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: $f : L \to L$

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

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

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

Prove $\hat{f}$ simulates f under α .

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 $\hat{f}$ simulates f under α .

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 α .

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: $\hat{f} : \hat{L} \to 2^{\hat{L}}$

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 α.

Illustrating the Standard Recipe

Malloc: The Language

v `:= malloc()`

Malloc: The Language

lab : v := `malloc()`

Concrete Semantics

$$State = Instruction \times Store$$

Concrete Semantics

$$State = Instruction \times Store$$

$$f(\varsigma) = \varsigma'$$

Concrete Semantics

$$State = Instruction \times Store$$

Fresh

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

Concrete Semantics

$$State = Instruction \times Store$$

Fresh

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

$$a' = \mathrm{alloc}(\varsigma)$$

Concrete Semantics

$$State = Instruction \times Store$$

Fresh

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

$$a' = \text{alloc}(\varsigma) = \max(\text{range}(\sigma)) + 1$$

Abstract Semantics

$$\widehat{State} = Instruction \times \widehat{Store}$$

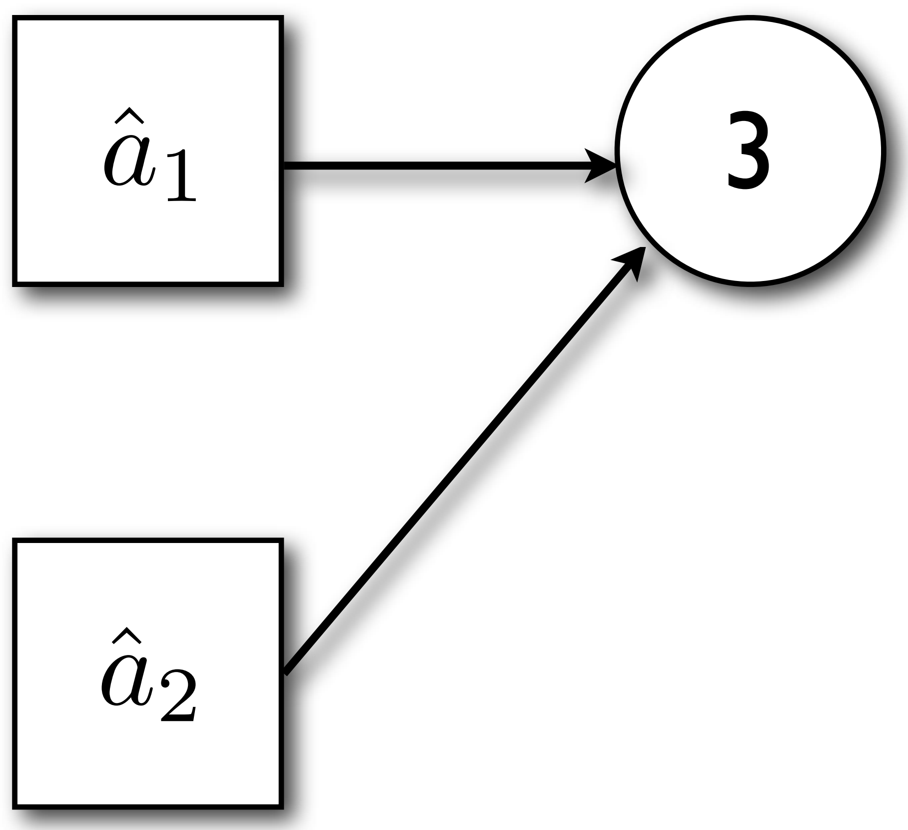
$$\hat{f}([\![\texttt{v := malloc()}]\!] : \vec{i}, \hat{\sigma}) = (\vec{i}, \hat{\sigma}[v \mapsto \hat{a}])$$

$\hat{a} = \widehat{\mathrm{alloc}}(\hat{\varsigma})$ (from some finite set)

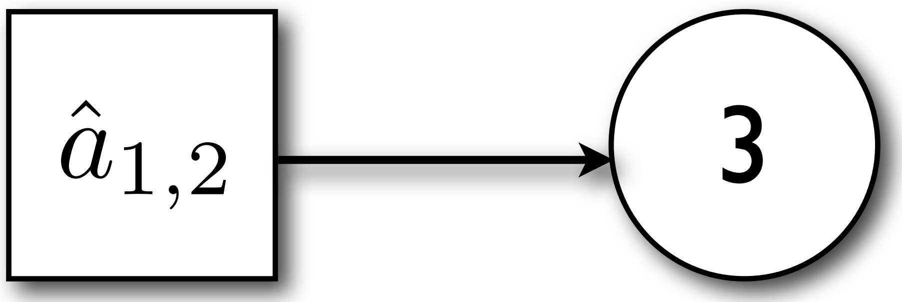
What to allocate?

