

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, 1 CFA, 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-
orderliness hard?

Why is higher-order-ness 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-
orderliness is **really** hard?

Which kind of higher-order-ness 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 × Data
- Closure = Lambda × Env
- Object = Class × 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 $(f \ e_1 \ \dots \ e_n)$, what is f ?

Control-flow problem

Given a call site $(f \ e_1 \ \dots \ e_n)$, what is f ?

Given a call site $o.m(e_1, \dots, e_n)$, what is o ?

Control-flow scenarios

$f(x)$

Control-flow scenarios

```
let f = λz.z  
    in f(x)
```

Control-flow scenarios

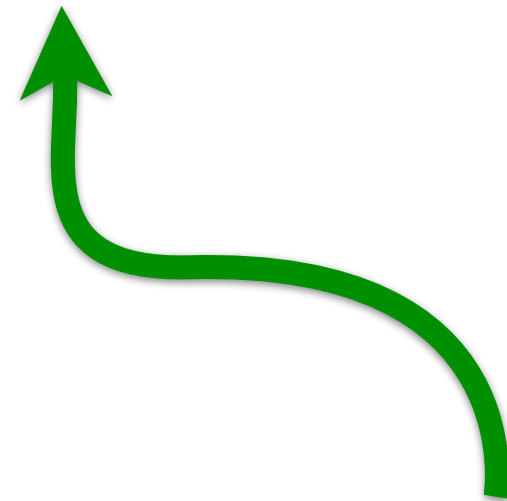
$\lambda f.f(x)$

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`

`FlowsTo[f] = {h}`

Example

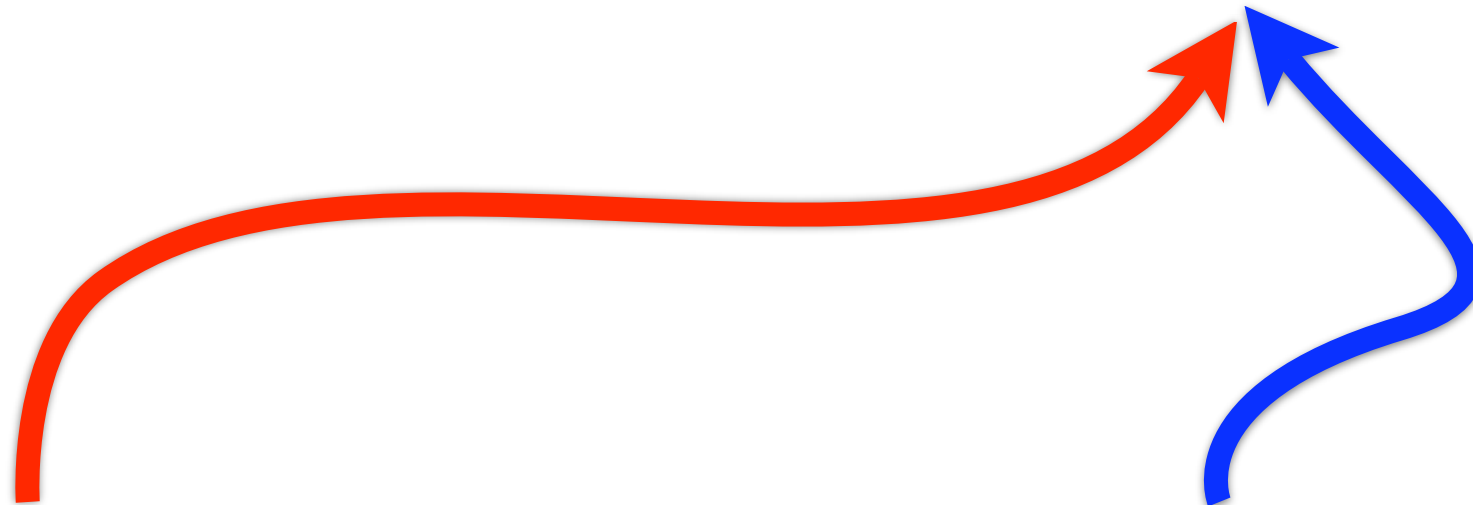
`apply f x = f(x)`

`h(x) = x + 1`

`g(x) = apply h 4`

`apply g 3`

`FlowsTo[f] = {h,g}`

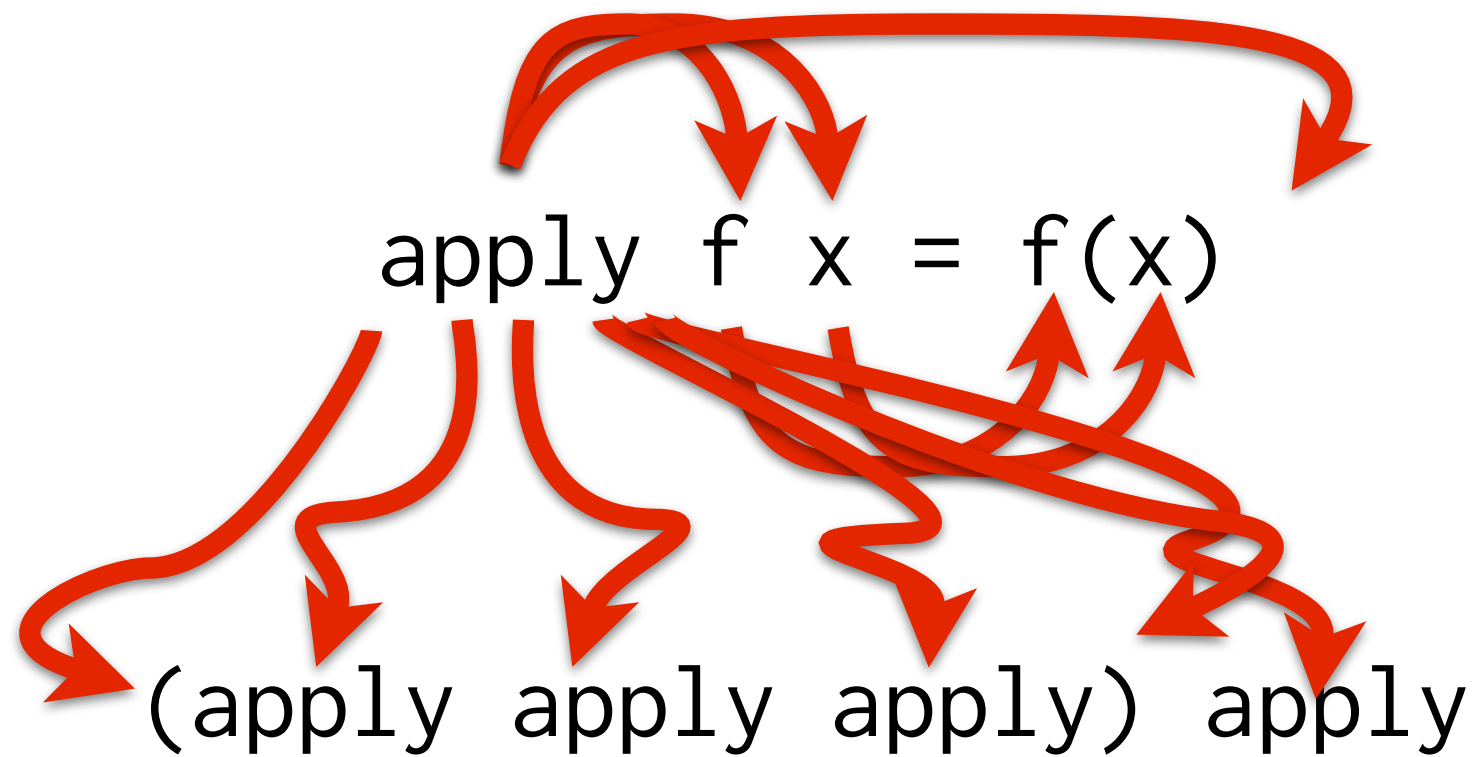


Example

`apply f x = f(x)`

