modular analysis be

$$[\![e_1]\!]^{\#}_M \propto^{\#} [\![e_2]\!]^{\#}_M$$

with

$$[\![e_1 \rtimes e_2]\!] \subseteq \gamma([\![e_1]\!]^{\#}_M \propto^{\#} [\![e_2]\!]^{\#}_M).$$

- What are such $[\![\cdot]\!]_M$ and $\propto$?

# Towards a Modular Static Analysis Framework

For program

$$e_1 \rtimes e_2 \quad \text{of two modules } e_1 \text{ and } e_2,$$

we want its modular semantics $[\![\cdot]\!]_M$ and linking $\propto$ to satisfy

$$[\![e_1 \rtimes e_2]\!] = [\![e_1]\!]_M \propto [\![e_2]\!]_M$$

so that a modular analysis be

$$[\![e_1]\!]^{\#}_M \propto^{\#} [\![e_2]\!]^{\#}_M$$

with

$$[\![e_1 \rtimes e_2]\!] \subseteq \gamma([\![e_1]\!]^{\#}_M \propto^{\#} [\![e_2]\!]^{\#}_M).$$

- What are such $[\![\cdot]\!]_M$ and $\propto$?
- Our requirements:
  - operational modular semantics (immediate to implement)
  - a few proofs to do (the rest proven in the framework)
  - align with the AI framework (general & small hurdle)

# Our Modular Analysis Framework: Recipe

# Our Modular Analysis Framework: Recipe

1. Define collecting semantics:

$$\mathsf{eval}(e, \Sigma)$$

for a set $\Sigma$ of states, where

$$\mathsf{eval}(e_1 \rtimes e_2, \Sigma) \triangleq \mathsf{eval}(e_2, \mathsf{eval}(e_1, \Sigma)).$$

# Our Modular Analysis Framework: Recipe

1. Define collecting semantics:

$$\mathsf{eval}(e, \Sigma)$$

   for a set $\Sigma$ of states, where

$$\mathsf{eval}(e_1 \rtimes e_2, \Sigma) \triangleq \mathsf{eval}(e_2, \mathsf{eval}(e_1, \Sigma)).$$

2. Prove *advance lemma*:

$$\mathsf{eval}(e, \Sigma \infty \Sigma_0) \subseteq \Sigma \infty \mathsf{eval}(e, \Sigma_0).$$

   Then follows

$$\mathsf{eval}(e_1 \rtimes e_2, \Sigma) \subseteq \mathsf{eval}(e_1, \Sigma) \infty \mathsf{eval}(e_2, \mathsf{Init}).$$

3. Hence a sound modular analysis is, for sound $\mathsf{eval}^{\#}$ and $\infty^{\#}$,

$$\mathsf{eval}^{\#}(e_1, \Sigma^{\#}) \infty^{\#} \mathsf{eval}^{\#}(e_2, \mathsf{Init}^{\#}).$$

# Enabler: Shadow Semantics

*Shadow semantics*: $\mathsf{eval}(e, \Sigma_0)$ with incomplete environment $\Sigma_0$

- to satisfy the *advance lemma*:

$$\mathsf{eval}(e, \Sigma \propto \Sigma_0) \subseteq \Sigma \propto \mathsf{eval}(e, \Sigma_0).$$

*Shadow*

- a composition of semantic operation symbols
- a semantic rep. of "symbolic-execution" results
- a semantic rep. of the computations to finish at link-time
- can be an infinite object.

# Shadow Semantics Examples (1/3)

For "`x + 1 + 2`"

- Shadow semantics with Init (unknown environment) is

$$\{\mathsf{Read}(\mathsf{Init}, \mathtt{x}) + 3\}$$

For "`f(x) + 1 + 2`"

- Shadow semantics with Init is

$$\{\mathsf{Call}(\mathsf{Read}(\mathsf{Init}, \mathtt{f}), \mathsf{Read}(\mathsf{Init}, \mathtt{x})) + 3\}$$

# Shadow Semantics Examples (2/3)

For

```
let sum f 0 = f 0
  | sum f n = (f n) + (sum f (n-1))
in sum g 3
```

Shadow semantics with Init is

$$\{\mathsf{Call}(G, 3) + \mathsf{Call}(G, 2) + \mathsf{Call}(G, 1) + \mathsf{Call}(G, 0)\}$$

where $G \triangleq \mathsf{Read}(\mathsf{Init}, \mathsf{g})$.