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

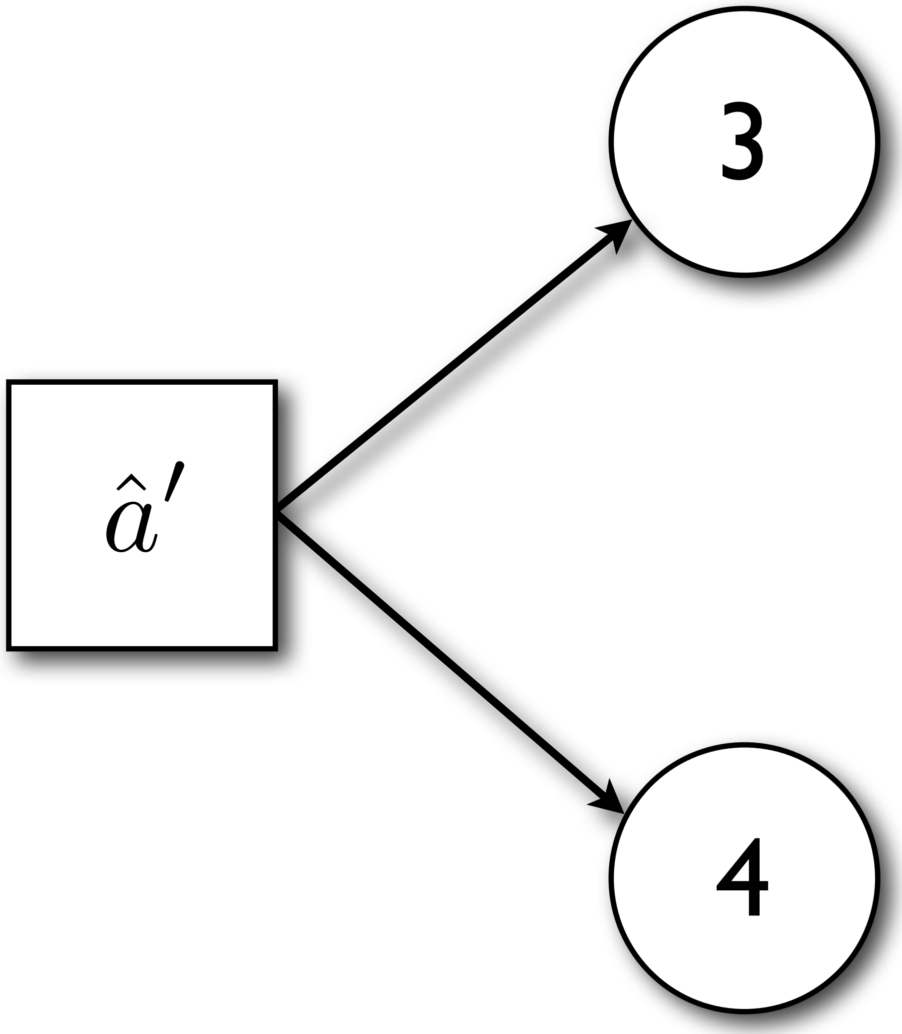
Example: Over-allocation



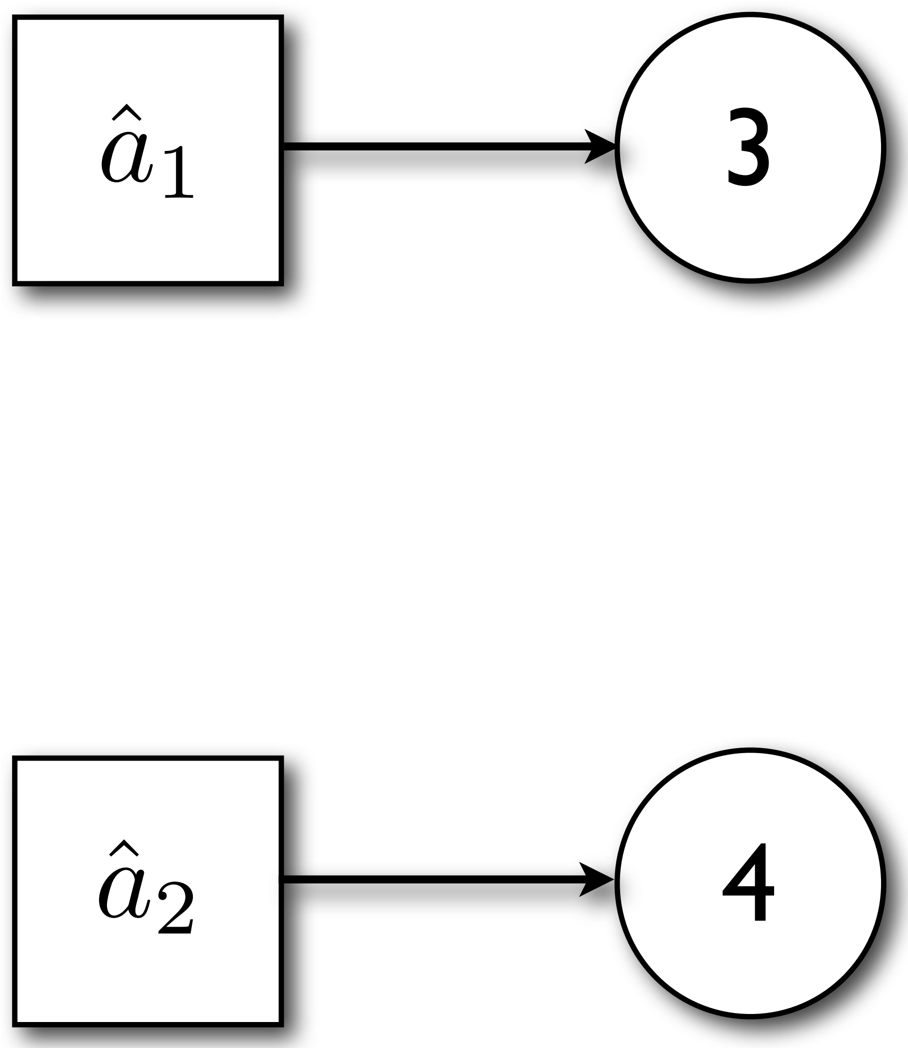
Example: Over-allocation



Example: Under-allocation



Example: Under-allocation



Allocation heuristics

Observation: Objects from like contexts act alike.

Allocation heuristics

Observation: Objects from like contexts act alike.

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

Annoyance: Soundness

If

$$\alpha(\varsigma) \sqsubseteq \hat{\varsigma}$$

then

$$\alpha(f(\varsigma)) \sqsubseteq \hat{f}(\hat{\varsigma})$$

Annoyance: Soundness

If

$$\alpha(\varsigma) \sqsubseteq \hat{\varsigma}$$

then

$$\alpha_{Addr}(\text{alloc}(\varsigma)) \sqsubseteq \widehat{\text{alloc}}(\hat{\varsigma})$$

The Issue

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

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

What abstraction map will work here?

Example

```
A : x := malloc()
B : y := malloc()
```

$$\sigma = [\mathsf{x} \rightarrow 1, \mathsf{y} \rightarrow 2]$$

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

Example

```
B : y := malloc()
A : x := malloc()
```

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

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

Example

```
B : y := malloc()
A : x := malloc()
```

$\sigma = [\mathsf{x} \rightarrow 2, \mathsf{y} \rightarrow 1]$

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

Example

```
B : y := malloc()
A : x := malloc()
```

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

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

Standard Solution

Change the concrete semantics!

Standard Solution

Change the concrete semantics!

$$Addr = \mathbb{N}$$

$$\mathrm{alloc}(\_, \sigma) = \max(\mathrm{range}(\sigma)) + 1$$

Standard Solution

Change the concrete semantics!

$$Addr = \mathbb{N} \times Lab$$

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

Standard Solution

Change the concrete semantics!

$$Addr = \mathbb{N} \times Lab$$

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

$$\alpha(\_, lab) = lab$$

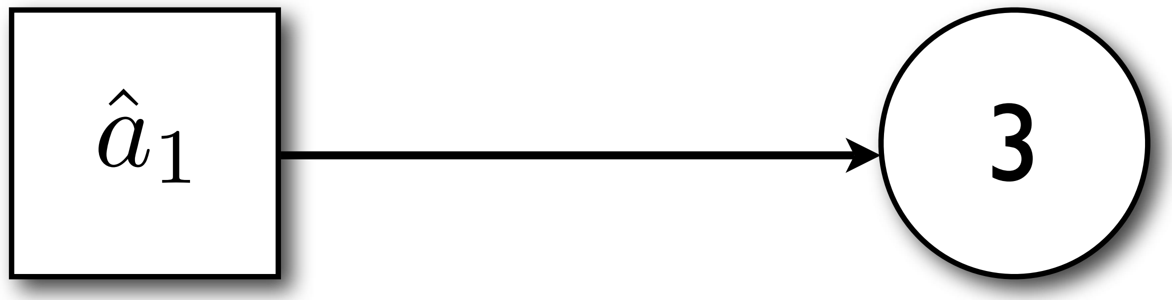
Another problem: Heuristics sometimes make stupid decisions

Another problem: Heuristics sometimes make stupid decisions

Why not adapt on the fly?

