Control-flow analysis of higher-order programs

Matt Might
University of Utah
matt.might.net

“I’m on a mission from *God!*”

Olin Shivers, PLDI 1988

The mission

- CFA is easy
- CFA is useful
- CFA is deep

Detailed outline

- Background
- Methods
- Applications

Tentative agenda

- Today: Background, 0CFA $\times$ 3
- Friday: *k*-CFA, 1CFA, poly/CFA, ...
- Monday: Applications, Beyond CFA

Background

What is *higher-order*?

What is *higher-order*?

A language is **higher-order** if a procedure may
(i) accept procedures as arguments; and/or
(ii) yield procedures as return values.

-Wikipedia

What makes analyzing higher-order programs hard?

Why is higher-orderness hard?

Why is higher-orderness hard?

- Data-flow depends on control-flow

Why is higher-orderness hard?

- Data-flow depends on control-flow
- Control-flow depends on data-flow

Which kind of higher-orderness is **really** hard?

Which kind of higher-orderness is **really** hard?

- Value = Code × Data

Which kind of higher-orderness is **really** hard?

- Value = Code × Data
- Closure = Lambda × Env

Which kind of higher-orderness is **really** hard?

- Value = Code $\times$ Data
- Closure = Lambda $\times$ Env
- Object = Class $\times$ Record

Languages of interest

- Lisp
- Scheme
- ML
- Haskell
- Smalltalk
- Perl
- Python
- Ruby
- C++
- Java
- C#
- ...

Control-flow analysis

Control-flow terminology

- Control-flow analysis of higher-order programs
- Higher-order control-flow analysis
- Control-flow analysis
- Closure analysis
- CFA

Control-flow problem

Control-flow problem

Given a call site `(f e`$_1$` ... e`$_n$`)`, what is `f`?

Control-flow problem

Given a call site `(f e1 ... en)`, what is `f`?

Given a call site `o.m(e1,...,en)`, what is `o`?

Control-flow scenarios

`f(x)`

Control-flow scenarios

```
let f = λz.z
     in f(x)
```

Control-flow scenarios

```
λf.f(x)
```

Example

```
apply f x = f(x)
```

Example

```
apply f x = f(x)
```

What procedures are called here?

Example

```
apply f x = f(x)
```

```
h(x) = x + 1

apply h 4
```

$$\mathrm{FlowsTo}[\mathtt{f}] = \{\mathtt{h}\}$$

Example

```
apply f x = f(x)

h(x) = x + 1                g(x) = apply h 4

                            apply g 3
```

$$\mathrm{FlowsTo}[\mathsf{f}] = \{\mathsf{h},\mathsf{g}\}$$

Example

```
apply f x = f(x)
```

```
(apply apply apply) apply
```

Example

```
apply f x = f(x)

(apply apply apply) apply
```

Why compute CFA?

The 80/20 rule

80% of benefit comes from:

- Inlining procedures
- Constant propagation
- Register allocation

…and the other 80%

- Classic data-flow analysis
- Data-flow optimizations
- Argument globalization
- Defunctionalization
- Static method resolution
- Model-checking
- Global constant propagation
- Global copy propagation
- Continuation promotion
- Loop detection/optimization
- Lightweight closure conversion
- Super-beta optimizations
- Escape analysis
- ...

Control-flow analysis

A **control-flow analysis** conservatively approximates the procedures which may be invoked at a given call site.

Control-flow analysis

A **control-flow analysis** conservatively approximates the procedures which may be invoked at a given call site.

A **value-flow analysis** conservatively approximates the values to which an expression may evaluate.

Example CFA answers

- Precise: $f \in \{\lambda x.x + 8\}$
- Coarse: $\hat{f} \in \{[\![\lambda x.x + z]\!], [\![\lambda x.x \times z]\!]\}$
- Intermediate: $f \in \{\lambda x.x + z \mid z \in \mathbb{N}_2\}$

Environment matters

$\lambda x.x$

$f(x)$

$\lambda x.z \times x$

Why understand CFA?

- Not enough to run CFA then do first-order analysis
- Need to integrate first-order analysis into CFA

Crude history

- 1981: Flow analysis of lambdas (Jones)
- 1988: 0CFA (Shivers)
- 1991: 1CFA, *k*-CFA (Shivers)
- 1992: Equality CFA (Heinglen)
- 1995: Constraint-based CFA (Palsberg)
- 1995-2005: *n* different values of *k*
- 2006: Environment analysis (Might, Shivers)

Midtgaard, 2008

- Exhaustive CFA literature review
- **171** entries in the bibliography!

Van Horn, 2009

- Exhaustive theoretical analysis
- (k>0)CFA is EXPTIME-complete
- 0CFA is $O(n^3/\lg n)$

Techniques for CFA

- *Ad hoc* techniques
- Constraint-solving
- Type-based analysis
- Abstract interpretation

Techniques for CFA

- *Ad hoc* techniques
- Constraint-solving
- Type-based analysis
- Abstract interpretation

Ad hoc CFAs

Null-CFA

Given a call site $f(x)$, what is f?

Null-CFA

Given a call site $f(x)$, what is f?

$$\top$$

Type-CFA

Given a call site $(f : t)(x)$, what is f?

Type-CFA

Given a call site $(f : t)(x)$, what is f?

All functions with the type t.

Arity-CFA

Given a call site $f(x_1,x_2,x_3)$, what is f?

Arity-CFA

Given a call site $f(x_1,x_2,x_3)$, what is f?

All functions with arity 3.

Smalltalk-CFA (Spoon, 2005)

Given a call site
`life setMeaning: 42`,
what is `setMeaning`?

First-order closure analysis

- Walk over the syntax tree
- Mark first-order call sites

```
let f(x) = x
 in f(3)
```

(Try this with monomorphization!)

Robust & efficient: 0CFA

What is 0CFA?

Lambda-flow analysis.