# Shadow Semantics Examples (3/3)

For

```
let sum f 0 = f 0
  | sum f n = (f n) + (sum f (n-1))
in sum g m
```

Shadow semantics with Init is

$$
\begin{aligned}
\{&\mathsf{Call}(G, 0),\\
&\mathsf{Call}(G, M) + \mathsf{Call}(G, 0),\\
&\mathsf{Call}(G, M) + \mathsf{Call}(G, \mathsf{Pred}(M)) + \mathsf{Call}(G, 0),\\
&\mathsf{Call}(G, M) + \mathsf{Call}(G, \mathsf{Pred}(M)) + \mathsf{Call}(G, \mathsf{Pred}^2(M)) + \mathsf{Call}(G, 0),\\
&\cdots\}
\end{aligned}
$$

where $G \triangleq \mathsf{Read}(\mathsf{Init}, \mathtt{g})$ and $M \triangleq \mathsf{Read}(\mathsf{Init}, \mathtt{m})$.

# Our Framework for Model Language L

Call-by-value, first-class functions, first-class modules, and recursive bindings

- Abstract syntax of L

$$
\begin{array}{llcll}
\text{Expression} & e & \rightarrow & x \mid \lambda x.e \mid e\ e & \lambda\text{-calculus} \\
 & & \mid & \varepsilon & \text{empty module} \\
 & & \mid & x = e \ ;\ e & \text{module def (bindings)} \\
 & & \mid & e \rtimes e & \text{module use (linking)}
\end{array}
$$

- Examples:

$$
\begin{array}{ll}
1) & (\mathtt{x} = 1 \ ;\ \mathtt{y} = 2 \ ;\ \varepsilon) \rtimes \mathtt{x} \\
2) & (\mathtt{M} = (\mathtt{x} = 1 \ ;\ \mathtt{y} = 2 \ ;\ \varepsilon) \ ;\ \varepsilon) \rtimes (\mathtt{M} \rtimes \mathtt{x})
\end{array}
$$

- Semantic domains

$$
\begin{array}{llcll}
\text{Value} & v & \rightarrow & \langle \lambda x.e, \sigma \rangle & \text{closure} \\
 & & \mid & \sigma & \text{environment} \\
\text{Environment} & \sigma & \rightarrow & \bullet & \text{empty environment} \\
 & & \mid & (x, v) :: \sigma & \text{add binding}
\end{array}
$$

# Shadow Semantics, to Satisfy the Advance Lemma

- Defined its normal semantics: concrete semantic rules

$$\sigma \vdash e \Downarrow v$$

# Shadow Semantics, to Satisfy the Advance Lemma

- Defined its normal semantics: concrete semantic rules

$$\sigma \vdash e \Downarrow v$$

- Extended to its shadow semantics: add rules with shadows for free vars

$$\sigma^\dagger \vdash e \Downarrow v^\dagger$$

| | | | | |
|---|---|---|---|---|
| Shadowed Env | $\sigma^\dagger$ | $\to$ | $\bullet \mid (x, v^\dagger) :: \sigma^\dagger$ | |
| | | $\mid$ | $S$ | shadow as env |
| Shadowed Val | $v^\dagger$ | $\to$ | $\langle \lambda x.e, \sigma^\dagger \rangle \mid \sigma^\dagger$ | |
| | | $\mid$ | $S$ | shadow as value |
| Shadow | $S$ | $\to$ | Init | initial unknown env |
| | | $\mid$ | Read$(S, x)$ | read shadow |
| | | $\mid$ | Call$(S, v^\dagger)$ | call shadow |

# Linking, to Satisfy the Advance Lemma

- $\backpropto \in \mathsf{Env} \to \mathsf{Shadow} \to \mathcal{P}(\mathsf{Val})$

- $\backpropto \in \mathsf{Env} \to \mathsf{Env} \to \mathcal{P}(\mathsf{Env})$
- $\backpropto \in \mathsf{Env} \to \mathsf{Val} \to \mathcal{P}(\mathsf{Val})$