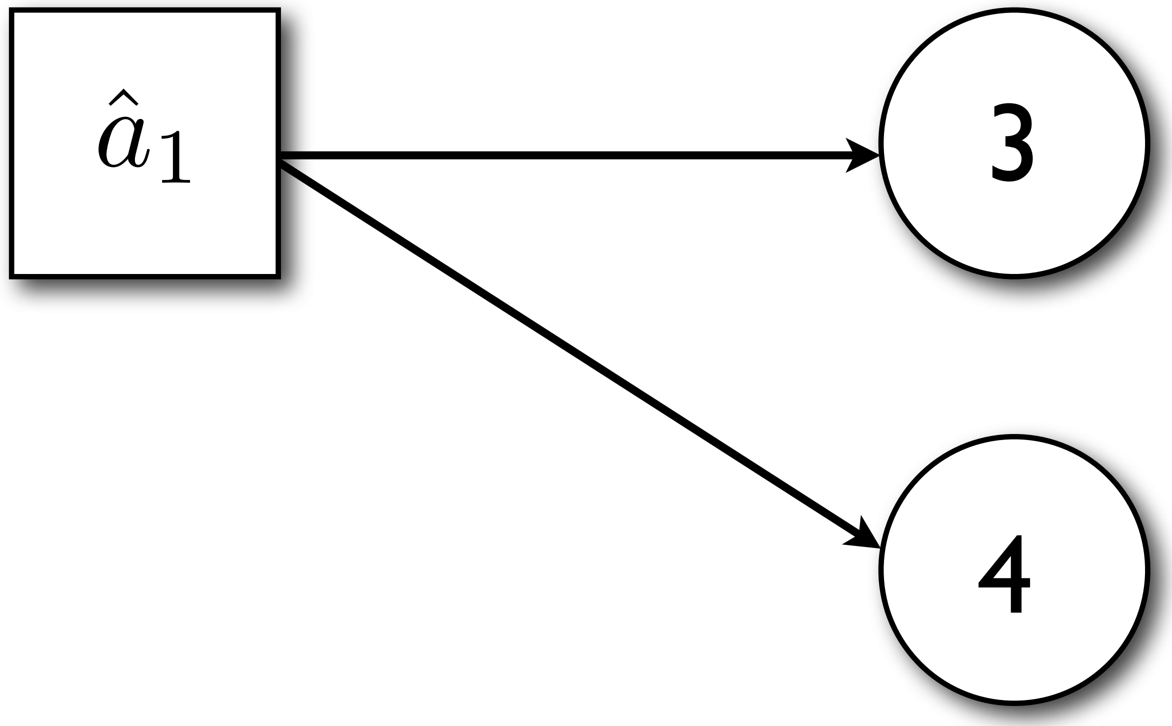
Example: Greedy Strategy

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



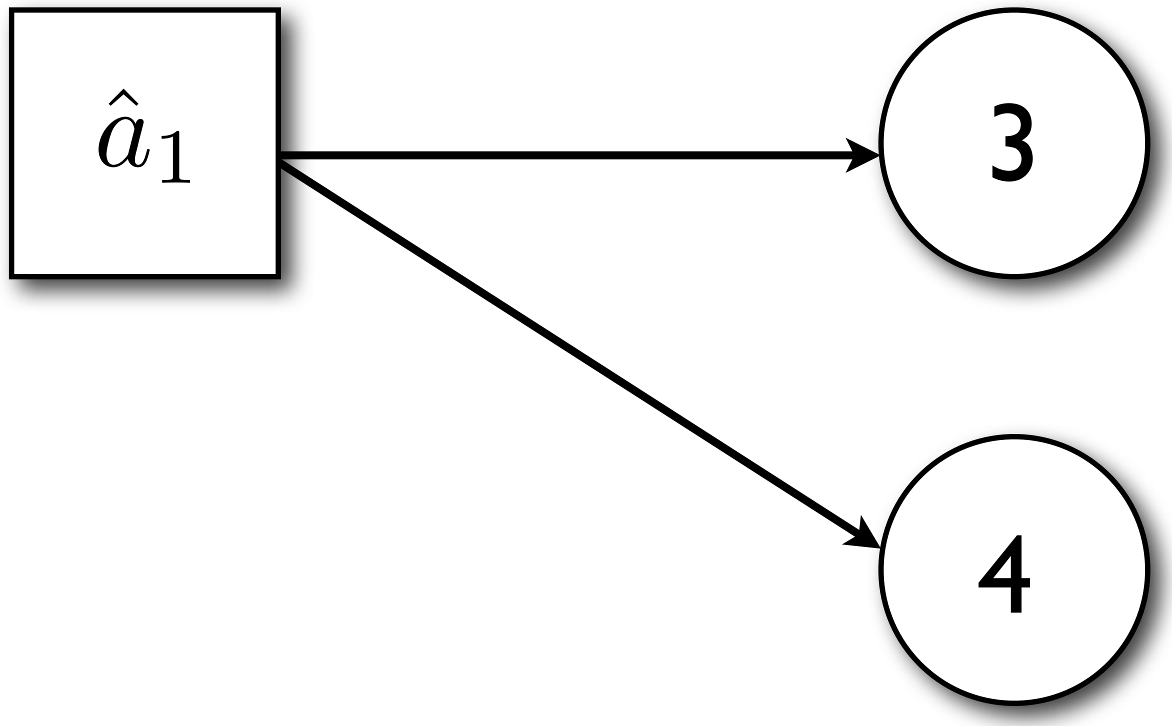
Example: Greedy Strategy

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



Example: Greedy Strategy

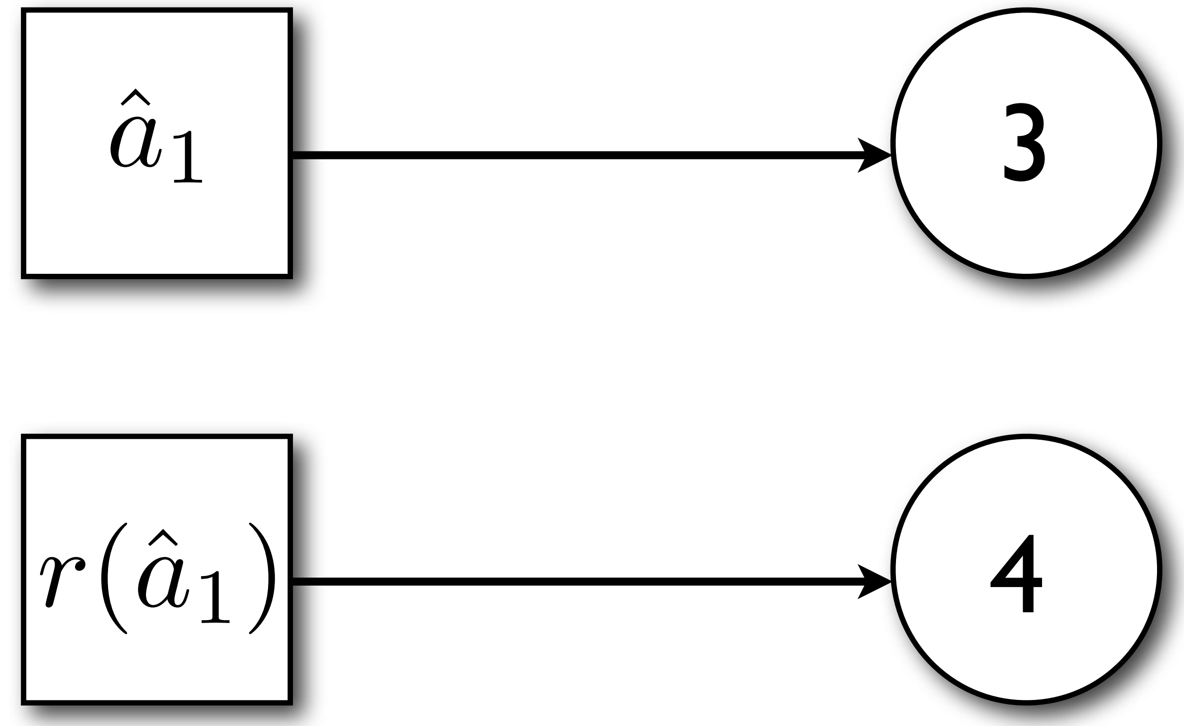
Heuristic says, "Allocate $\hat{a}_1$, and bind 4."



Adaptive allocator says, "Try $r(\hat{a}_1)$ first."

Example: Greedy Strategy

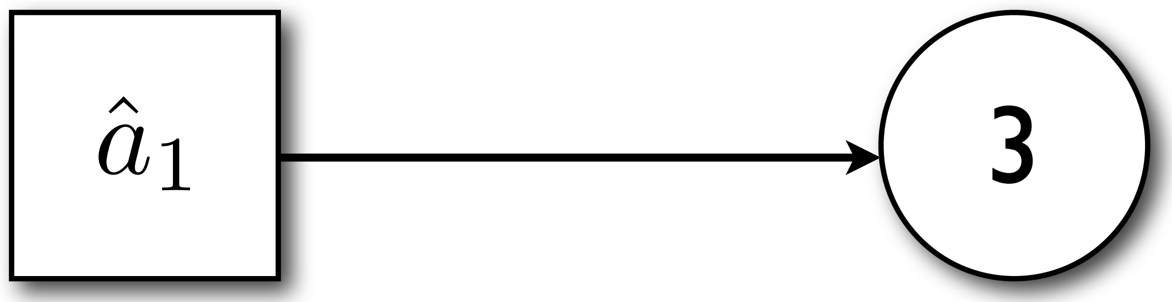
Heuristic says, "Allocate $\hat{a}_1$, and bind 4."



Adaptive allocator says, "Try $r(\hat{a}_1)$ first."