The 0CFA approximation

- Value = Code x Data
- Closure = Lambda x Env
- Object = Class x Record

The 0CFA approximation

- Value = Code
- Closure = Lambda
- Object = Class

0CFA

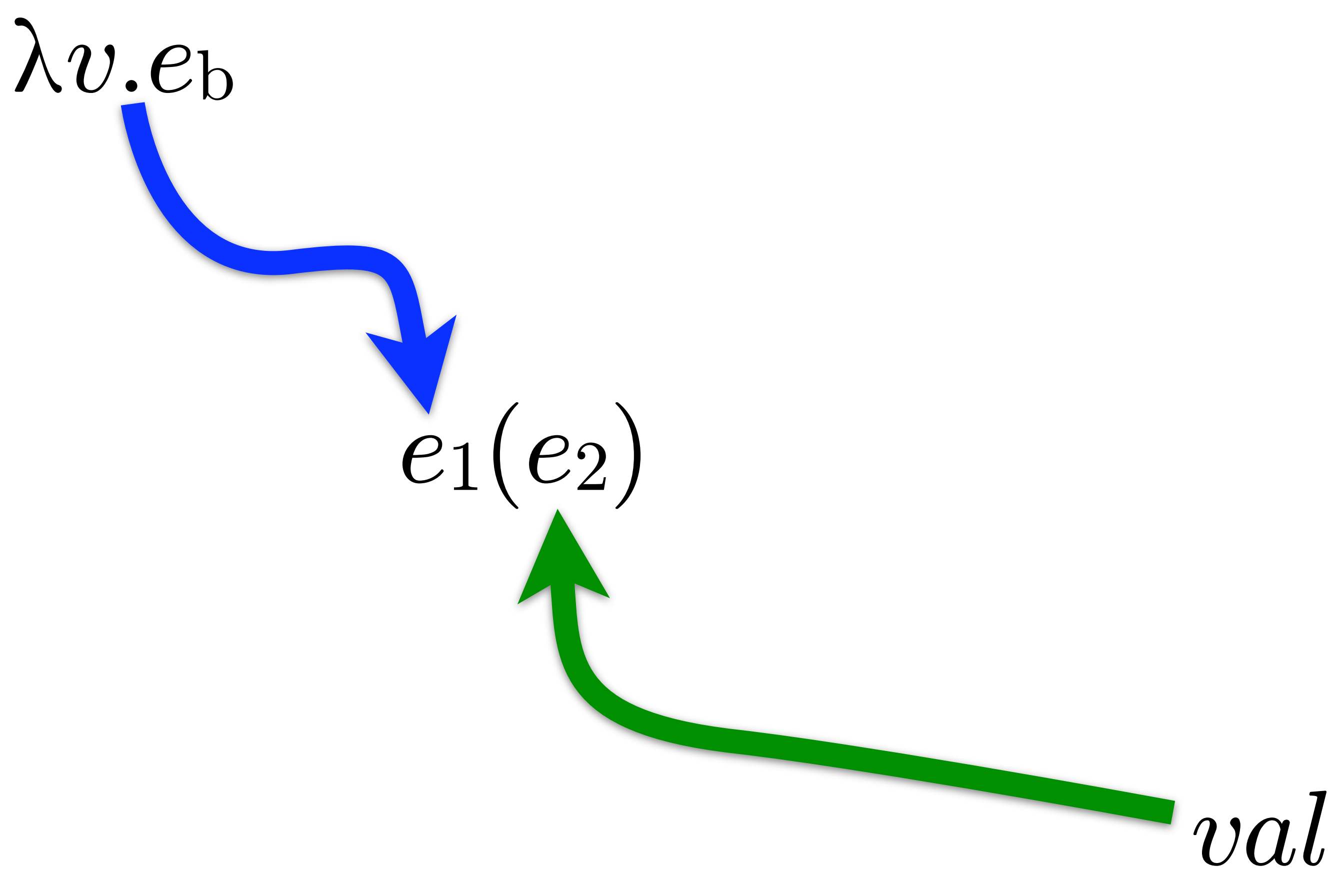
$$e_1(e_2)$$

0CFA

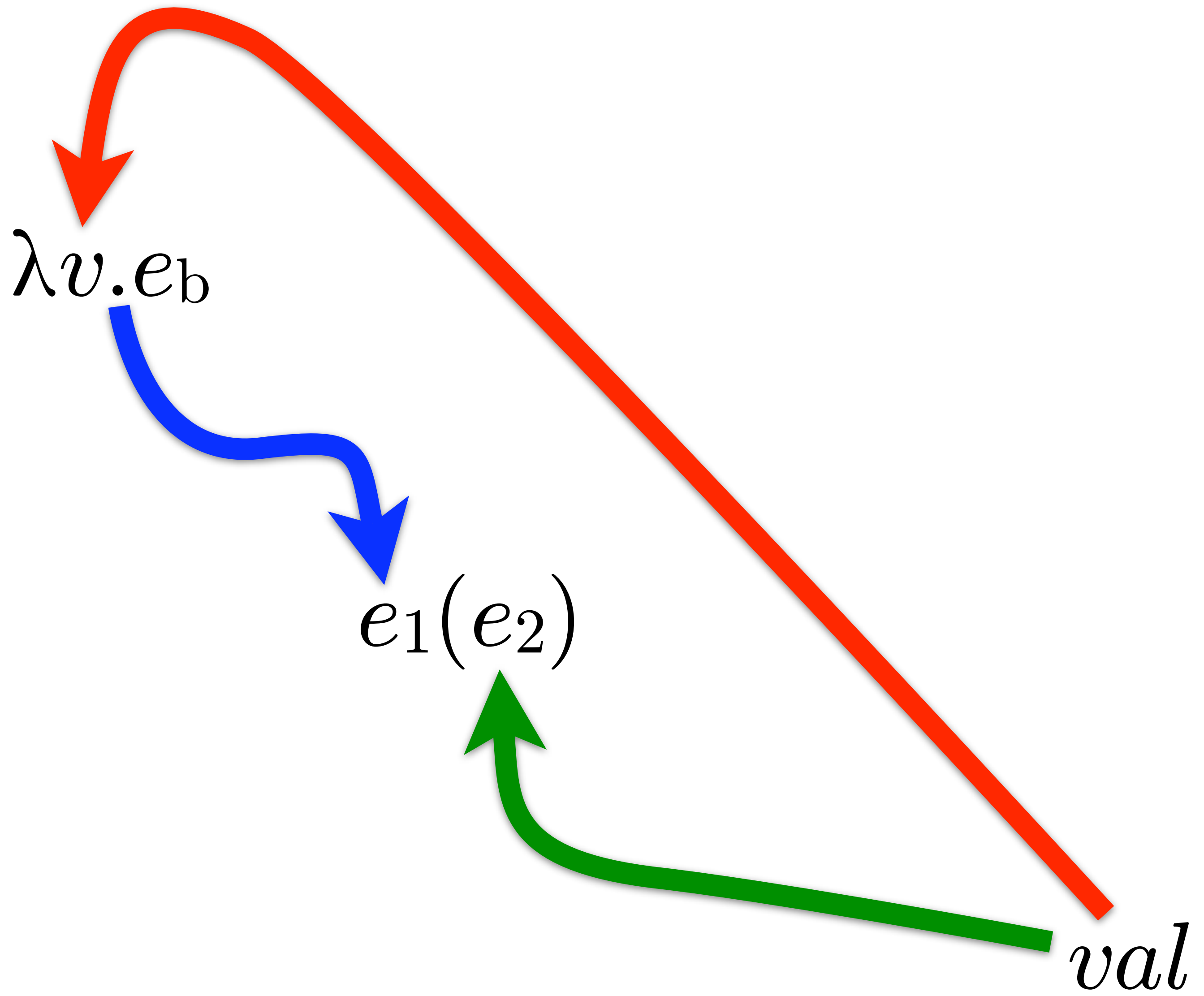
$\lambda v.e_{\mathrm{b}}$

$e_1(e_2)$

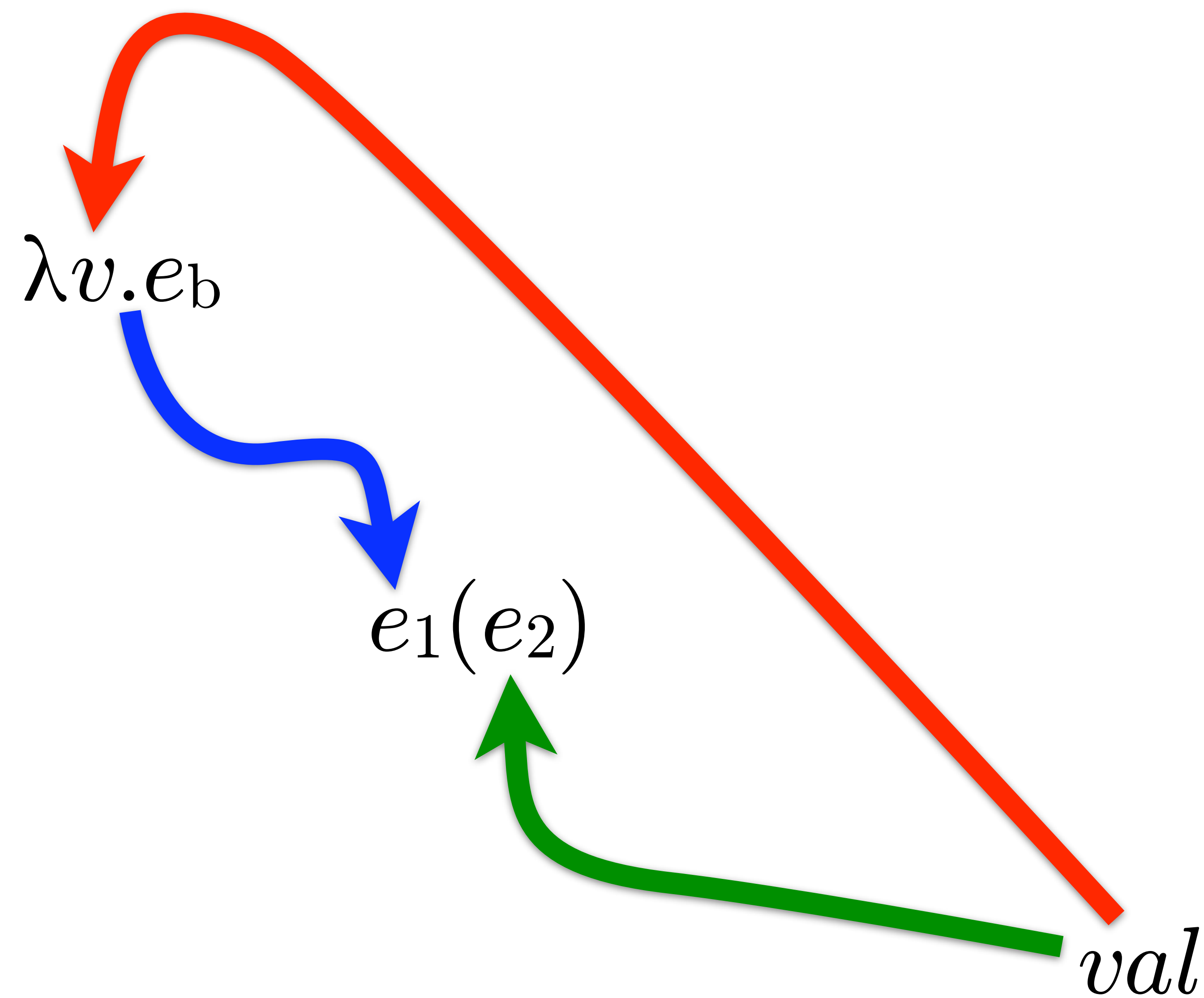
0CFA



0CFA



0CFA



$$\frac{\lambda v.e_{\mathrm{b}} \in \mathrm{FlowsTo}[e_1] \quad \text{and} \quad val \in \mathrm{FlowsTo}[e_2]}{val \in \mathrm{FlowsTo}[v]}$$

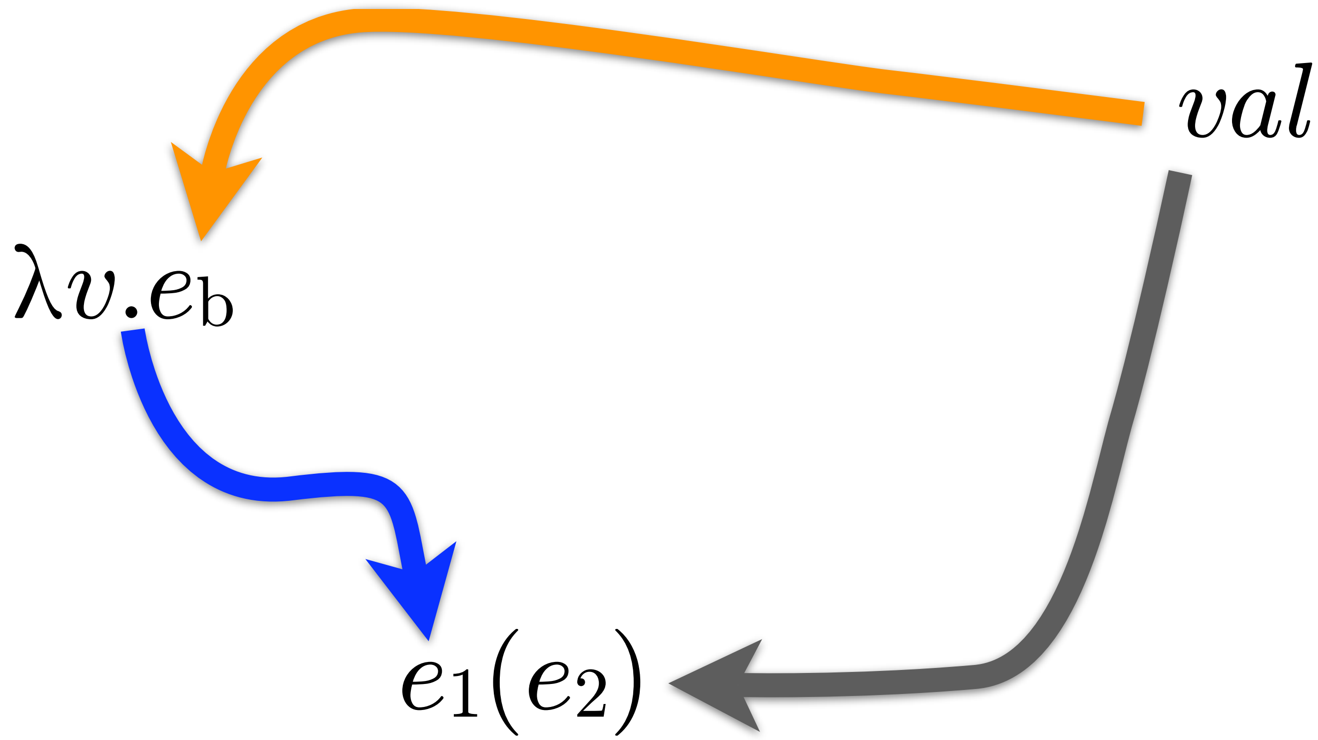
0CFA

val

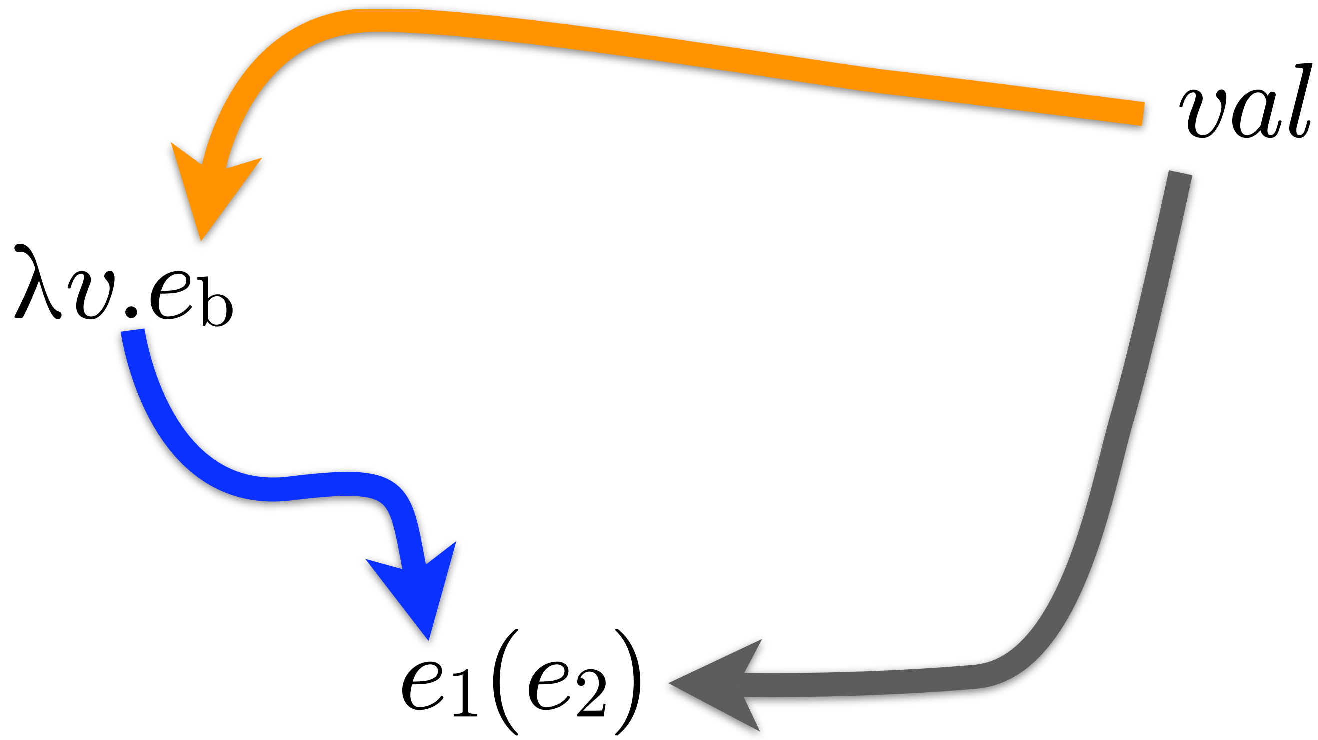
$\lambda v.e_{\mathrm{b}}$

$e_1(e_2)$

0CFA



0CFA



$$\frac{\lambda v.e_{\mathrm{b}} \in \mathrm{FlowsTo}[e_1] \quad \text{and} \quad val \in \mathrm{FlowsTo}[e_{\mathrm{b}}]}{val \in \mathrm{FlowsTo}[e_1(e_2)]}$$

0CFA

$$\frac{\lambda v.e_{\mathrm{b}} \in \mathrm{FlowsTo}[e_1] \quad \text{and} \quad val \in \mathrm{FlowsTo}[e_{\mathrm{b}}]}{val \in \mathrm{FlowsTo}[e_1(e_2)]}$$

0CFA

$$\lambda v.e_{\mathrm{b}} \in \mathrm{FlowsTo}[\lambda v.e_{\mathrm{b}}]$$

$$\frac{\lambda v.e_{\mathrm{b}} \in \mathrm{FlowsTo}[e_1] \quad \text{and} \quad val \in \mathrm{FlowsTo}[e_{\mathrm{b}}]}{val \in \mathrm{FlowsTo}[e_1(e_2)]}$$

$$\frac{\lambda v.e_{\mathrm{b}} \in \mathrm{FlowsTo}[e_1] \quad \text{and} \quad val \in \mathrm{FlowsTo}[e_2]}{val \in \mathrm{FlowsTo}[v]}$$

0CFA (Palsberg, 1995)

$$\{\lambda v.e_{\mathrm{b}}\} \subseteq \mathrm{FlowsTo}[\lambda v.e_{\mathrm{b}}]$$

$$\frac{\lambda v.e_{\mathrm{b}} \in \mathrm{FlowsTo}[e_1]}{\mathrm{FlowsTo}[e_{\mathrm{b}}] \subseteq \mathrm{FlowsTo}[e_1(e_2)]}$$

$$\frac{\lambda v.e_{\mathrm{b}} \in \mathrm{FlowsTo}[e_1]}{\mathrm{FlowsTo}[e_2] \subseteq \mathrm{FlowsTo}[v]}$$

Computing 0CFA

- $\mathrm{FlowsTo} \subseteq$ Exp × Lam
- Start at the bottom: $\emptyset$
- Iterate to a fixed point