`(apply apply apply) apply`

Example



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

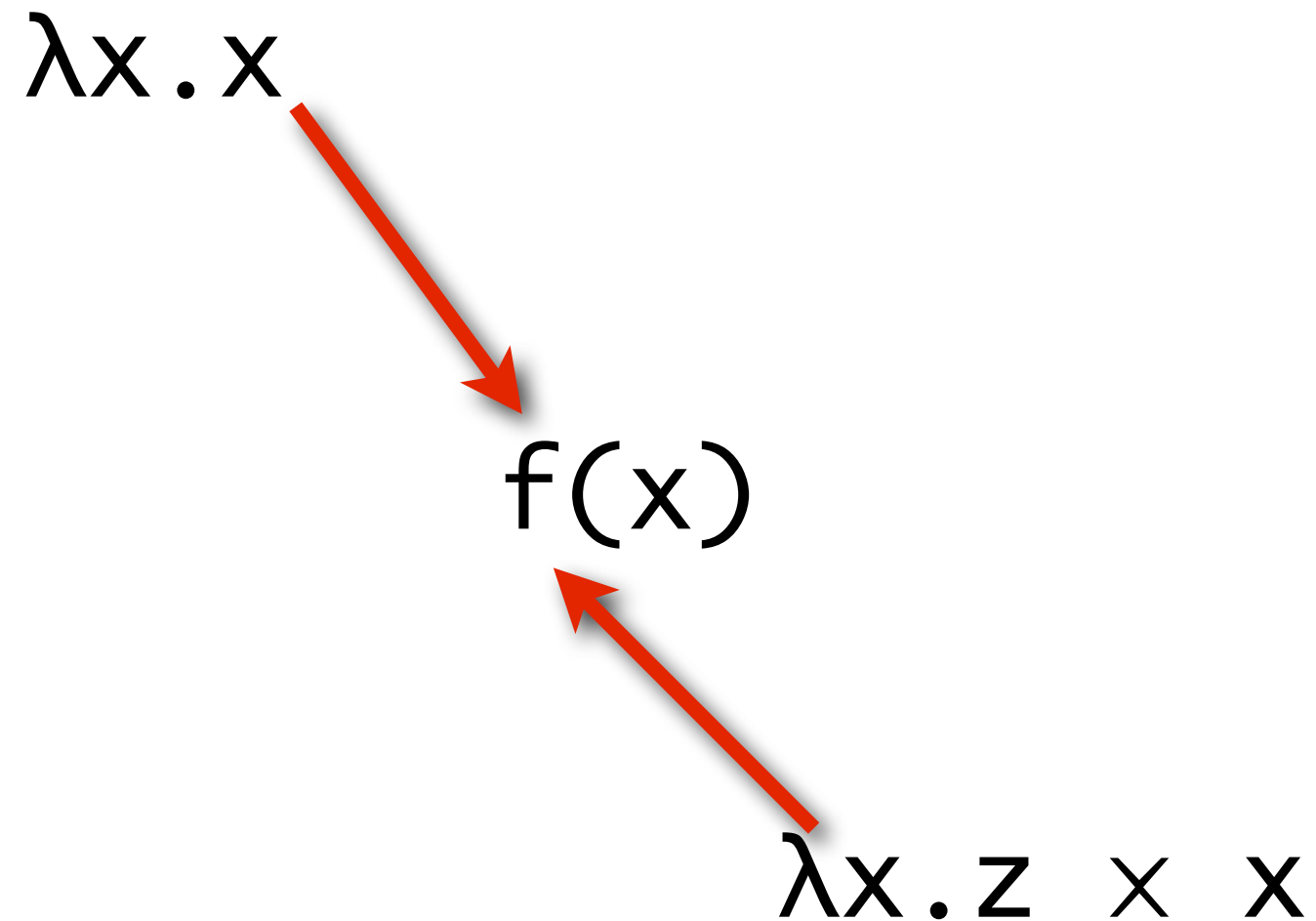
A **control-flow analysis** conservatively approximates the procedures which may be invoked at a given call site.

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:** $\hat{f} \in \{\llbracket \lambda x.x + z \rrbracket, \llbracket \lambda x.x \times z \rrbracket\}$
- **Intermediate:** $f \in \{\lambda x.x + z \mid z \in \mathbb{N}_2\}$

Environment matters



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 ?