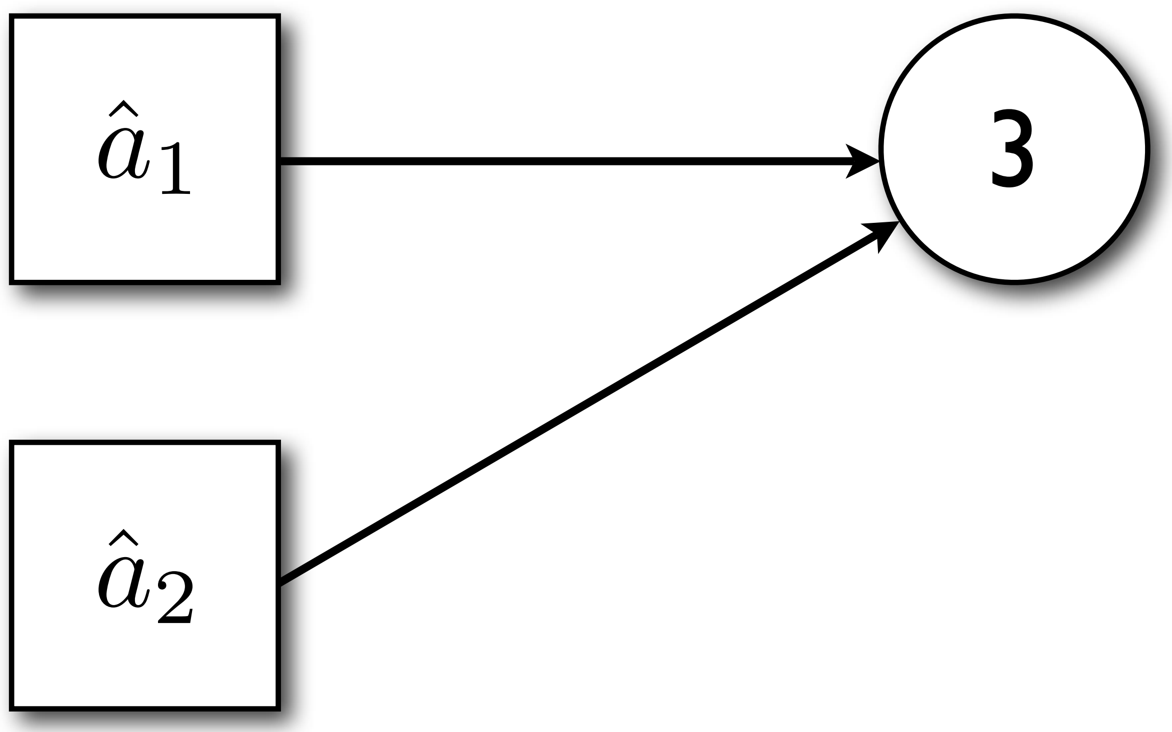
Example: Greedy Strategy

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



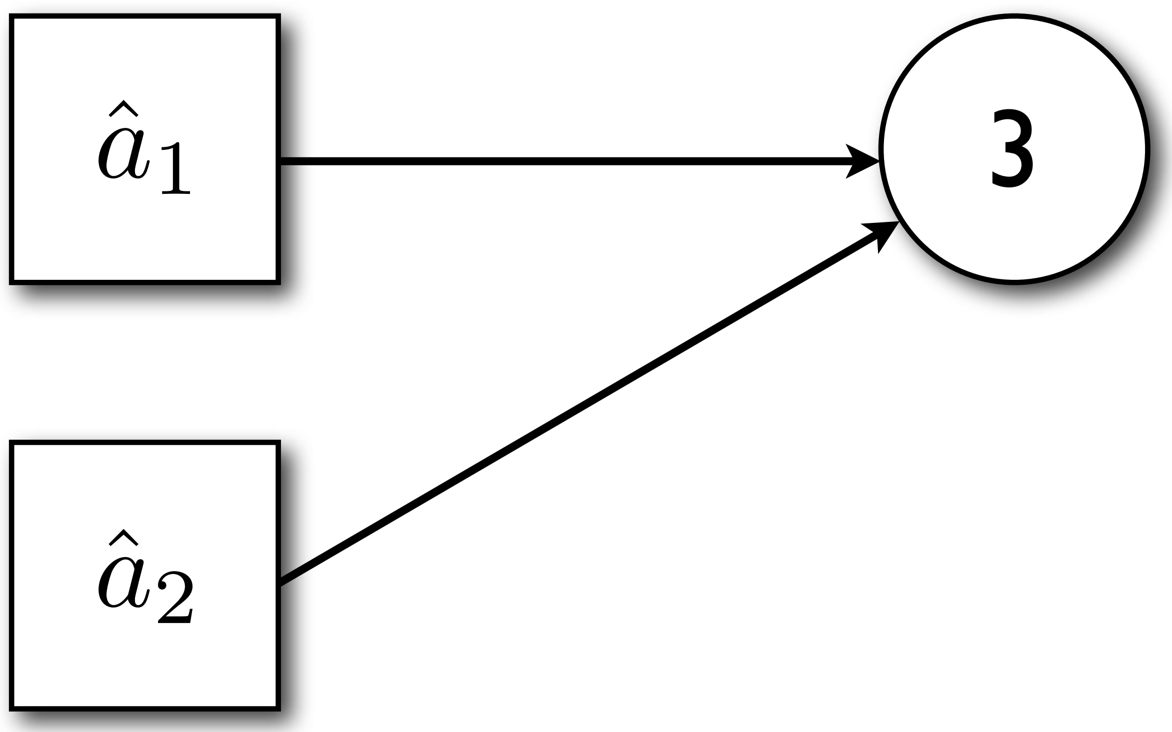
Example: Greedy Strategy

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



Example: Greedy Strategy

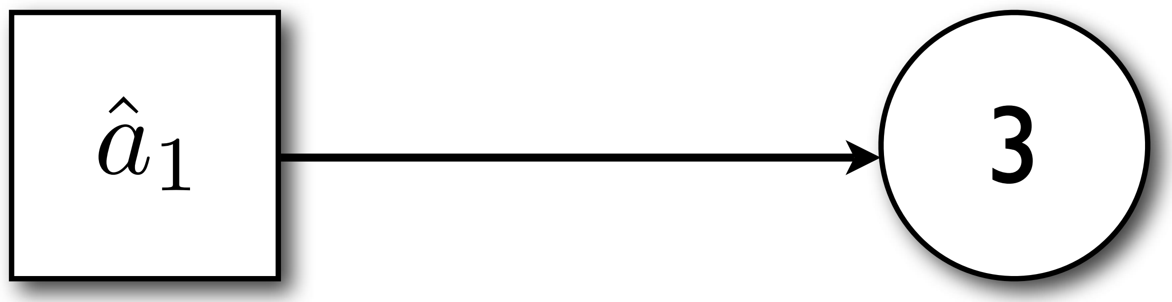
Heuristic says, "Allocate $\hat{a}_2$, and bind 3."



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 $\hat{a}_1$."

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

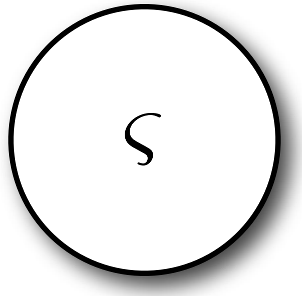
- Factor allocation out of semantics.
- Make allocation nondeterministic.
- Prove nondeterministic allocation sound.

Locative = Address

(But also times, bindings, contours, *etc.*)