Example: 0CFA

```
let id = λx.x
    v1 = id(3)
    v2 = id(4)
 in v2
```

Example: 0CFA

```
let id = λx.x
    v1 = id(3)
    v2 = id(4)
 in v2
```

Example: 0CFA

```
let id = λx.x
    v1 = id(3)
    v2 = id(4)
 in v2
```

$\text{FlowsTo}[\texttt{id}] = \{\lambda\texttt{x.x}\}$

Example: 0CFA

```
let id = λx.x
    v1 = id(3)
    v2 = id(4)
 in v2
```

$\text{FlowsTo}[\mathtt{id}] = \{\mathtt{λx.x}\}$

$\text{FlowsTo}[\mathtt{x}] = \{\mathtt{3}\}$

Example: 0CFA

```
let id = λx.x
    v1 = id(3)
    v2 = id(4)
 in v2
```

$\mathrm{FlowsTo}[\mathtt{id}] = \{\mathtt{λx.x}\}$

$\mathrm{FlowsTo}[\mathtt{x}] = \{\mathtt{3}\}$

$\mathrm{FlowsTo}[\mathtt{id(3)}] = \{\mathtt{3}\}$

Example: 0CFA

```
let id = λx.x
    v1 = id(3)
    v2 = id(4)
 in v2
```

$\mathrm{FlowsTo}[\mathtt{id}] = \{\mathtt{λx.x}\}$

$\mathrm{FlowsTo}[\mathtt{x}] = \{\mathtt{3}\}$

$\mathrm{FlowsTo}[\mathtt{id(3)}] = \{\mathtt{3}\}$

$\mathrm{FlowsTo}[\mathtt{v1}] = \{\mathtt{3}\}$

Example: 0CFA

```
let id = λx.x
    v1 = id(3)
    v2 = id(4)
 in v2
```

$\mathrm{FlowsTo}[\mathtt{id}] = \{\mathtt{λx.x}\}$

$\mathrm{FlowsTo}[\mathtt{x}] = \{\mathtt{3,4}\}$

$\mathrm{FlowsTo}[\mathtt{id(3)}] = \{\mathtt{3}\}$

$\mathrm{FlowsTo}[\mathtt{v1}] = \{\mathtt{3}\}$

Example: 0CFA

```
let id = λx.x
    v1 = id(3)
    v2 = id(4)
 in v2
```

$$\mathrm{FlowsTo}[\mathtt{id}] = \{\mathtt{\lambda x.x}\}$$

$$\mathrm{FlowsTo}[\mathtt{x}] = \{\mathtt{3,4}\}$$

$$\mathrm{FlowsTo}[\mathtt{id(3)}] = \{\mathtt{3}\}$$

$$\mathrm{FlowsTo}[\mathtt{v1}] = \{\mathtt{3}\}$$

$$\mathrm{FlowsTo}[\mathtt{id(4)}] = \{\mathtt{3,4}\}$$

Example: 0CFA

```
let id = λx.x
    v1 = id(3)
    v2 = id(4)
 in v2
```

$$\mathrm{FlowsTo}[\texttt{id}] = \{\texttt{λx.x}\}$$

$$\mathrm{FlowsTo}[\texttt{x}] = \{\texttt{3,4}\}$$

$$\mathrm{FlowsTo}[\texttt{id(3)}] = \{\texttt{3}\}$$

$$\mathrm{FlowsTo}[\texttt{v1}] = \{\texttt{3}\}$$

$$\mathrm{FlowsTo}[\texttt{id(4)}] = \{\texttt{3,4}\}$$

$$\mathrm{FlowsTo}[\texttt{v2}] = \{\texttt{3,4}\}$$

Example: 0CFA

```
let id = λx.x
    v1 = id(3)
    v2 = id(4)
 in v2
```

$$\mathrm{FlowsTo}[\mathtt{id}] = \{\mathtt{\lambda x.x}\}$$

$$\mathrm{FlowsTo}[\mathtt{x}] = \{\mathtt{3,4}\}$$

$$\mathrm{FlowsTo}[\mathtt{id(3)}] = \{\mathtt{3,4}\}$$

$$\mathrm{FlowsTo}[\mathtt{v1}] = \{\mathtt{3}\}$$

$$\mathrm{FlowsTo}[\mathtt{id(4)}] = \{\mathtt{3,4}\}$$

$$\mathrm{FlowsTo}[\mathtt{v2}] = \{\mathtt{3,4}\}$$

Example: 0CFA

```
let id = λx.x
    v1 = id(3)
    v2 = id(4)
 in v2
```

$$\mathrm{FlowsTo}[\mathtt{id}] = \{\mathtt{λx.x}\}$$

$$\mathrm{FlowsTo}[\mathtt{x}] = \{\mathtt{3,4}\}$$

$$\mathrm{FlowsTo}[\mathtt{id(3)}] = \{\mathtt{3,4}\}$$

$$\mathrm{FlowsTo}[\mathtt{v1}] = \{\mathtt{3,4}\}$$

$$\mathrm{FlowsTo}[\mathtt{id(4)}] = \{\mathtt{3,4}\}$$

$$\mathrm{FlowsTo}[\mathtt{v2}] = \{\mathtt{3,4}\}$$

Equality-based CFA

0CFA (Palsberg, 1995)

$$\{\lambda v.e_{\mathrm{b}}\} \subseteq \mathrm{FlowsTo}[\lambda v.e_{\mathrm{b}}]$$

$$\frac{\lambda v.e_{\mathrm{b}} \in \mathrm{FlowsTo}[e_1]}{\mathrm{FlowsTo}[e_{\mathrm{b}}] \subseteq \mathrm{FlowsTo}[e_1(e_2)]}$$

$$\frac{\lambda v.e_{\mathrm{b}} \in \mathrm{FlowsTo}[e_1]}{\mathrm{FlowsTo}[e_2] \subseteq \mathrm{FlowsTo}[v]}$$

Equality CFA (Heinglen, 1992)

$$\{\lambda v.e_{\mathrm{b}}\} \subseteq \mathrm{FlowsTo}[\lambda v.e_{\mathrm{b}}]$$

$$\frac{\lambda v.e_{\mathrm{b}} \in \mathrm{FlowsTo}[e_1]}{\mathrm{FlowsTo}[e_{\mathrm{b}}] = \mathrm{FlowsTo}[e_1(e_2)]}$$

$$\frac{\lambda v.e_{\mathrm{b}} \in \mathrm{FlowsTo}[e_1]}{\mathrm{FlowsTo}[e_2] = \mathrm{FlowsTo}[v]}$$

Computing (=)CFA

- Walk over parse tree
- Join flows with union-find
- Almost linear-time!

Example: (=)CFA

```
let id = λx.x
    v1 = (id 3)
    v2 = (id 4)
 in v2
```

Example: (=)CFA

```
let id = λx.x
    v1 = (id 3)
    v2 = (id 4)
 in v2
```

Type-based 0CFA

Typed lambda calculus

$$
\begin{aligned}
t &::= e : \tau \\
e &::= v \\
&\quad \mid \ \lambda^{\ell} v.t \\
&\quad \mid \ t\, t'
\end{aligned}
$$

Flow types

$$\tau \in \mathcal{P}(\mathsf{Lab})$$

Type rules

$$\frac{\ell \in \tau}{\Gamma \vdash (\lambda^{\ell} v.t) : \tau}$$

$$\frac{\Gamma \vdash t : \tau \quad \Gamma \vdash t' : \tau' \quad (\lambda^{\ell} v_\ell . t_\ell) \in \tau \implies \Gamma[v_\ell \mapsto \tau'] \vdash t_\ell : \tau''}{\Gamma \vdash (t\, t') : \tau''}$$

$$\frac{\Gamma \vdash e : \tau \quad \tau \subseteq \tau'}{\Gamma \vdash e : \tau'}$$

Example: Omega

$$((\lambda^A f.(f\ f))$$
$$(\lambda^B f.(f\ f))$$

Example: Omega

$$((\lambda^A f.(f : \{?\}\ f : \{?\}) : \{?\}) : \{?\}$$
$$(\lambda^B f.(f : \{?\}\ f : \{?\}) : \{?\}) : \{?\}) : \{?\}$$

Example: Omega

$$((\lambda^A f.(f : \{B\}\ f : \{B\}) : \{\}) : \{A\}$$
$$(\lambda^B f.(f : \{B\}\ f : \{B\}) : \{\}) : \{B\}) : \{\}$$

Flow-typed functions

$$\tau \in \mathcal{P}\,(\mathsf{Lab})$$

Flow-typed functions

$$\tau ::= \tau_1 \xrightarrow{L} \tau_2 \mid \cdots$$

Typed type-based rules

$$\frac{\ell \in L \quad \Gamma[v \mapsto \tau] \vdash t : \tau'}{\Gamma \vdash (\lambda^\ell v.t) : \tau \xrightarrow{L} \tau'}$$

$$\frac{\Gamma \vdash t : \tau' \xrightarrow{L} \tau'' \quad \Gamma \vdash t' : \tau' \quad (\lambda^\ell v_\ell.t_\ell) \in L \implies \Gamma[v_\ell \mapsto \tau'] \vdash t_\ell : \tau''}{\Gamma \vdash (t\ t') : \tau''}$$

$$\frac{\Gamma \vdash e : \tau \xrightarrow{L} \tau' \quad L \subseteq L'}{\Gamma \vdash e : \tau \xrightarrow{L'} \tau'}$$

Typed type-based rules

$$\frac{\ell \in L \qquad \Gamma[v \mapsto \tau] \vdash t : \tau'}{\Gamma \vdash (\lambda^{\ell} v.t) \boxed{: \tau \xrightarrow{L} \tau'}}$$

$$\frac{\Gamma \vdash t \boxed{\tau' \xrightarrow{L} \tau''} \qquad \Gamma \vdash t' : \tau' \qquad (\lambda^{\ell} v_{\ell}.t_{\ell}) \in L \implies \Gamma[v_{\ell} \mapsto \tau'] \vdash t_{\ell} : \tau''}{\Gamma \vdash (t\ t') : \tau''}$$

$$\frac{\Gamma \vdash e \boxed{: \tau \xrightarrow{L} \tau'} \qquad L \subseteq L'}{\Gamma \vdash e \boxed{: \tau \xrightarrow{L'} \tau'}}$$

Small-step CFA paradigm

Small-step strategy

- Build small-step concrete machine
- Abstract into finite-state machine
- Analysis is just graph-reachability

Advantages

- Makes implementation easy
- Serves as universal framework
- Makes total correctness easy

Small-step machine

Small-step machine

- Convert program e into machine state s_0

$$
\begin{array}{c}
e \\
\Big\downarrow \\
s_0
\end{array}
$$

Small-step machine

- Convert program e into machine state s_0
- Transition from state s_n to state s_{n+1}

$$\begin{tikzcd} e \arrow[d] \\ s_0 \arrow[r] & s_1 \arrow[r] & s_2 \arrow[r] & s_3 \arrow[r] & s_4 \arrow[r] & \ldots \end{tikzcd}$$

Abstract machine

$$\begin{tikzcd} e \arrow[d] \\ s_0 \arrow[r] & s_1 \arrow[r] & s_2 \arrow[r] & s_3 \arrow[r] & s_4 \arrow[r] & \ldots \end{tikzcd}$$

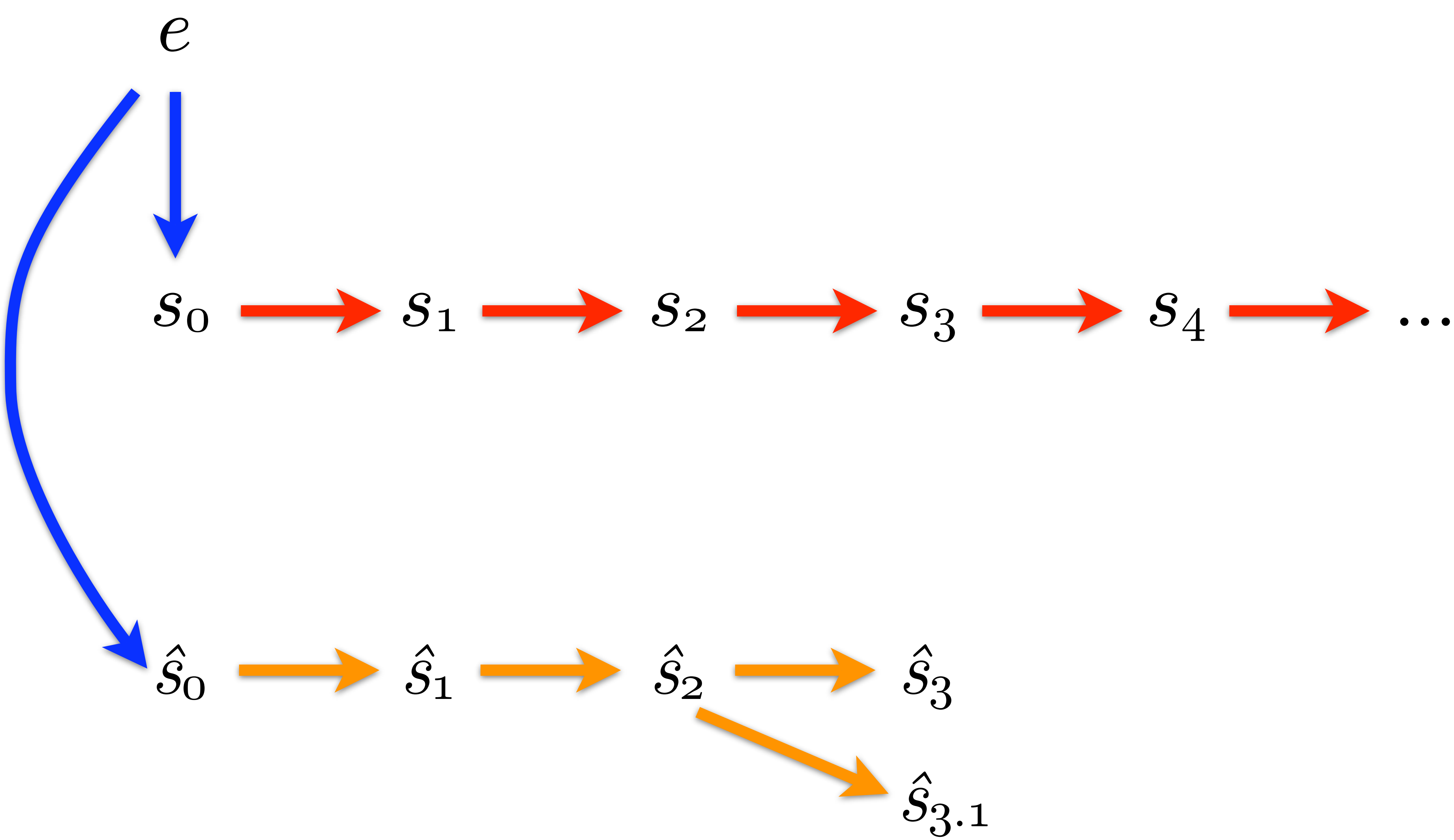
Abstract machine

$$
\begin{tikzcd}
e \arrow[d] \\
s_0 \arrow[r] & s_1 \arrow[r] & s_2 \arrow[r] & s_3 \arrow[r] & s_4 \arrow[r] & \ldots
\end{tikzcd}
$$