T

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

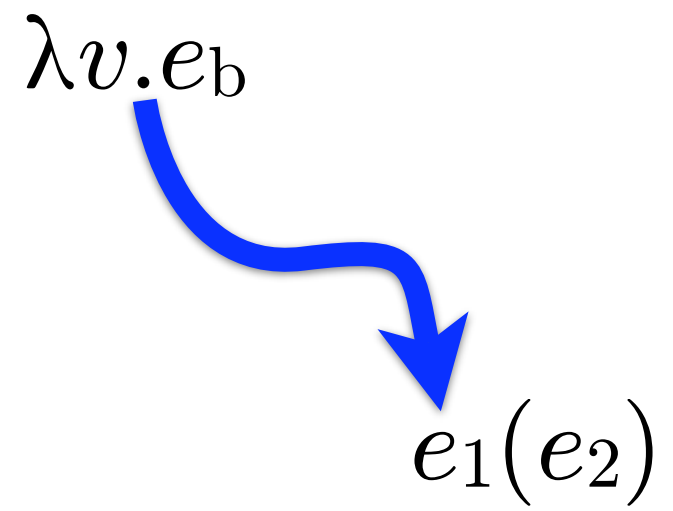
The 0CFA approximation

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

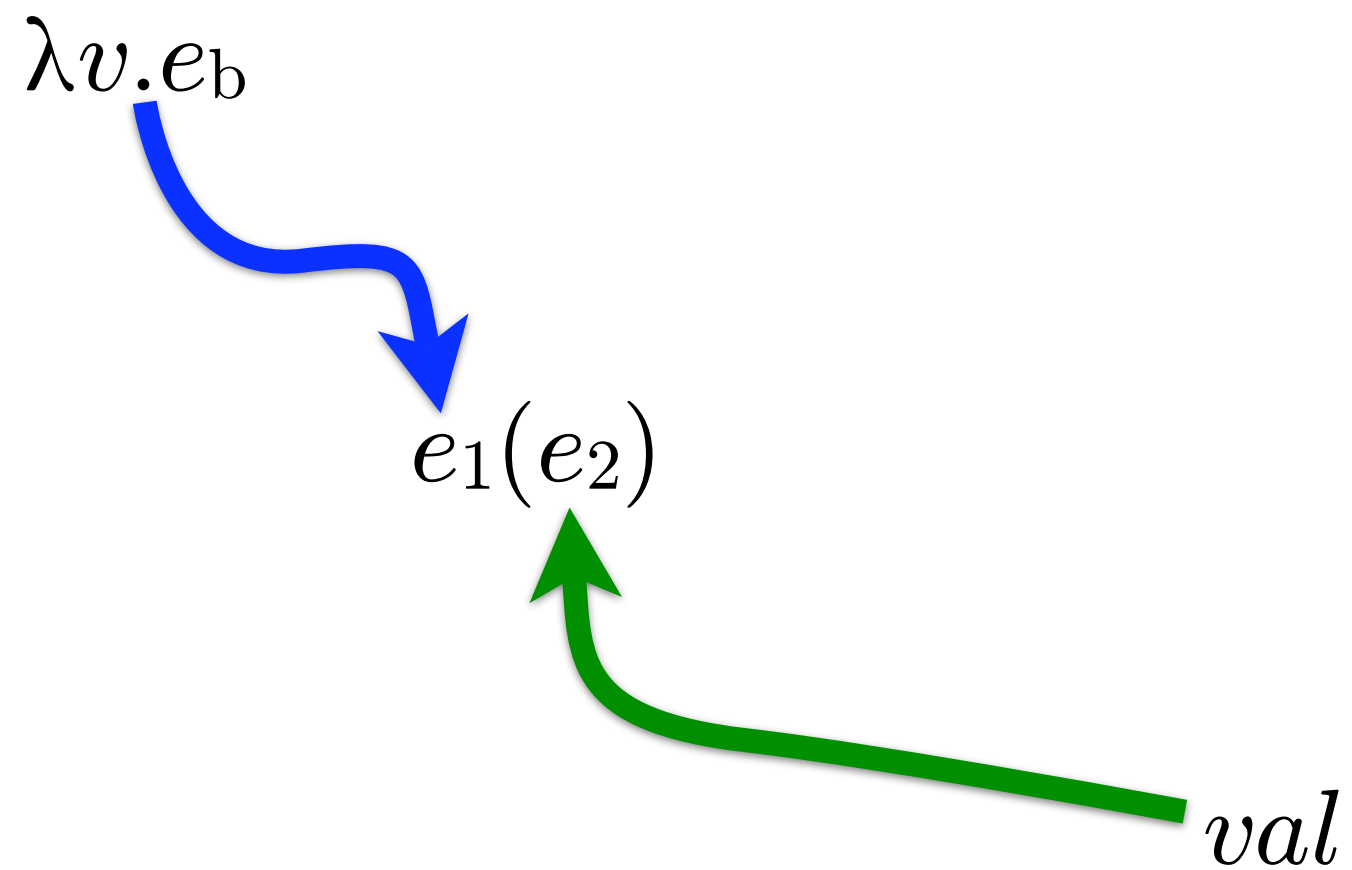
OCFA

$$e_1(e_2)$$

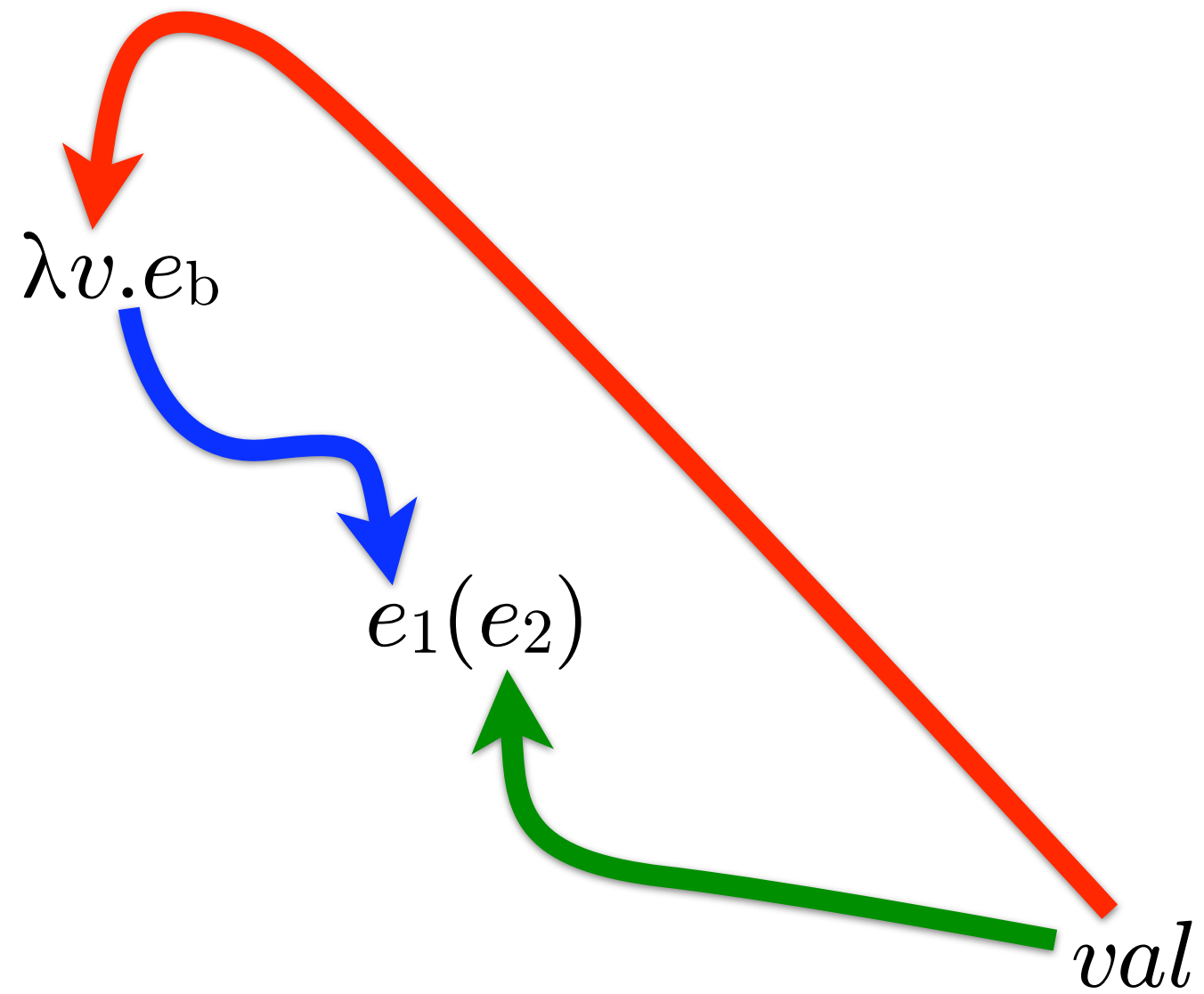
OCFA