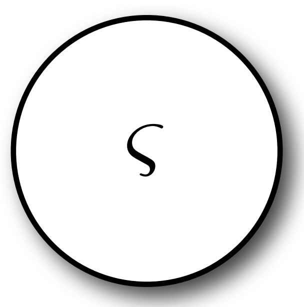
Factoring out allocation

$$f : State \rightarrow State$$

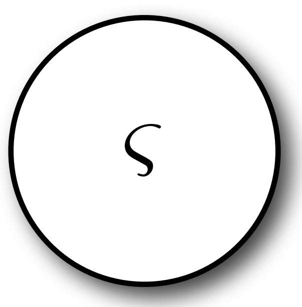


$$f : State \to State$$

$$f : State \rightarrow State$$

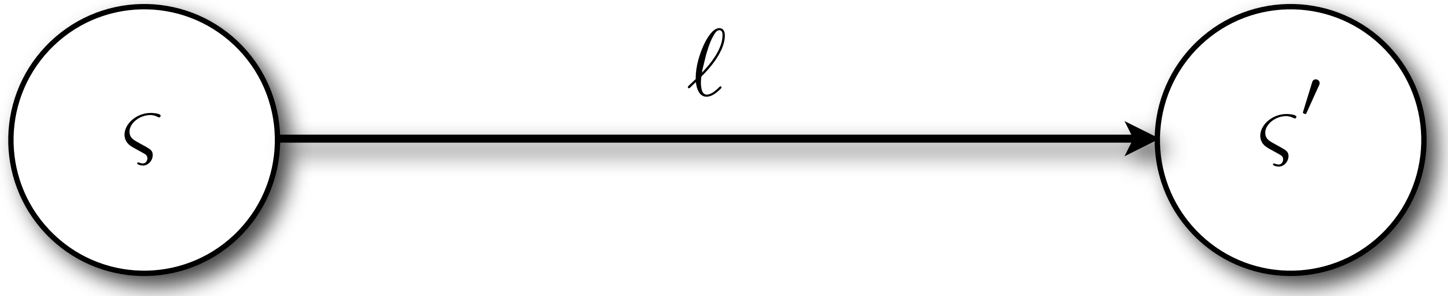


$$F : State \rightarrow Loc \rightarrow State$$

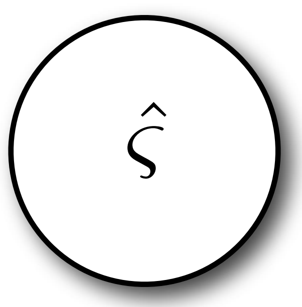


$$F : State \rightarrow Loc \rightarrow State$$

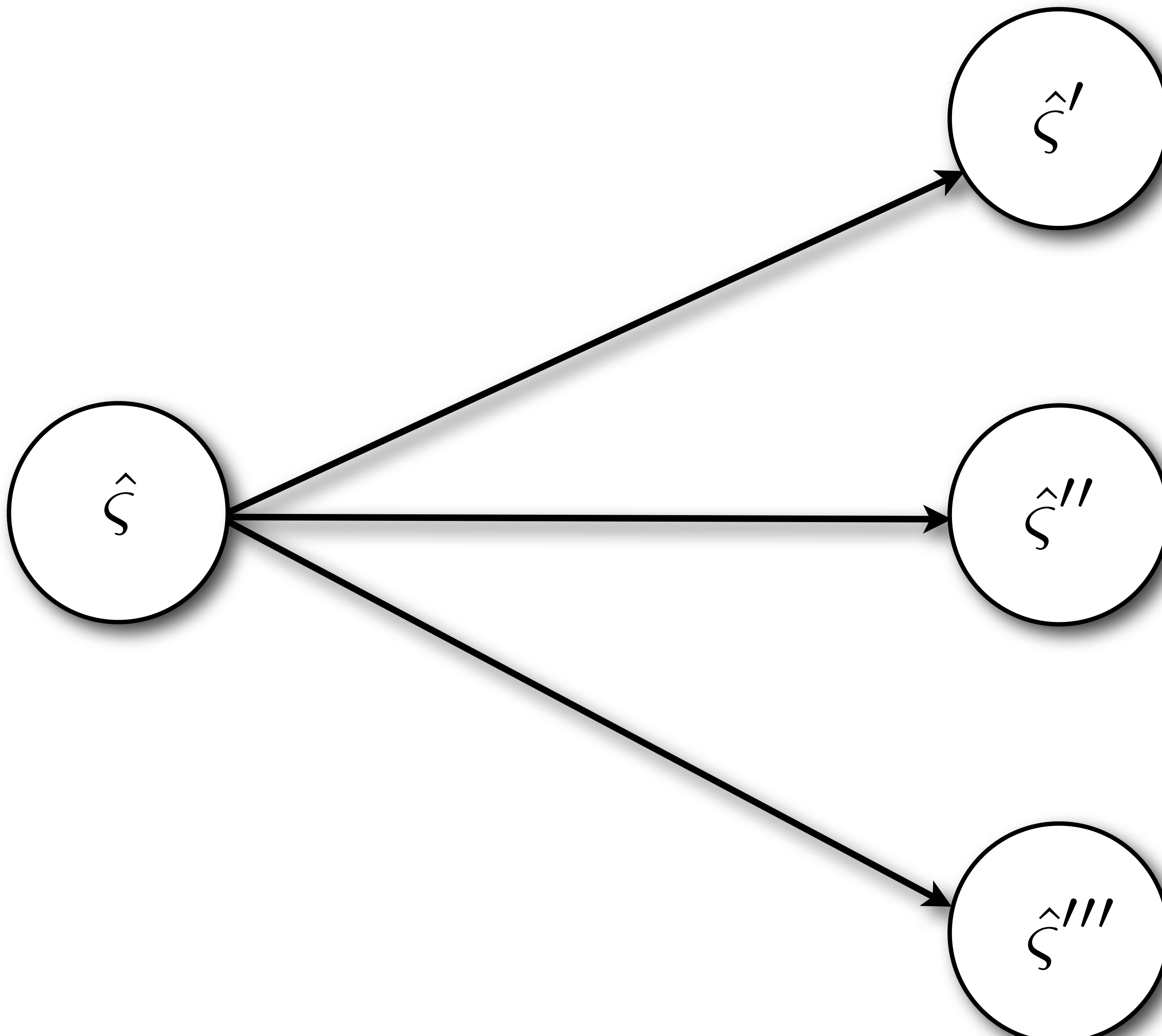
$$F : State \to Loc \to State$$



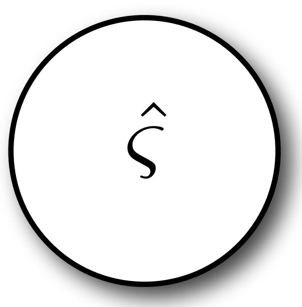
$$\hat{f} : \widehat{State} \rightarrow 2^{\widehat{State}}$$



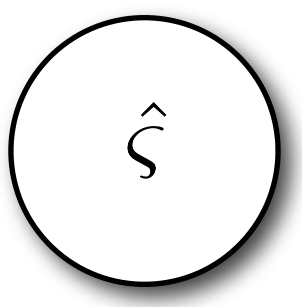
$$\hat{f} : \widehat{State} \to 2^{\widehat{State}}$$



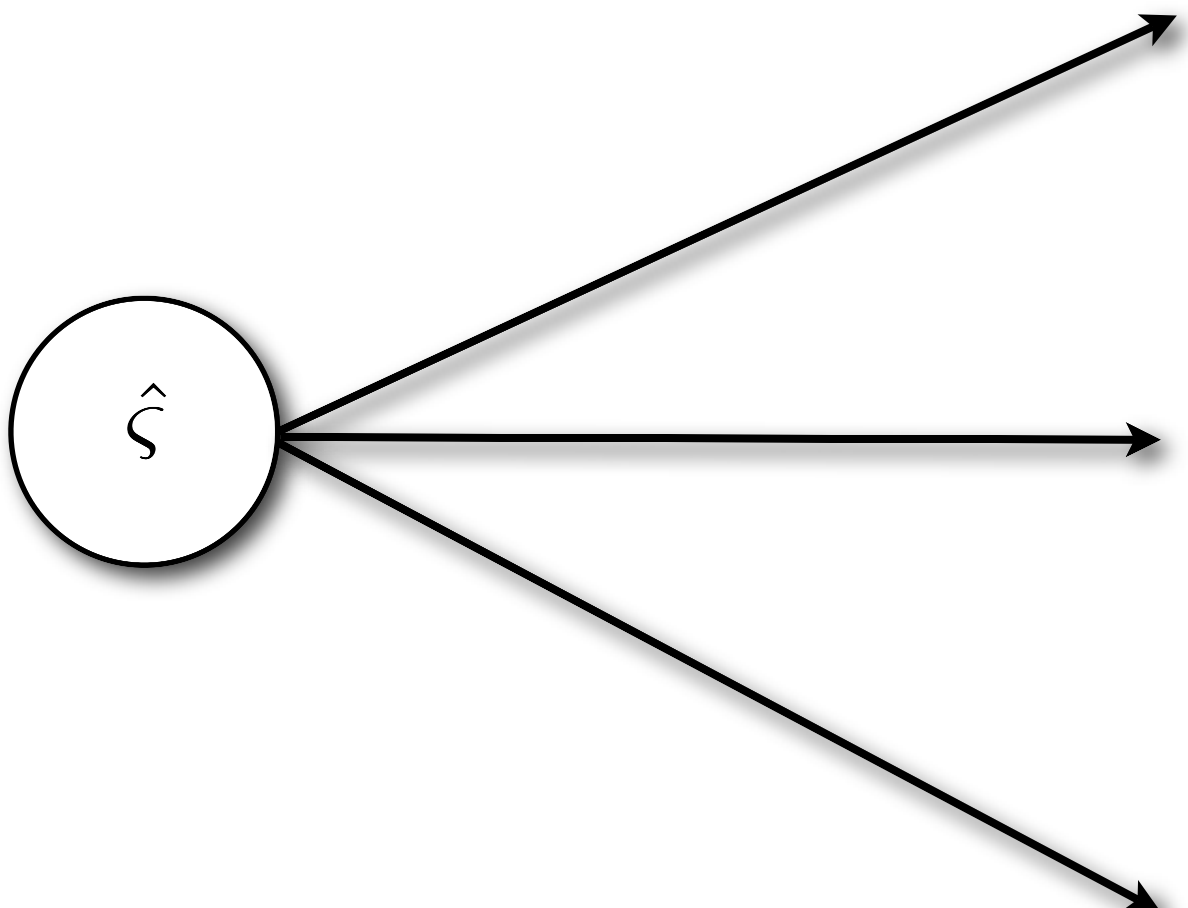
$$\hat{f} : \widehat{State} \rightarrow 2^{\widehat{State}}$$