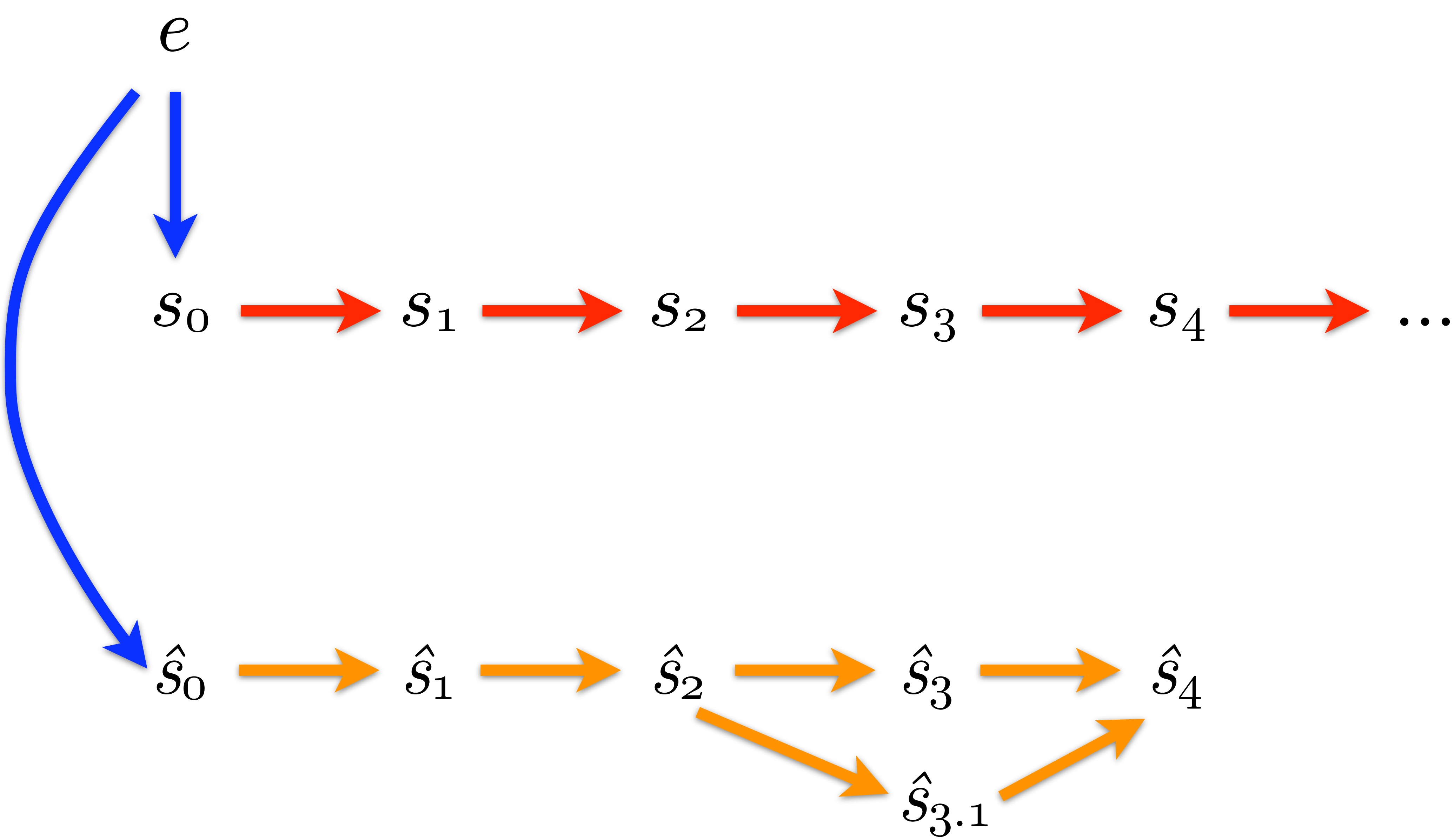
Abstract machine

$$\begin{tikzcd} e \arrow[d] \arrow[dd, bend right] \\ s_0 \arrow[r] & s_1 \arrow[r] & s_2 \arrow[r] & s_3 \arrow[r] & s_4 \arrow[r] & \ldots \\ \hat{s}_0 \end{tikzcd}$$