# Linking, to Satisfy the Advance Lemma

- $\propto \in \mathsf{Env} \to \mathsf{Shadow} \to \mathcal{P}(\mathsf{Val})$

$$\sigma_0 \quad \propto \quad \mathsf{Init} \qquad\qquad \triangleq \quad \{\sigma_0\}$$

- $\propto \in \mathsf{Env} \to \mathsf{Env} \to \mathcal{P}(\mathsf{Env})$
- $\propto \in \mathsf{Env} \to \mathsf{Val} \to \mathcal{P}(\mathsf{Val})$

# Linking, to Satisfy the Advance Lemma

- $\propto \in \mathsf{Env} \to \mathsf{Shadow} \to \mathcal{P}(\mathsf{Val})$

$$
\begin{array}{lcll}
\sigma_0 & \propto & \mathsf{Init} & \triangleq & \{\sigma_0\} \\
\sigma_0 & \propto & \mathsf{Read}(S, x) & \triangleq & \{v_+ \mid \sigma_+ \in \sigma_0 \propto S,\ \sigma_+(x) = v_+\}
\end{array}
$$

- $\propto \in \mathsf{Env} \to \mathsf{Env} \to \mathcal{P}(\mathsf{Env})$
- $\propto \in \mathsf{Env} \to \mathsf{Val} \to \mathcal{P}(\mathsf{Val})$

# Linking, to Satisfy the Advance Lemma

- $\infty \in \mathsf{Env} \to \mathsf{Shadow} \to \mathcal{P}(\mathsf{Val})$

$$
\begin{array}{lcll}
\sigma_0 & \infty & \mathsf{Init} & \triangleq \{\sigma_0\} \\
\sigma_0 & \infty & \mathsf{Read}(S, x) & \triangleq \{v_+ \mid \sigma_+ \in \sigma_0 \infty S,\ \sigma_+(x) = v_+\} \\
\sigma_0 & \infty & \mathsf{Call}(S, v) & \triangleq \\
\end{array}
$$
$$
\{v'_+ \mid \langle \lambda x.e, \sigma_+\rangle \in \sigma_0 \infty S,\ v_+ \in \sigma_0 \infty v,\ (x, v_+) :: \sigma_+ \vdash e \Downarrow v'_+\}
$$
$$
\cup \{\mathsf{Call}(S_+, v_+) \mid S_+ \in \sigma_0 \infty S,\ v_+ \in \sigma_0 \infty v\}
$$

- $\infty \in \mathsf{Env} \to \mathsf{Env} \to \mathcal{P}(\mathsf{Env})$
- $\infty \in \mathsf{Env} \to \mathsf{Val} \to \mathcal{P}(\mathsf{Val})$

# The Advance Lemma for Model Language L

(Coq-verified)

# The Advance Lemma for Model Language L

(Coq-verified)

- Defined its collecting semantics

$$\mathsf{eval}(e, \Sigma) \triangleq \bigcup_{\sigma^\dagger \in \Sigma} \{v^\dagger \mid \sigma^\dagger \vdash e \Downarrow v^\dagger\}.$$

# The Advance Lemma for Model Language L

(Coq-verified)

- Defined its collecting semantics

$$\mathsf{eval}(e, \Sigma) \triangleq \bigcup_{\sigma^\dagger \in \Sigma} \{v^\dagger \mid \sigma^\dagger \vdash e \Downarrow v^\dagger\}.$$

- Proved the *advance lemma*

$$\mathsf{eval}(e, \Sigma \propto \Sigma_0) \subseteq \Sigma \propto \mathsf{eval}(e, \Sigma_0)$$

hence follows

$$\mathsf{eval}(e_1 \rtimes e_2, \Sigma) \subseteq \mathsf{eval}(e_1, \Sigma) \propto \mathsf{eval}(e_2, \mathsf{Init}).$$

# The Advance Lemma for Model Language L

(Coq-verified)

- Defined its collecting semantics

$$\mathsf{eval}(e, \Sigma) \triangleq \bigcup_{\sigma^\dagger \in \Sigma} \{v^\dagger \mid \sigma^\dagger \vdash e \Downarrow v^\dagger\}.$$