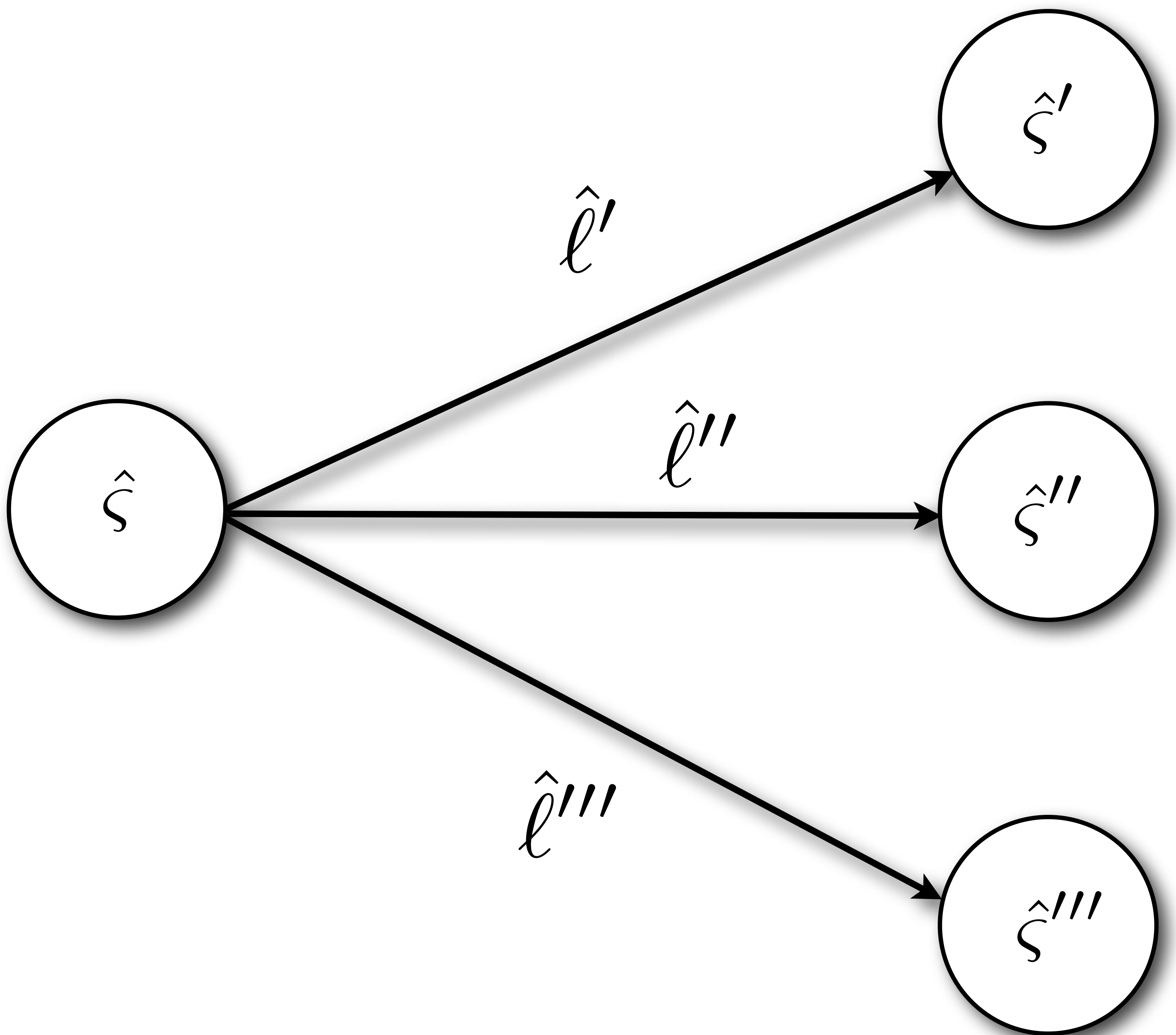
$$\hat{F} : \widehat{State} \rightarrow 2^{\widehat{Loc} \rightarrow \widehat{State}}$$



$$\hat{F} : \widehat{State} \rightarrow 2^{\widehat{Loc} \rightarrow \widehat{State}}$$