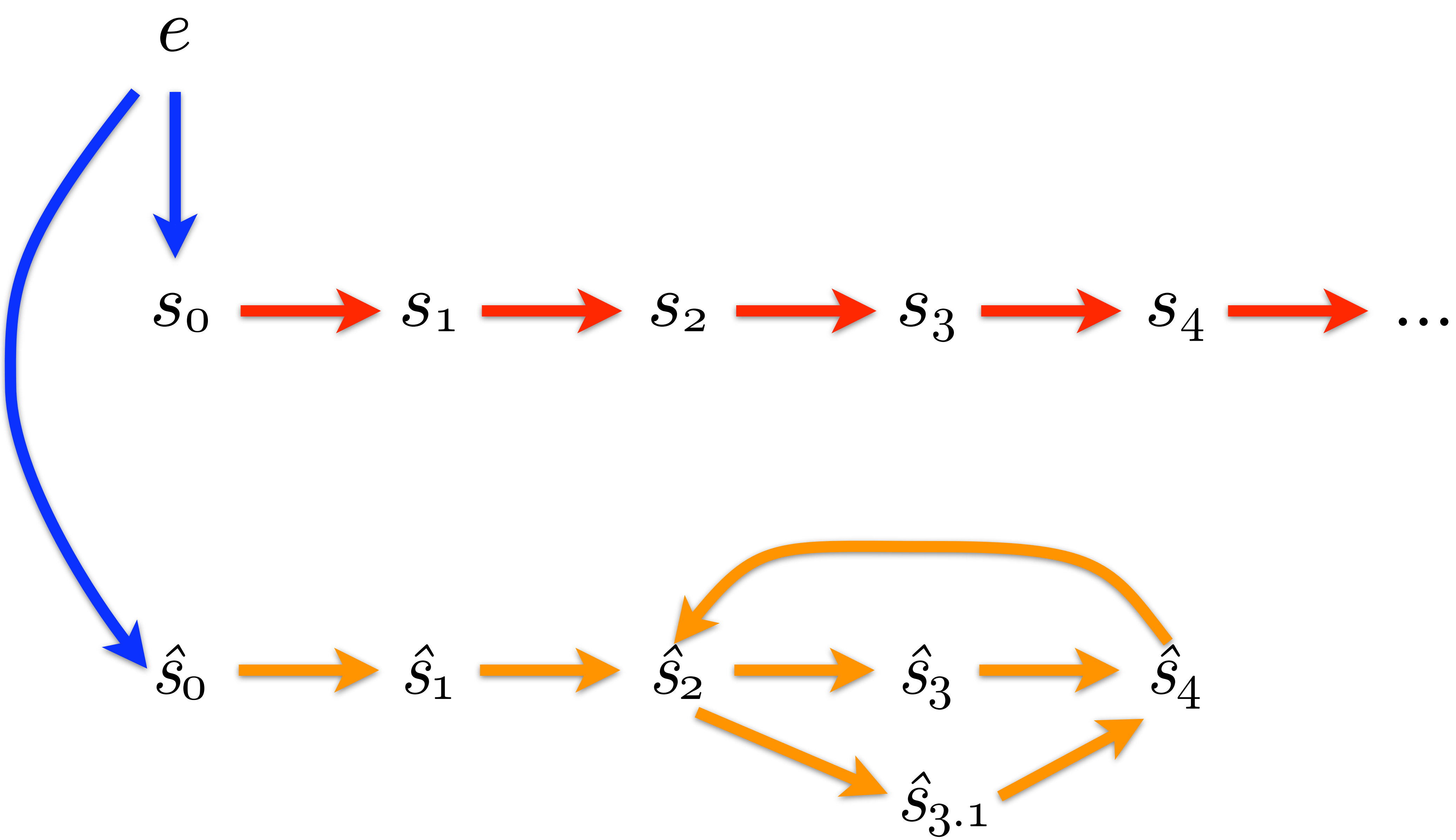
Abstract machine



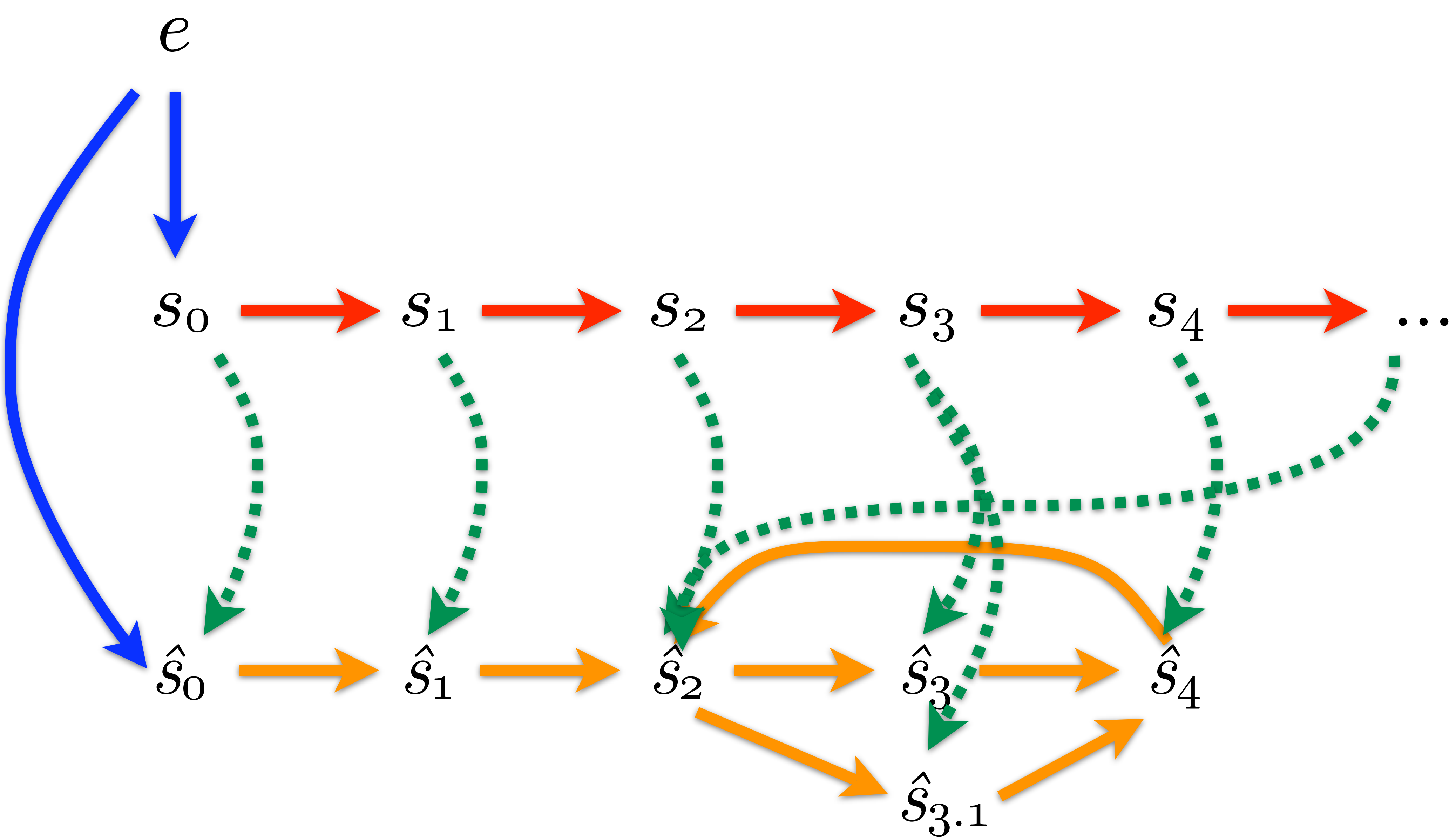
Abstract machine



Abstract machine



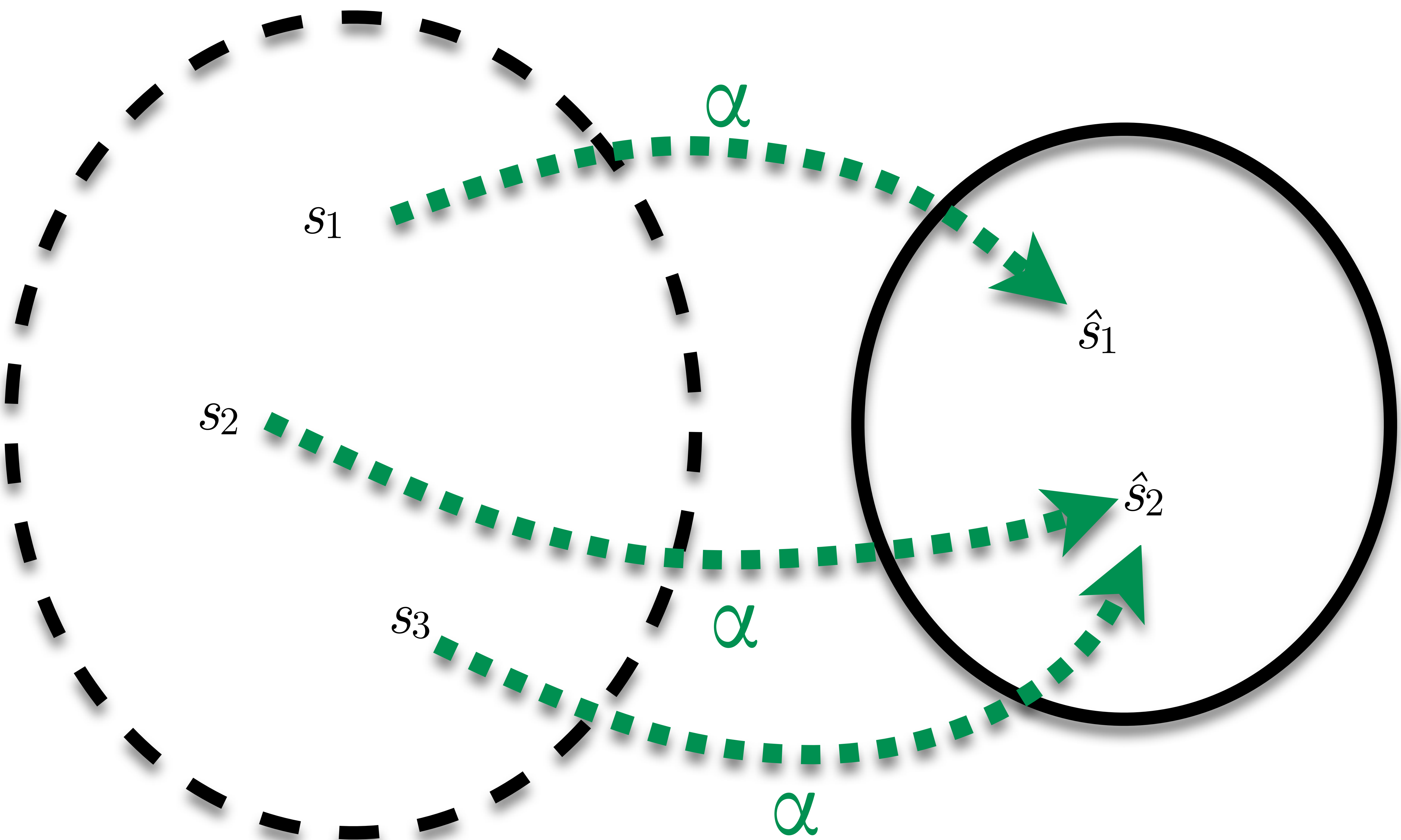
Abstract machine



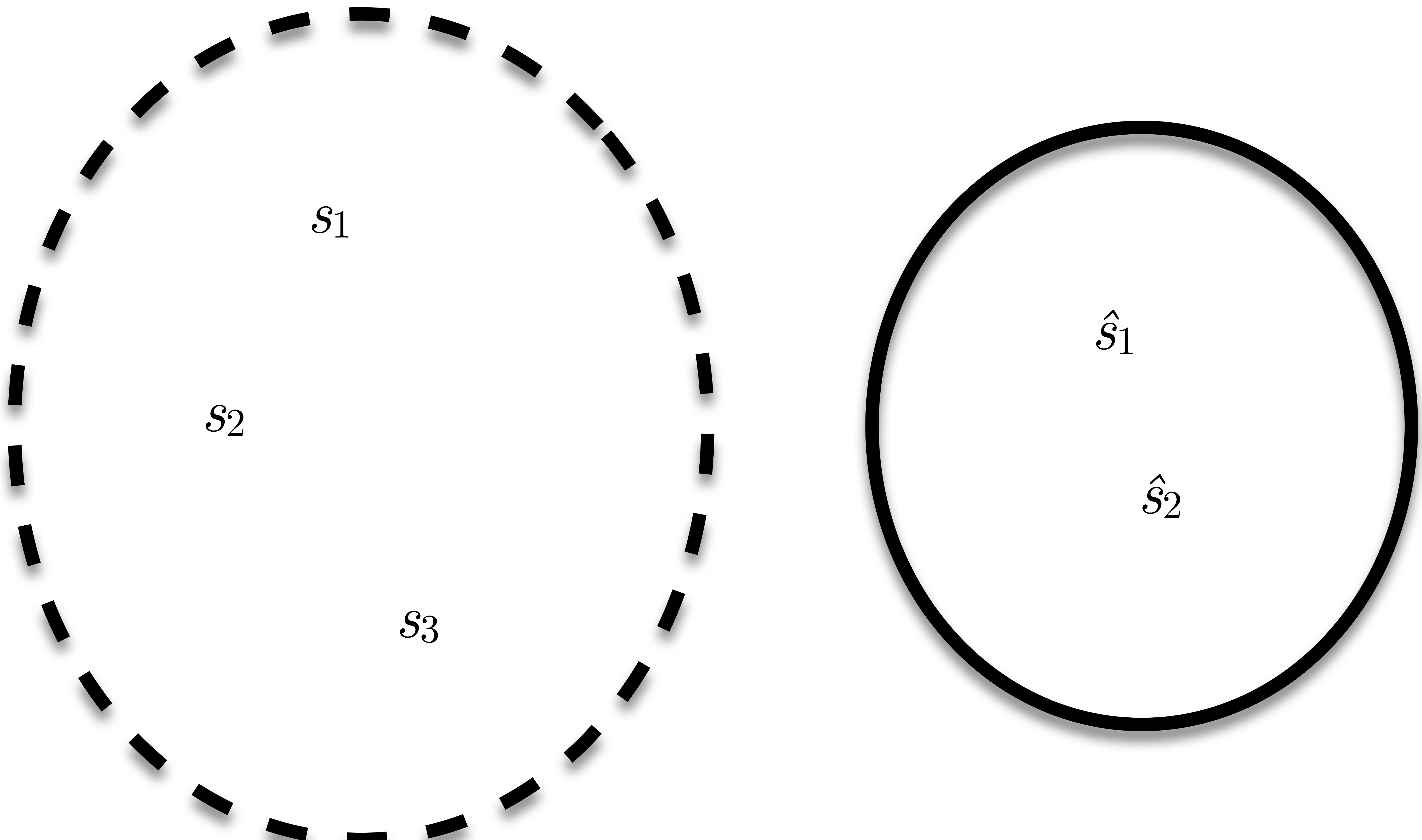
Theorem: The abstract simulates the concrete.