- Proved the *advance lemma*

$$\mathsf{eval}(e, \Sigma \propto \Sigma_0) \subseteq \Sigma \propto \mathsf{eval}(e, \Sigma_0)$$

  hence follows

$$\mathsf{eval}(e_1 \rtimes e_2, \Sigma) \subseteq \mathsf{eval}(e_1, \Sigma) \propto \mathsf{eval}(e_2, \mathsf{Init}).$$

Sound modular analyses are:

- For sound $\mathsf{eval}^\#$ and $\propto^\#$,

$$\mathsf{eval}(e_1 \rtimes e_2, \Sigma) \subseteq \gamma(\mathsf{eval}^\#(e_1, \Sigma^\#) \propto^\# \mathsf{eval}^\#(e_2, \mathsf{Init}^\#)).$$

# Framework Instance

- Function-summary-based approach

# Framework Instance

- Function-summary-based approach
  - for closure
$$\langle \texttt{f(x) = } e, \Sigma^{\#} \rangle,$$
its summary $f^{\#}$ is
$$f^{\#} \triangleq \mathsf{eval}^{\#}(e, \mathsf{Init}^{\#}).$$

# Framework Instance

- Function-summary-based approach
  - for closure
    $$\langle \texttt{f(x) = } e, \Sigma^\# \rangle,$$
    its summary $f^\#$ is
    $$f^\# \triangleq \mathsf{eval}^\#(e, \mathsf{Init}^\#).$$
  - its call $f(e_1)$ under ${\Sigma_1}^\#$ is a linking:
    $$(\mathsf{eval}^\#(\mathtt{x} = e_1, {\Sigma_1}^\#) ::^\# \Sigma^\#) \propto^\# f^\#.$$

# Summing Up

Modular Analysis

- check: advance lemma for semantics and linking
- then follows: $\mathsf{eval}(e_1 \rtimes e_2, \Sigma) \subseteq \mathsf{eval}(e_1, \Sigma) \propto \mathsf{eval}(e_2, \mathsf{Init})$
- modular analysis: $\mathsf{eval}^\#(e_1, \Sigma^\#) \propto^\# \mathsf{eval}^\#(e_2, \mathsf{Init}^\#)$

# Summing Up

Modular Analysis

- check: advance lemma for semantics and linking
- then follows: $\mathsf{eval}(e_1 \rtimes e_2, \Sigma) \subseteq \mathsf{eval}(e_1, \Sigma) \propto \mathsf{eval}(e_2, \mathsf{Init})$
- modular analysis: $\mathsf{eval}^{\#}(e_1, \Sigma^{\#}) \propto^{\#} \mathsf{eval}^{\#}(e_2, \mathsf{Init}^{\#})$

# Summing Up

Modular Analysis

- check: advance lemma for semantics and linking
- then follows: $\mathsf{eval}(e_1 \rtimes e_2, \Sigma) \subseteq \mathsf{eval}(e_1, \Sigma) \propto \mathsf{eval}(e_2, \mathsf{Init})$
- modular analysis: $\mathsf{eval}^\#(e_1, \Sigma^\#) \propto^\# \mathsf{eval}^\#(e_2, \mathsf{Init}^\#)$

## Summing Up

Modular Analysis

- check: advance lemma for semantics and linking
- then follows: $\mathsf{eval}(e_1 \rtimes e_2, \Sigma) \subseteq \mathsf{eval}(e_1, \Sigma) \propto \mathsf{eval}(e_2, \mathsf{Init})$
- modular analysis: $\mathsf{eval}^{\#}(e_1, \Sigma^{\#}) \propto^{\#} \mathsf{eval}^{\#}(e_2, \mathsf{Init}^{\#})$

# Summing Up

Modular Analysis

- check: advance lemma for semantics and linking
- then follows: $\text{eval}(e_1 \rtimes e_2, \Sigma) \subseteq \text{eval}(e_1, \Sigma) \propto \text{eval}(e_2, \text{Init})$
- modular analysis: $\text{eval}^{\#}(e_1, \Sigma^{\#}) \propto^{\#} \text{eval}^{\#}(e_2, \text{Init}^{\#})$

$\boxed{\text{Thank you}}$