$$\hat{F} : \widehat{State} \to 2^{\widehat{Loc} \to \widehat{State}}$$



Nondeterministic Abstract Interpretation

Nondeterministic Abstract Interpretation

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

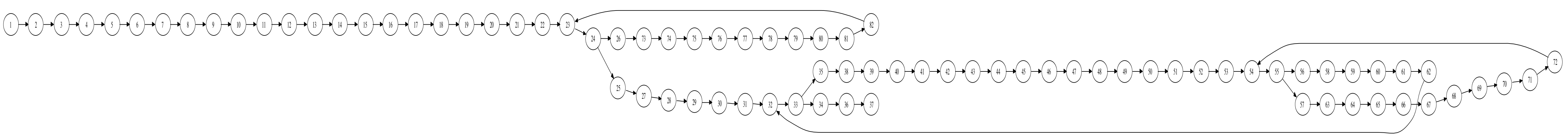
Transition Graphs

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

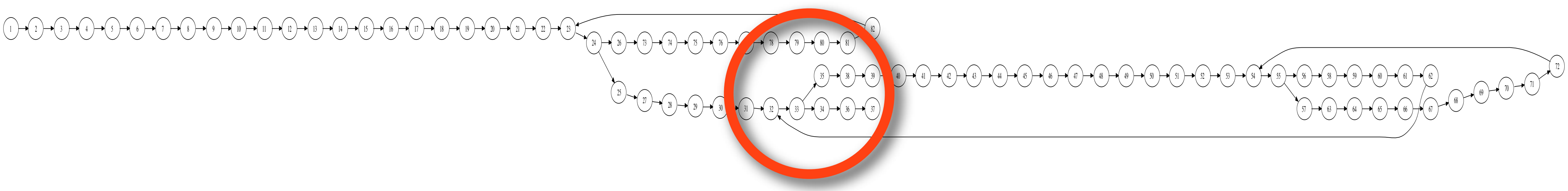
Sealed Graphs

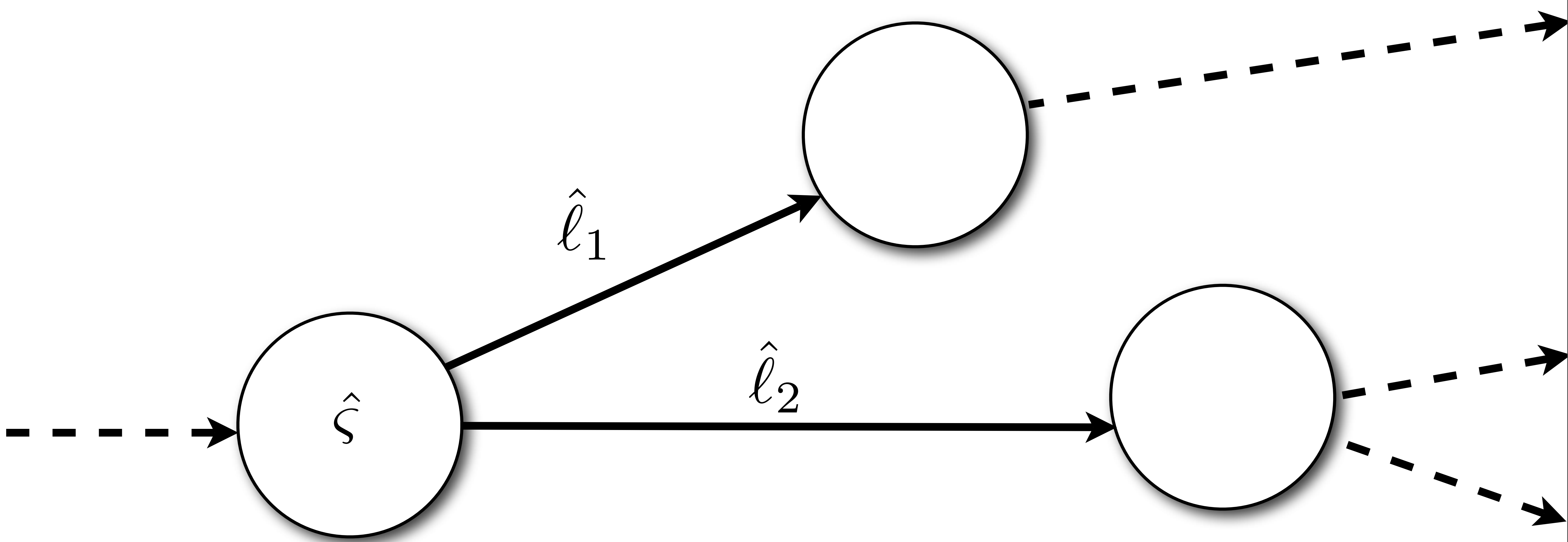
Graph is **sealed** under factored semantics iff every state has an edge to cover every transition.