Abstraction

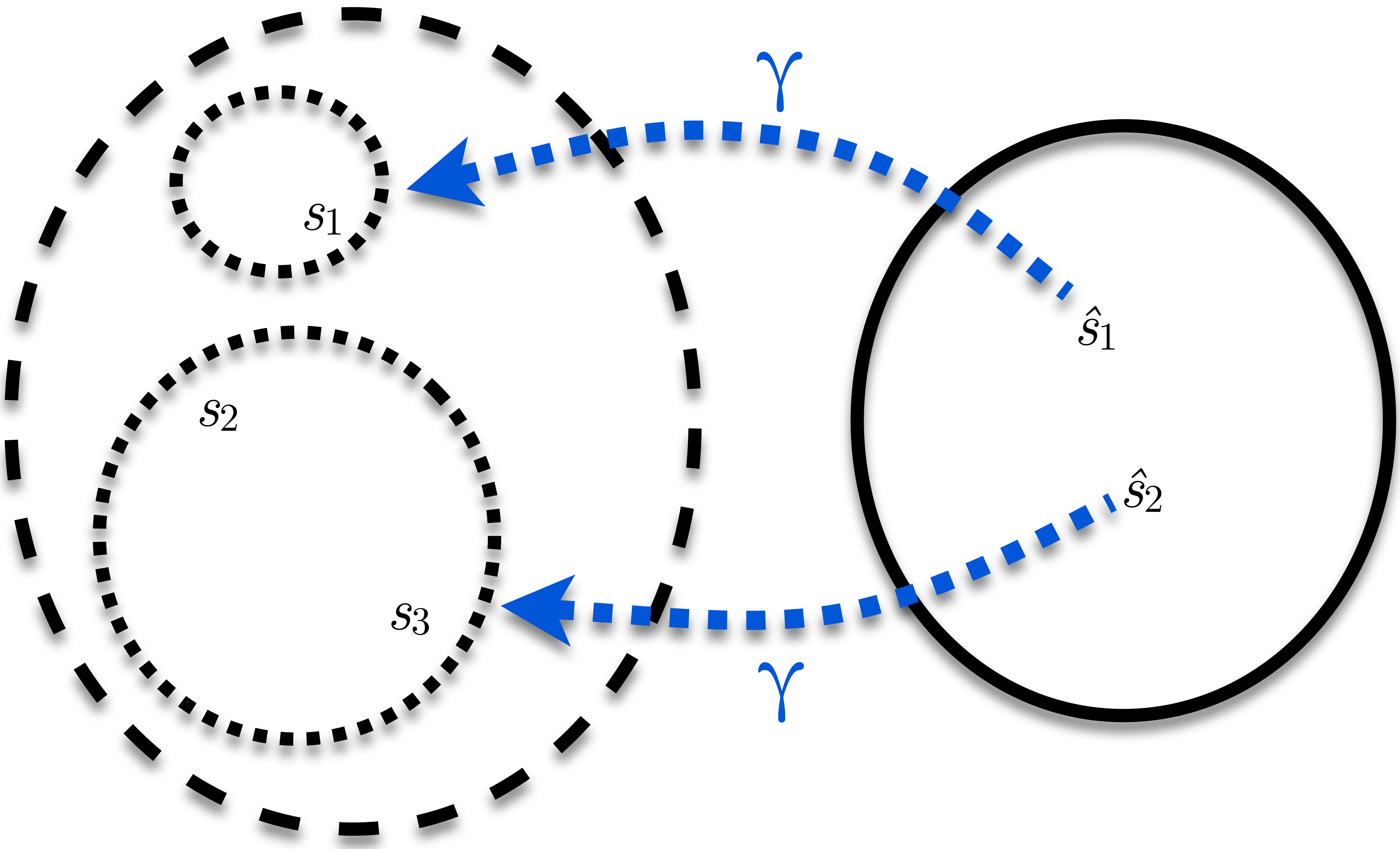
Abstraction



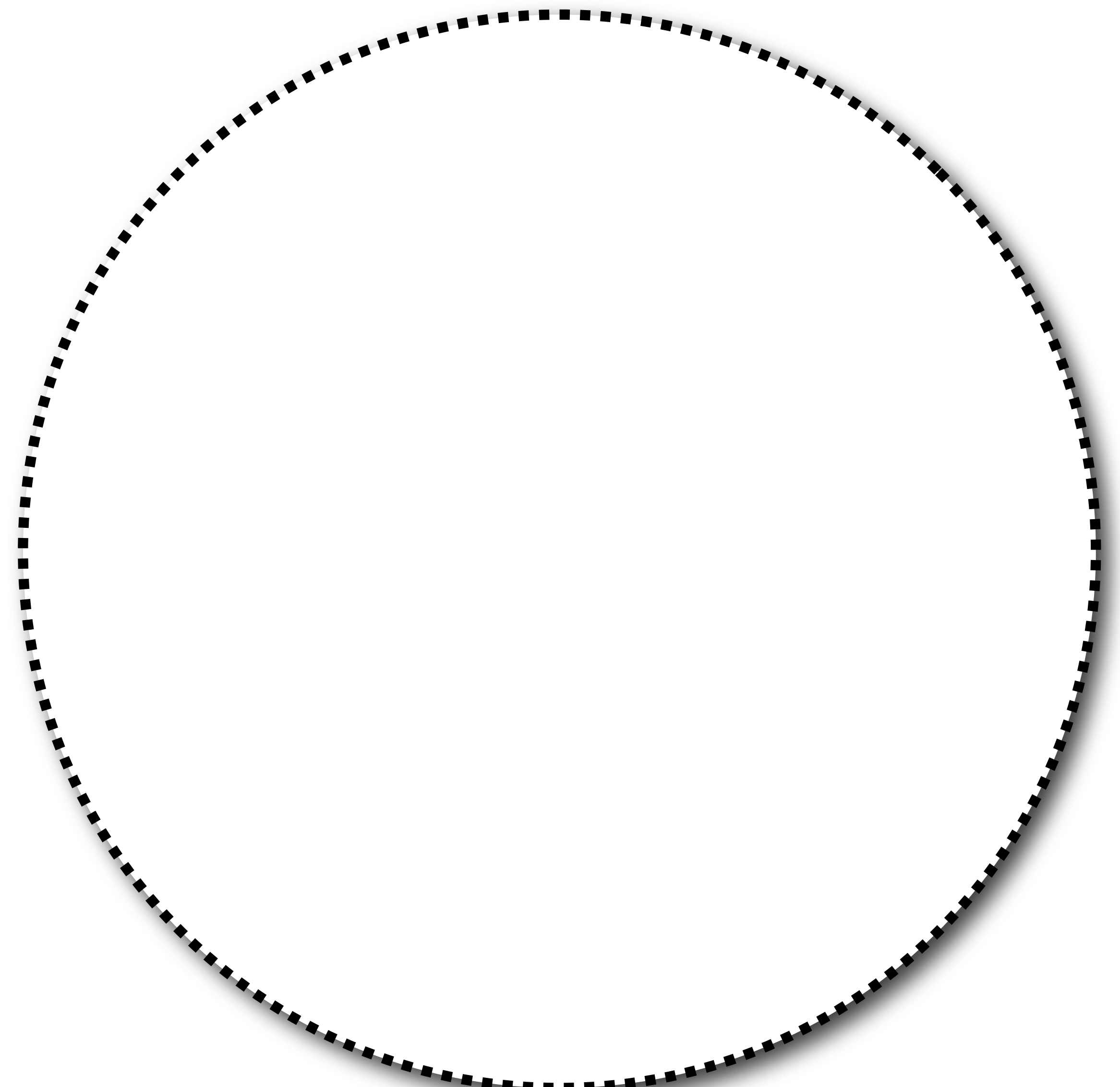
Concretization



Concretization

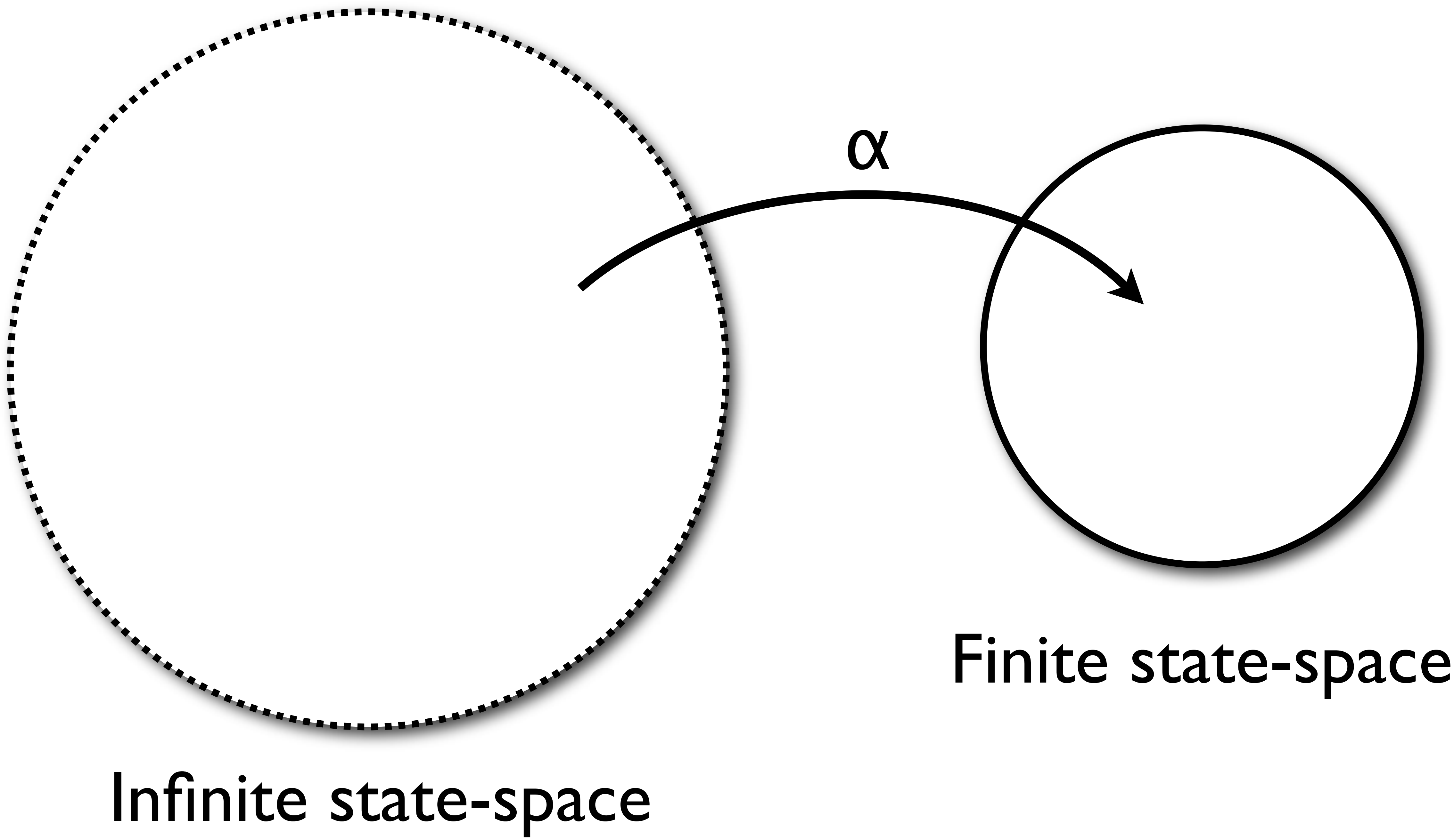


Small-step CFAs...



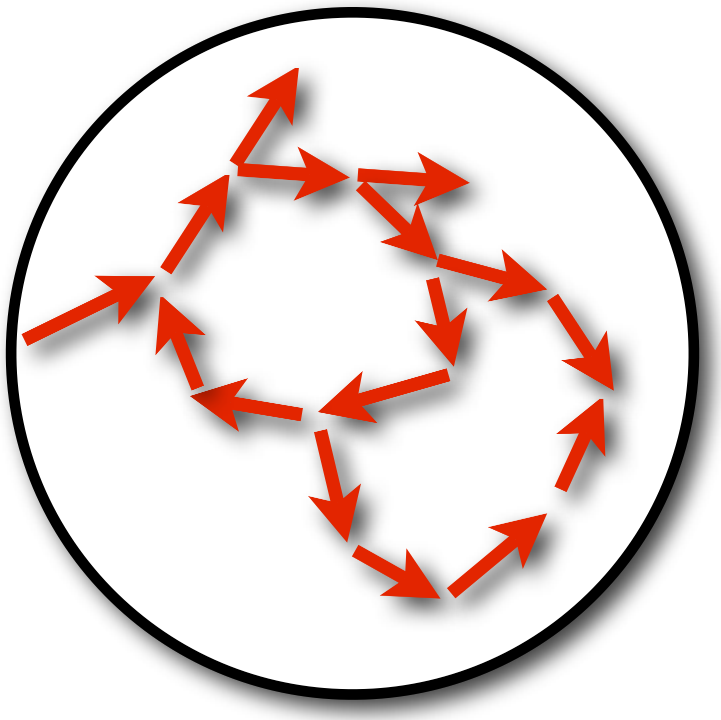
Infinite state-space

Small-step CFAs...



...are finite state machines.

...are finite state machines.

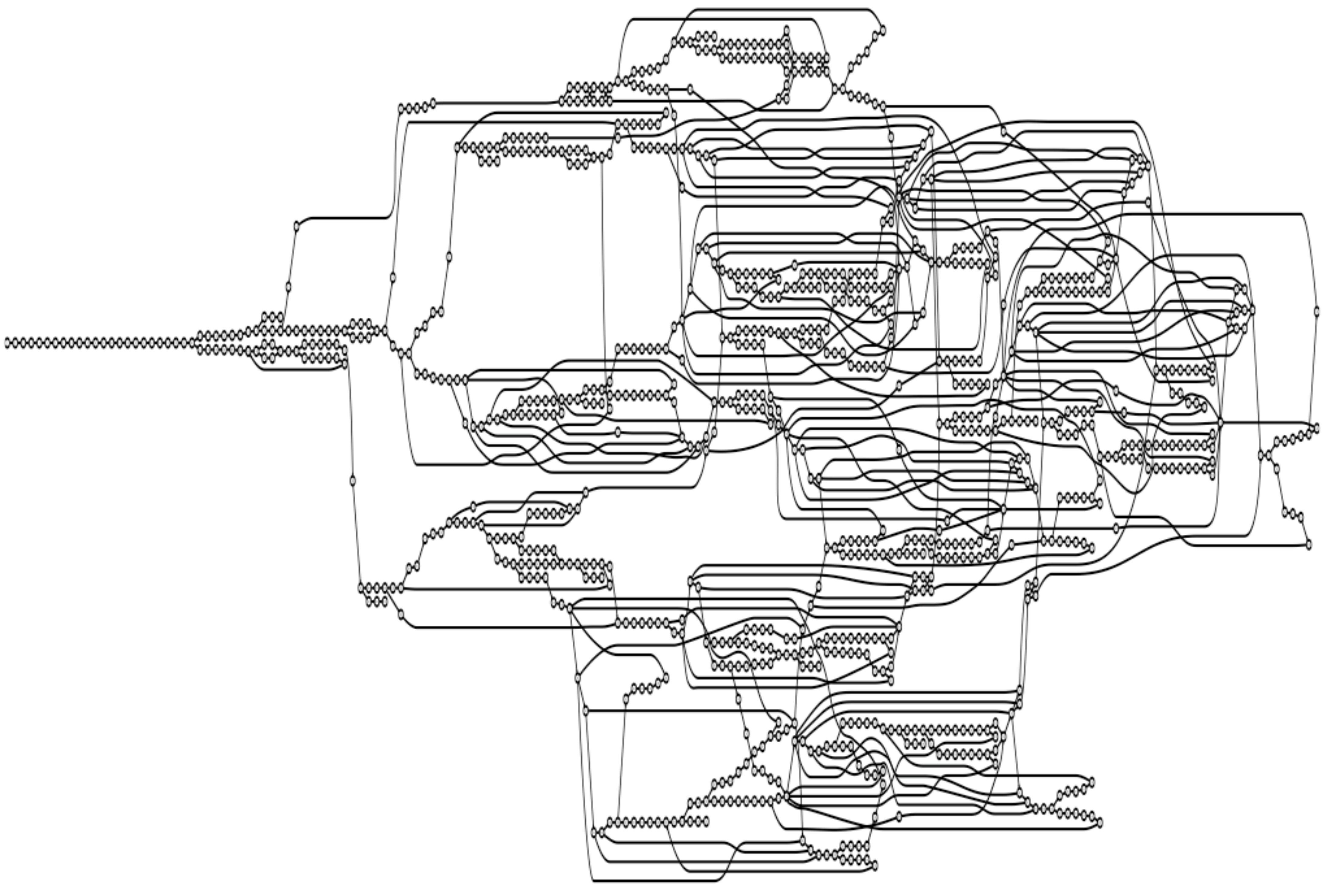


Finite state-space

Example: Abstract graph

```
(letrec ((lp1 (λ (i x)
                (if (= 0 i)  x
                    (letrec ((lp2 (λ (j f y) (if (= 0 j)
                                                 (lp1 (- i 1) y)
                                                 (lp2 (- j 1) f
                                                      (f y))))))
                      (lp2 10 (λ (n) (+ n i)) x))))))
  (lp1 10 0))
```

Example: Abstract graph



Small-step 0CFA for CPS

Why use CPS?

- Evaluation of CPS is atomic
- Control is reified as data
- Control-flow = data-flow

Continuation-passing style

$$
\begin{aligned}
v &\in \mathsf{Var} \\
f, e &\in \mathsf{Exp} = \mathsf{Var} + \mathsf{Lam} \\
lam &\in \mathsf{Lam} ::= (\lambda\ (v_1 \ldots v_n)\ \ call) \\
call &\in \mathsf{Call} ::= (f\ e_1 \ldots e_n)
\end{aligned}
$$

Concrete state-space 1

$$\varsigma \in \Sigma = \mathsf{Call} \times Env$$

$$\rho \in Env = \mathsf{Var} \rightharpoonup Clo$$

$$clo \in Clo = \mathsf{Lam} \times Env$$

Concrete injector

$$\mathcal{I} : \textsf{Call} \to \Sigma$$

$$\mathcal{I}(call) = (call, [])$$

Concrete semantics I

$$(\Rightarrow) \subseteq \Sigma \times \Sigma$$

$$\mathcal{E} : \mathsf{Exp} \times Env \rightharpoonup Clo$$

$$\mathcal{E} : \mathsf{Exp} \times Env \rightharpoonup Clo$$

$$\mathcal{E}(lam, \rho) = (lam, \rho)$$

$$\mathcal{E}(v, \rho) = \rho(v)$$

$$(\Rightarrow) \subseteq \Sigma \times \Sigma$$

$$\begin{aligned}
([\![(f\ e_1 \ldots e_n)]\!], \rho) &\Rightarrow (call, \rho''), \text{ where} \\
([\![(\lambda\ (v_1 \ldots v_n)\ call)]\!], \rho') &= \mathcal{E}(f, \rho) \\
clo_i &= \mathcal{E}(e_i, \rho) \\
\rho'' &= \rho'[v_i \mapsto clo_i]
\end{aligned}$$

Abstracting to 0CFA

- Abstract closures into lambdas
- Merge all environments together

Abstract state-space I

$$\hat{\varsigma} \in \hat{\Sigma} = \mathsf{Call} \times \widehat{Env}$$

$$\hat{\rho} \in \widehat{Env} = \mathsf{Var} \rightharpoonup \mathcal{P}\left(\widehat{Clo}\right)$$

$$\widehat{clo} \in \widehat{Clo} = \mathsf{Lam}$$

Abstraction I

$$\alpha(lam, \rho) = lam$$

Abstraction I

$$\alpha(call, \rho) = (call, \alpha(\rho))$$

Abstraction I

$$\alpha(\rho) = \lambda v.\left\{\alpha(\rho'(v)) : \rho' \text{ is reachable in } \rho\right\}$$

Abstract injector

$$\hat{\mathcal{I}} : \text{Call} \rightarrow \hat{\Sigma}$$

$$\hat{\mathcal{I}}(call) = (call, \bot)$$

Abstract semantics I

$$(\rightsquigarrow) \subseteq \hat{\Sigma} \times \hat{\Sigma}$$

$$\hat{\mathcal{E}} : \mathsf{Exp} \times \widehat{Env} \to \mathcal{P}\left(\widehat{Clo}\right)$$

$$\hat{\mathcal{E}} : \mathsf{Exp} \times \widehat{Env} \to \mathcal{P}\left(\widehat{Clo}\right)$$

$$\hat{\mathcal{E}}(lam, \hat{\rho}) = \{lam\}$$

$$\hat{\mathcal{E}}(v, \hat{\rho}) = \hat{\rho}(v)$$

$$(\rightsquigarrow) \subseteq \hat{\Sigma} \times \hat{\Sigma}$$

$$([\![(f\ e_1 \ldots e_n)]\!], \hat{\rho}) \rightsquigarrow (call, \hat{\rho}'), \text{ where}$$

$$[\![(\lambda\ (v_1 \ldots v_n)\ call)]\!] \in \hat{\mathcal{E}}(f, \hat{\rho})$$

$$\hat{C}_i = \hat{\mathcal{E}}(e_i, \hat{\rho})$$

$$\hat{\rho}' = \hat{\rho} \sqcup [v_i \mapsto \hat{C}_i]$$

$$(\rightsquigarrow) \subseteq \hat{\Sigma} \times \hat{\Sigma}$$

$([\![(f\ e_1 \ldots e_n)]\!], \hat{\rho}) \rightsquigarrow (call, \hat{\rho}')$, where

$$[\![(\lambda\ (v_1 \ldots v_n)\ \ call)]\!] \in \hat{\mathcal{E}}(f, \hat{\rho})$$

$$\hat{C}_i = \hat{\mathcal{E}}(e_i, \hat{\rho})$$

$$\hat{\rho}' = \hat{\rho} \sqcup [v_i \mapsto \hat{C}_i]$$

$$(\rightsquigarrow) \subseteq \hat{\Sigma} \times \hat{\Sigma}$$

$$\begin{aligned}
([\![(f\ e_1 \ldots e_n)]\!], \hat{\rho}) &\rightsquigarrow (call, \hat{\rho}'), \text{ where} \\
[\![(\lambda\ (v_1 \ldots v_n)\ \ call)]\!] &\in \hat{\mathcal{E}}(f, \hat{\rho}) \\
\hat{C}_i &= \hat{\mathcal{E}}(e_i, \hat{\rho}) \\
\hat{\rho}' &= \hat{\rho} \sqcup [v_i \mapsto \hat{C}_i]
\end{aligned}$$

$$\begin{aligned}
([\![(f\ e_1 \ldots e_n)]\!], \rho) &\Rightarrow (call, \rho''), \text{ where} \\
([\![(\lambda\ (v_1 \ldots v_n)\ \ call)]\!], \rho') &= \mathcal{E}(f, \rho) \\
clo_i &= \mathcal{E}(e_i, \rho) \\
\rho'' &= \rho'[v_i \mapsto clo_i]
\end{aligned}$$

Join operation

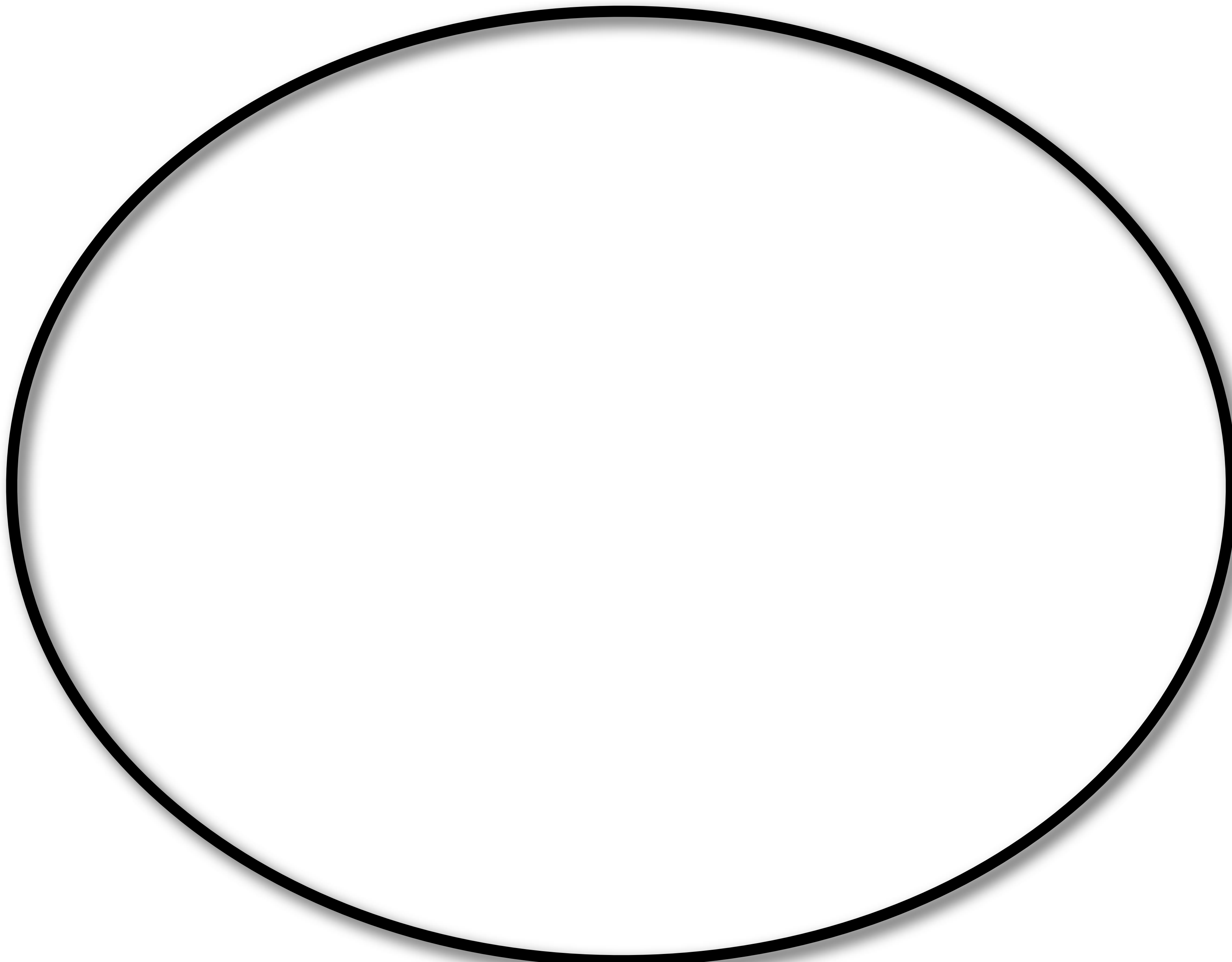
$$(\hat{\rho} \sqcup \hat{\rho}') = \lambda v.(\hat{\rho}(v) \cup \hat{\rho}'(v))$$

Soundness

$$\begin{tikzcd} \varsigma \arrow[r, "\Rightarrow"] \arrow[d, "\alpha"'] & \varsigma' \arrow[d, "\alpha"] \\ {} \arrow[d, "\sqsubseteq"'] & {} \arrow[d, "\sqsubseteq"] \\ \hat{\varsigma} \arrow[r, dotted, "\leadsto"'] & \hat{\varsigma}' \end{tikzcd}$$

Theorem: If the concrete takes a step,
then the abstract can take a matching step.