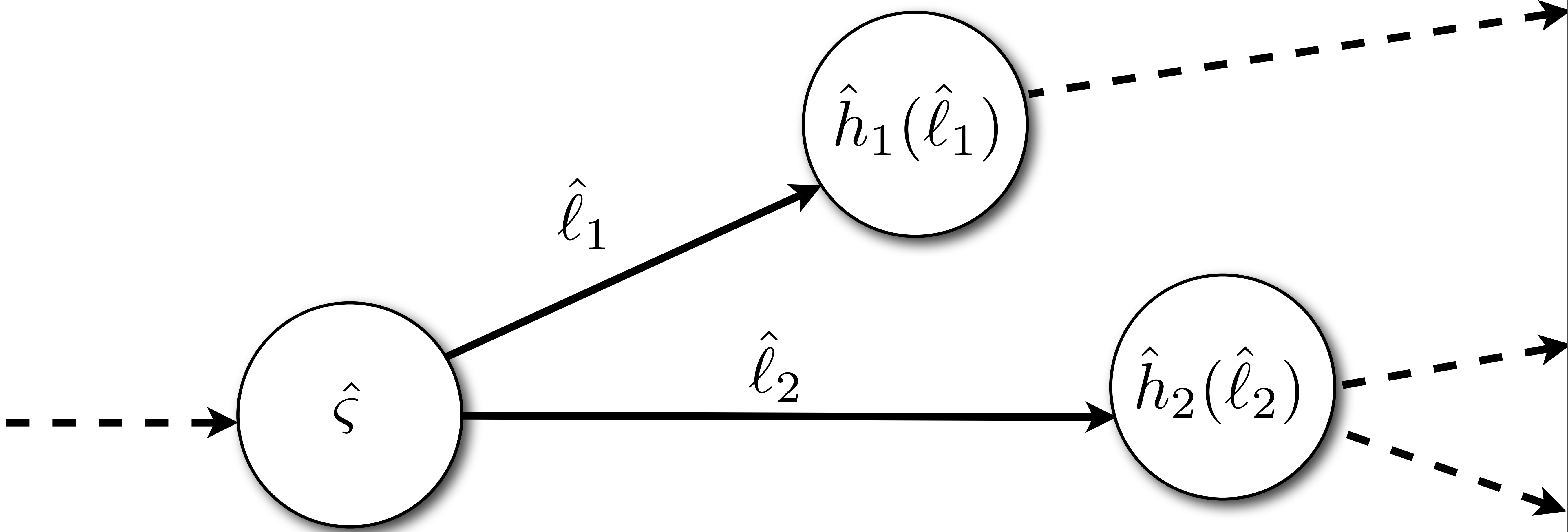
Example: *Un*sealed Graph



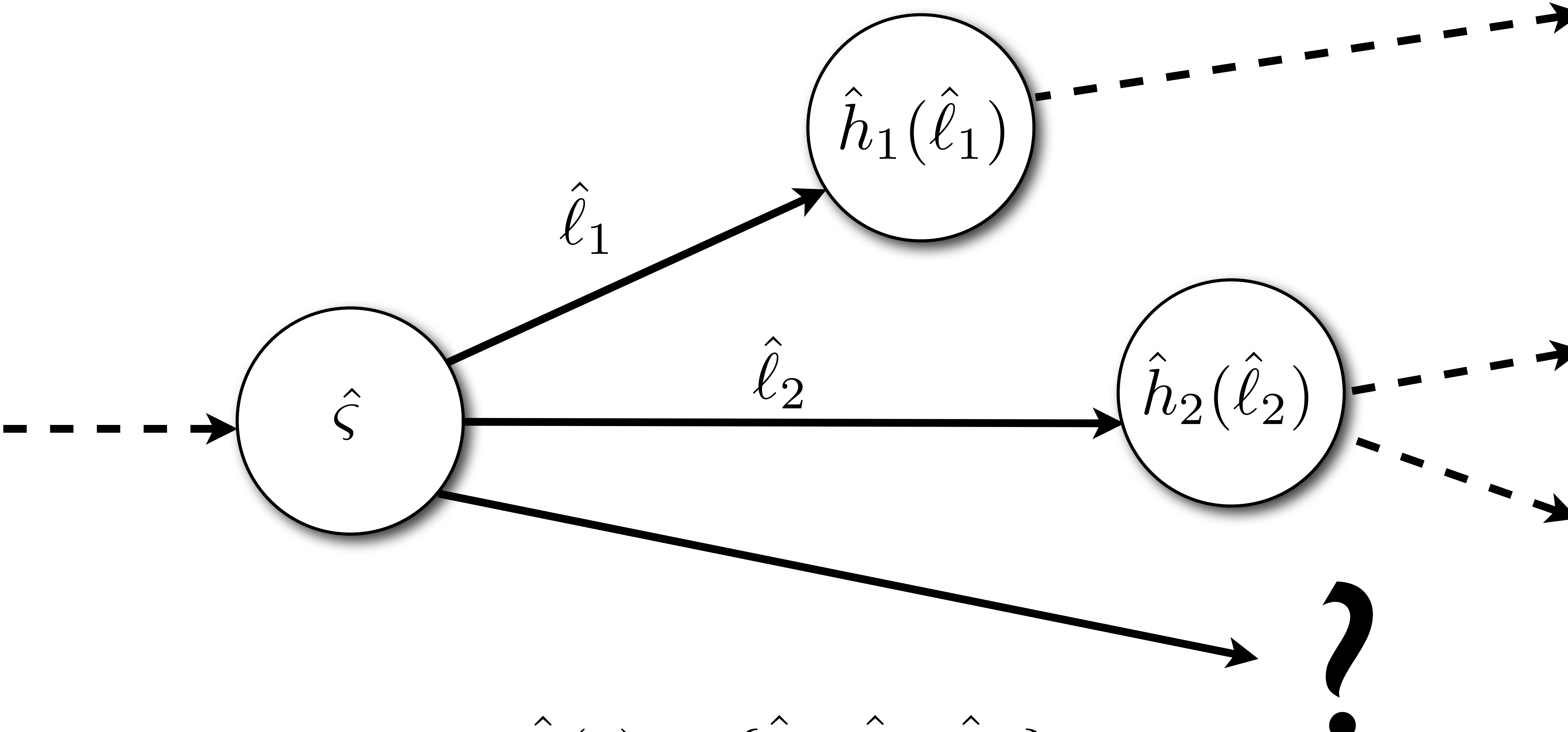
Example: *Un*sealed Graph




$\hat{\ell}_1$
$\hat{\ell}_2$
$\hat{\varsigma}$



$$\hat{F}(\hat{\varsigma}) = \{\hat{h}_1, \hat{h}_2, \hat{h}_3\}$$



$$\hat{F}(\hat{\varsigma}) = \{\hat{h}_1, \hat{h}_2, \hat{h}_3\}$$

Proving Sealed Graphs Sound

Factoring Abstraction

$$\alpha : State \rightarrow \widehat{State}$$

Factoring Abstraction

$$\alpha : State \to \widehat{State}$$

$$\beta : (Loc \to \widehat{Loc}) \to (State \to \widehat{State})$$

Dependent Simulation

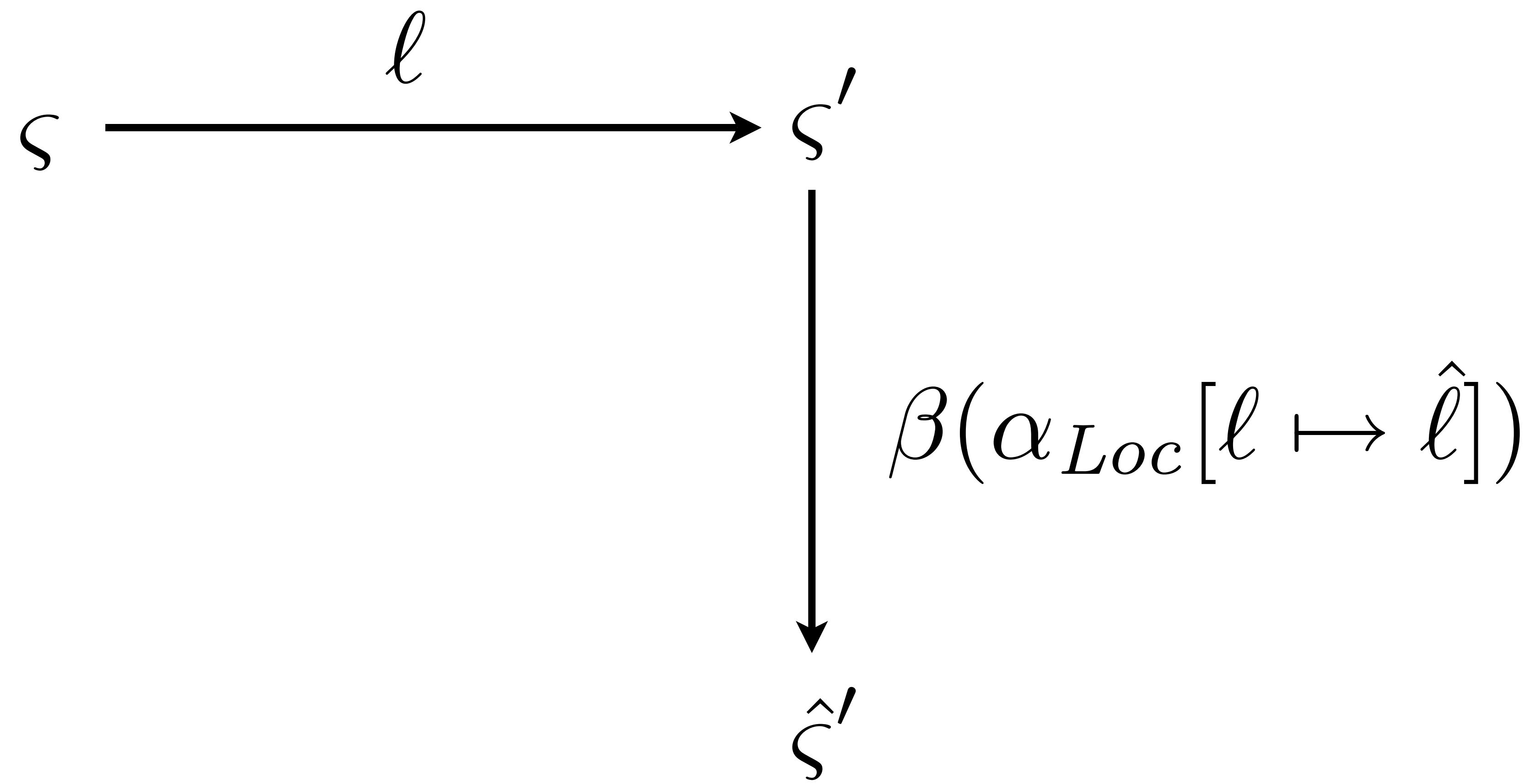
Dependent Simulation

ς

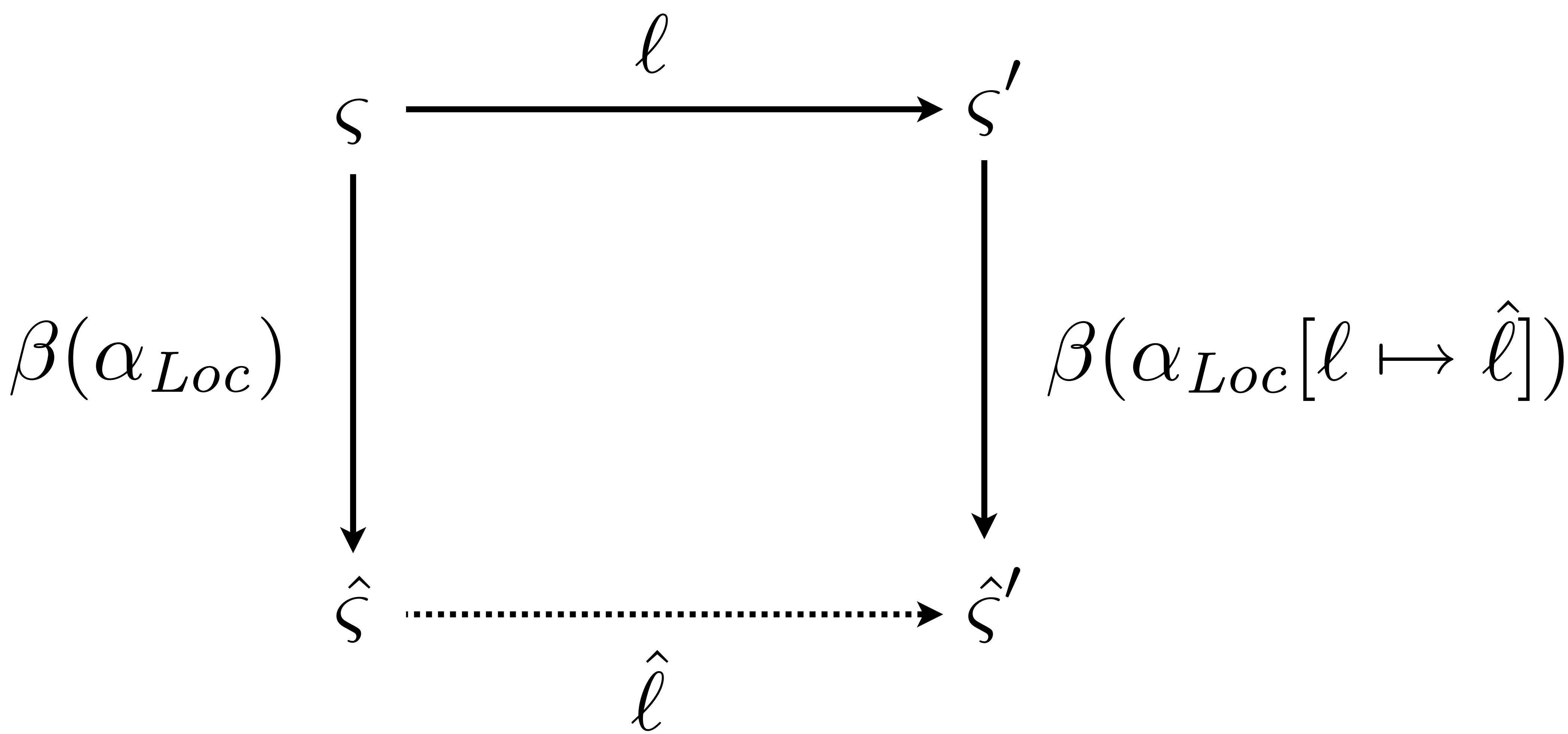
Dependent Simulation

$$\varsigma \xrightarrow{\ell} \varsigma'$$

Dependent Simulation



Dependent Simulation



A Posteriori Theorem

Dependent simulation → 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: ∃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