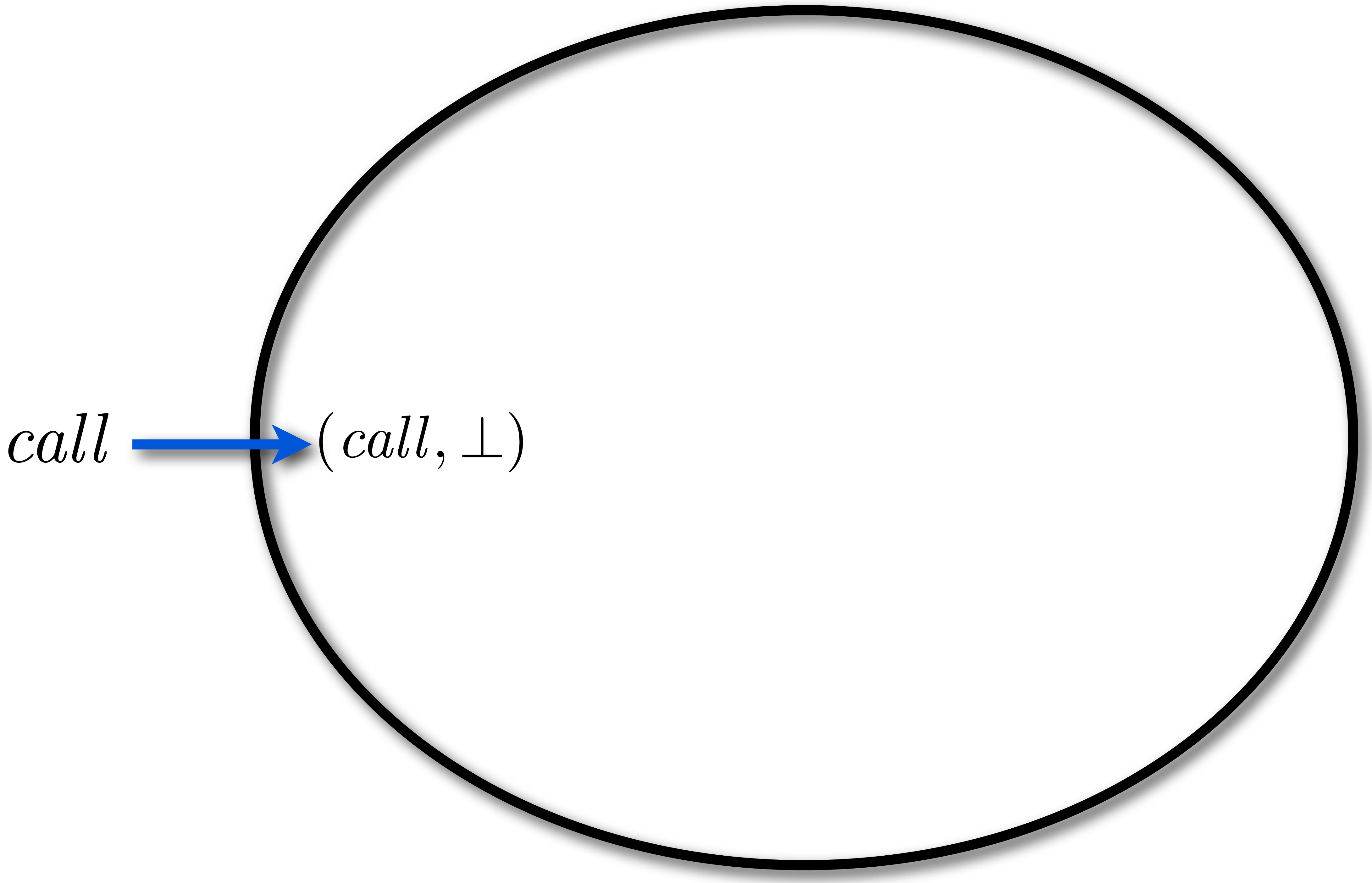
Running 0CFA



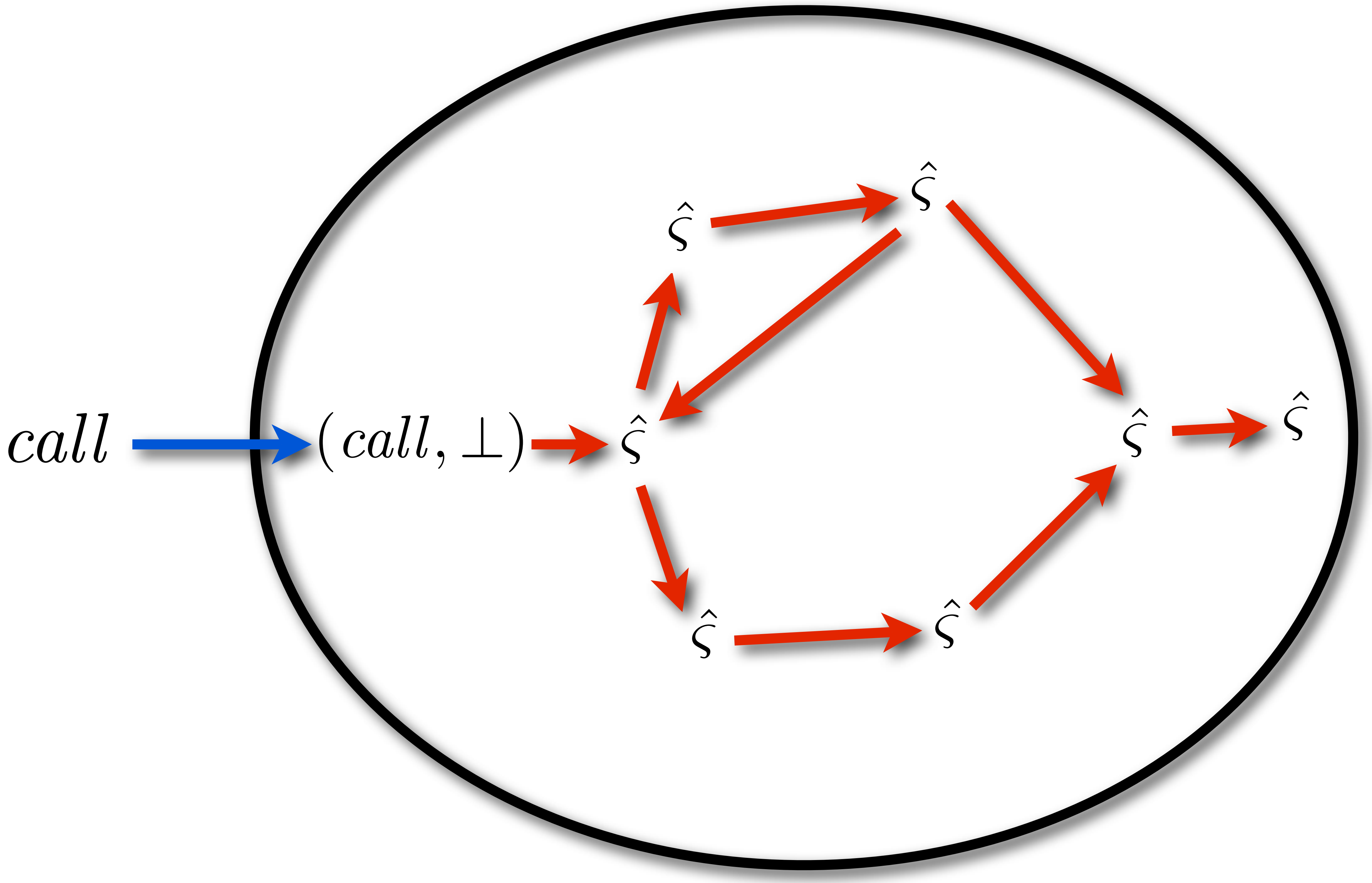
Running 0CFA

$$\hat{\Sigma}$$

Running 0CFA



Running 0CFA



Computing flow sets

$$\text{FlowsTo} = \bigsqcup_{\hat{\mathcal{I}}(call) \leadsto^* (\_,\hat{\rho})} \hat{\rho}$$

Example: Omega

$lam = (\lambda\ \texttt{(f) (f f))}$

Example: Omega

$$lam = (\lambda\ \texttt{(f)}\ \texttt{(f f)})$$

$$call = (lam\ \ lam)$$

Example: Omega

$$lam = (\lambda\ \texttt{(f) (f f)})$$

$$call = (lam\ \ lam)$$

$$\hat{\mathcal{I}}(call) = (call, \perp)$$

Example: Omega

$$lam = (\lambda\ \texttt{(f)}\ \texttt{(f f)})$$

$$call = (lam\ \ lam)$$

$$\hat{\mathcal{I}}(call) = (call, \bot) \longrightarrow (\llbracket\texttt{(f f)}\rrbracket, [\llbracket\texttt{f}\rrbracket \mapsto \{lam\}])$$

Example: Omega

$$lam = (\lambda\ \texttt{(f)}\ \texttt{(f f)})$$

$$call = (lam\ \ lam)$$

$$\hat{\mathcal{I}}(call) = (call, \bot) \longrightarrow (⟦\texttt{(f f)}⟧, [⟦\texttt{f}⟧ \mapsto \{lam\}])$$

$$\text{FlowsTo} = [\texttt{f} \mapsto \{lam\}]$$

CPS with a `let` form

$$
\begin{aligned}
v &\in \mathsf{Var} \\
f, e &\in \mathsf{Exp} = \mathsf{Var} + \mathsf{Lam} \\
lam &\in \mathsf{Lam} ::= (\lambda\ (v_1 \ldots v_n)\ call) \\
call &\in \mathsf{Call} ::= (f\ e_1 \ldots e_n) \\
&\qquad\quad \mid\ (\texttt{let}\ ((v\ e))\ call)
\end{aligned}
$$

Let-transition rule

$$(\llbracket \texttt{(let ((}v\ e\texttt{))}\ call\texttt{)} \rrbracket, \rho) \Rightarrow (call, \rho'), \text{ where}$$
$$\rho' = \rho[v \mapsto \mathcal{E}(e, \rho)]$$

Let-transition rule

$$(\llbracket \texttt{(let ((}v\ e\texttt{))}\ call\texttt{)} \rrbracket, \hat{\rho}) \leadsto (call, \hat{\rho}'), \text{ where}$$
$$\hat{\rho}' = \hat{\rho} \sqcup [v \mapsto \hat{\mathcal{E}}(e, \hat{\rho})]$$

Example

$$
\begin{aligned}
call_1 &= \texttt{(let ((id (}\lambda\texttt{ (x q) (q x))))} \\
call_2 &= \quad\texttt{(id 3 (}\lambda\texttt{ (v1)} \\
call_3 &= \qquad\texttt{(id 4 (}\lambda\texttt{ (v2)} \\
call_4 &= \qquad\quad\texttt{(halt v2))))}
\end{aligned}
$$

Example

$call_1 =$ `(let ((id (λ (x q) (q x))))`

$call_2 =$ `  (id 3 (λ (v1)`

$call_3 =$ `    (id 4 (λ (v2)`

$call_4 =$ `      (halt v2))))`

Example

$call_1 =$ `(let ((id (λ (x q) (q x))))`

$call_2 =$ `  (id 3 (λ (v1)`

$call_3 =$ `    (id 4 (λ (v2)`

$call_4 =$ `      (halt v2))))`

$$(call_1, \perp)$$

Example

$call_1 =$ `(let ((id (λ (x q) (q x))))`
$call_2 =$ `  (id 3 (λ (v1)`
$call_3 =$ `    (id 4 (λ (v2)`
$call_4 =$ `      (halt v2))))`

$$(call_1, \perp)$$

$$(call_2, [\texttt{id} \mapsto \{\lambda_1\}])$$

Example

$call_1 =$ `(let ((id (λ (x q) (q x))))`

$call_2 =$ `  (id 3 (λ (v1)`

$call_3 =$ `    (id 4 (λ (v2)`

$call_4 =$ `      (halt v2))))`

$$(call_1, \perp)$$

$$(call_2, [\texttt{id} \mapsto \{\lambda_1\}])$$

$$((\texttt{q x}), [\ldots, \texttt{q} \mapsto \{\lambda_2\}, \texttt{x} \mapsto \{3\}])$$

Example

$call_1 =$ `(let ((id (λ (x q) (q x))))`

$call_2 =$ `  (id 3 (λ (v1)`

$call_3 =$ `    (id 4 (λ (v2)`

$call_4 =$ `      (halt v2))))`

$$(call_1, \bot)$$

$$(call_2, [\texttt{id} \mapsto \{\lambda_1\}])$$

$$((\texttt{q x}), [\ldots, \texttt{q} \mapsto \{\lambda_2\}, \texttt{x} \mapsto \{3\}])$$

$$(call_3, [\ldots, \texttt{v1} \mapsto \{3\}])$$

Example

$call_1 =$ `(let ((id (λ (x q) (q x))))`

$call_2 =$ `  (id 3 (λ (v1)`

$call_3 =$ `    (id 4 (λ (v2)`

$call_4 =$ `      (halt v2))))`

$$(call_1, \bot)$$

$$(call_2, [\texttt{id} \mapsto \{\lambda_1\}])$$

$$((\texttt{q x}), [\ldots, \texttt{q} \mapsto \{\lambda_2\}, \texttt{x} \mapsto \{3\}])$$

$$(call_3, [\ldots, \texttt{v1} \mapsto \{3\}])$$

$$((\texttt{q x}), [\ldots, \texttt{q} \mapsto \{\lambda_2, \lambda_3\}, \texttt{x} \mapsto \{3, 4\}])$$

Example

$call_1 =$ `(let ((id (λ (x q) (q x))))`

$call_2 =$ `  (id 3 (λ (v1)`

$call_3 =$ `    (id 4 (λ (v2)`

$call_4 =$ `      (halt v2))))`

$$(call_1, \perp)$$

$$(call_2, [\texttt{id} \mapsto \{\lambda_1\}])$$

$$((\texttt{q x}), [\ldots, \texttt{q} \mapsto \{\lambda_2\}, \texttt{x} \mapsto \{3\}])$$

$$(call_3, [\ldots, \texttt{v1} \mapsto \{3\}])$$

$$((\texttt{q x}), [\ldots, \texttt{q} \mapsto \{\lambda_2, \lambda_3\}, \texttt{x} \mapsto \{3, 4\}])$$

$$((\texttt{halt v2}), [\ldots, \texttt{v2} \mapsto \{3, 4\}])$$

$$(call_3, [\ldots, \texttt{v1} \mapsto \{3, 4\}])$$

Example

$call_1 =$ `(let ((id (λ (x q) (q x))))`
$call_2 =$ `  (id 3 (λ (v1)`
$call_3 =$ `    (id 4 (λ (v2)`
$call_4 =$ `      (halt v2))))`

$$(call_1, \bot)$$
$$(call_2, [\texttt{id} \mapsto \{\lambda_1\}])$$
$$(\texttt{(q x)}, [\ldots, \texttt{q} \mapsto \{\lambda_2\}, \texttt{x} \mapsto \{3\}])$$
$$(call_3, [\ldots, \texttt{v1} \mapsto \{3\}])$$
$$(\texttt{(q x)}, [\ldots, \texttt{q} \mapsto \{\lambda_2, \lambda_3\}, \texttt{x} \mapsto \{3, 4\}])$$
$$(\texttt{(halt v2)}, [\ldots, \texttt{v2} \mapsto \{3, 4\}])$$
$$(call_3, [\ldots, \texttt{v1} \mapsto \{3, 4\}])$$
$$(\texttt{(q x)}, [\ldots])$$

Example

$call_1 =$ `(let ((id (λ (x q) (q x))))`

$call_2 =$ `  (id 3 (λ (v1)`

$call_3 =$ `    (id 4 (λ (v2)`

$call_4 =$ `      (halt v2)))))`

$$(call_1, \bot)$$

$$(call_2, [\texttt{id} \mapsto \{\lambda_1\}])$$

$$(\texttt{(q x)}, [\ldots, \texttt{q} \mapsto \{\lambda_2\}, \texttt{x} \mapsto \{3\}])$$

$$(call_3, [\ldots, \texttt{v1} \mapsto \{3\}])$$

$$(\texttt{(q x)}, [\ldots, \texttt{q} \mapsto \{\lambda_2, \lambda_3\}, \texttt{x} \mapsto \{3, 4\}])$$

$$(\texttt{(halt v2)}, [\ldots, \texttt{v2} \mapsto \{3, 4\}])$$

$$(call_3, [\ldots, \texttt{v1} \mapsto \{3, 4\}])$$

$$(\texttt{(q x)}, [\ldots])$$

$$(\texttt{(halt v2)}, [\ldots])$$

Results

$$\begin{aligned}
\mathrm{FlowsTo}[\mathtt{id}] &= \{\lambda_1\} \\
\mathrm{FlowsTo}[\mathtt{q}] &= \{\lambda_2, \lambda_3\} \\
\mathrm{FlowsTo}[\mathtt{x}] &= \{3, 4\} \\
\mathrm{FlowsTo}[\mathtt{v1}] &= \{3, 4\} \\
\mathrm{FlowsTo}[\mathtt{v2}] &= \{3, 4\}
\end{aligned}$$

Questions?