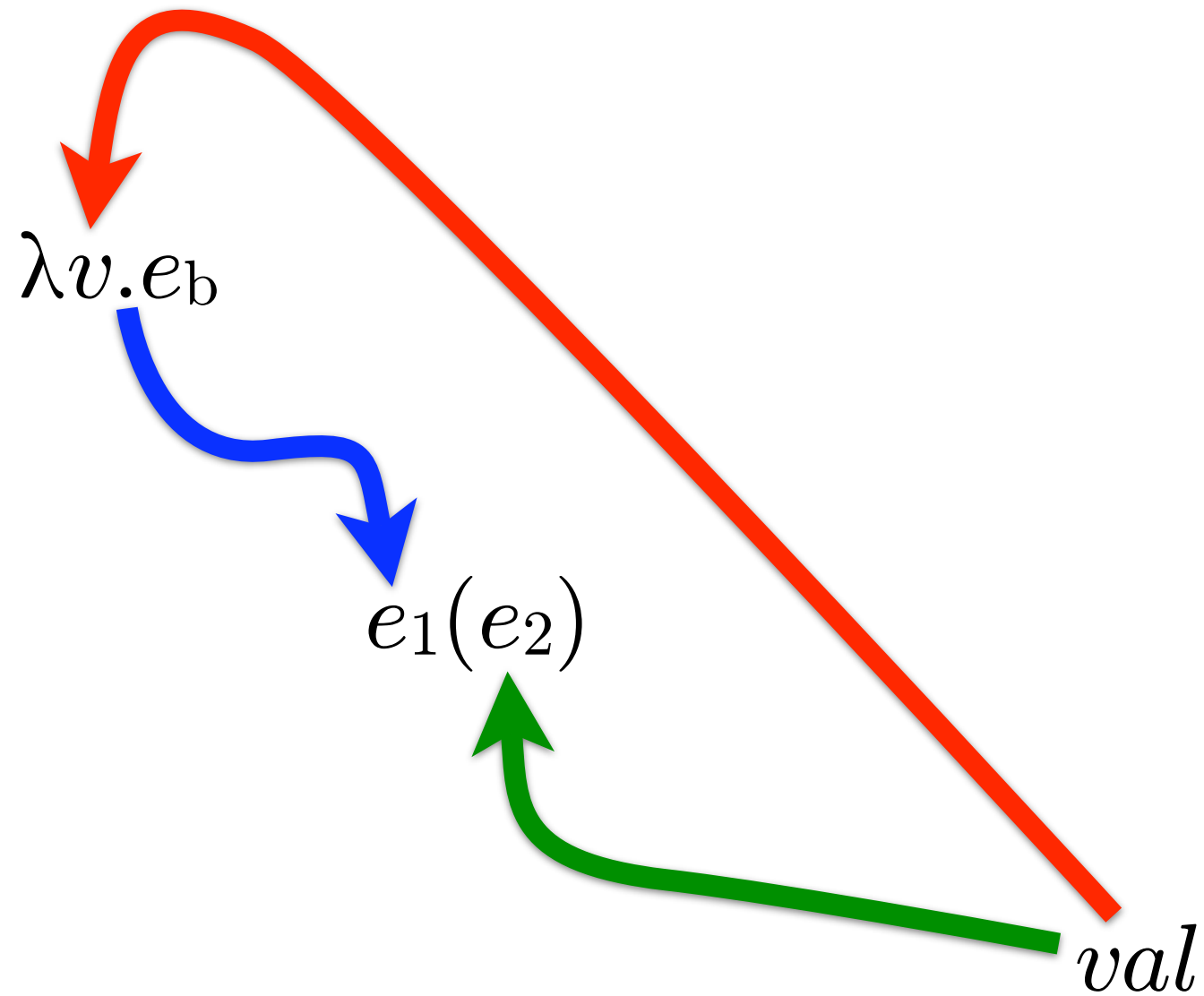
OCFA



OCFA

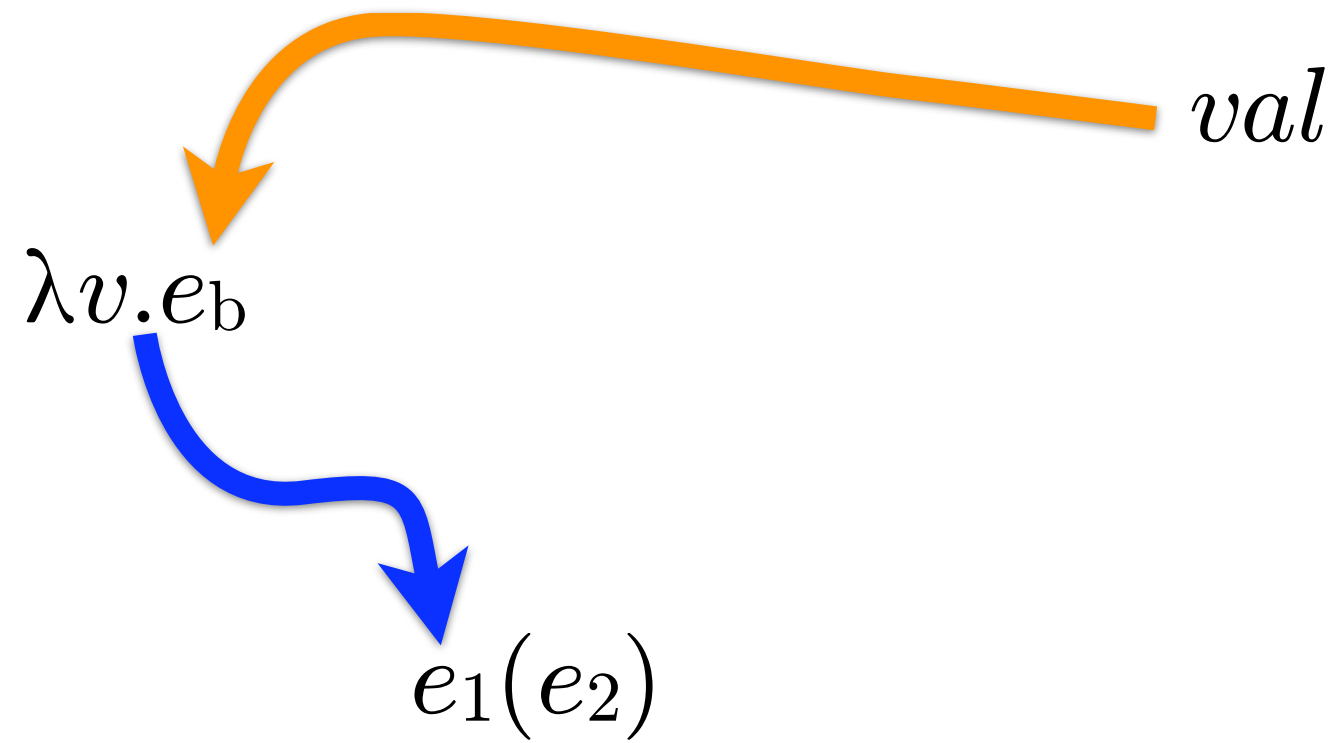


OCFA

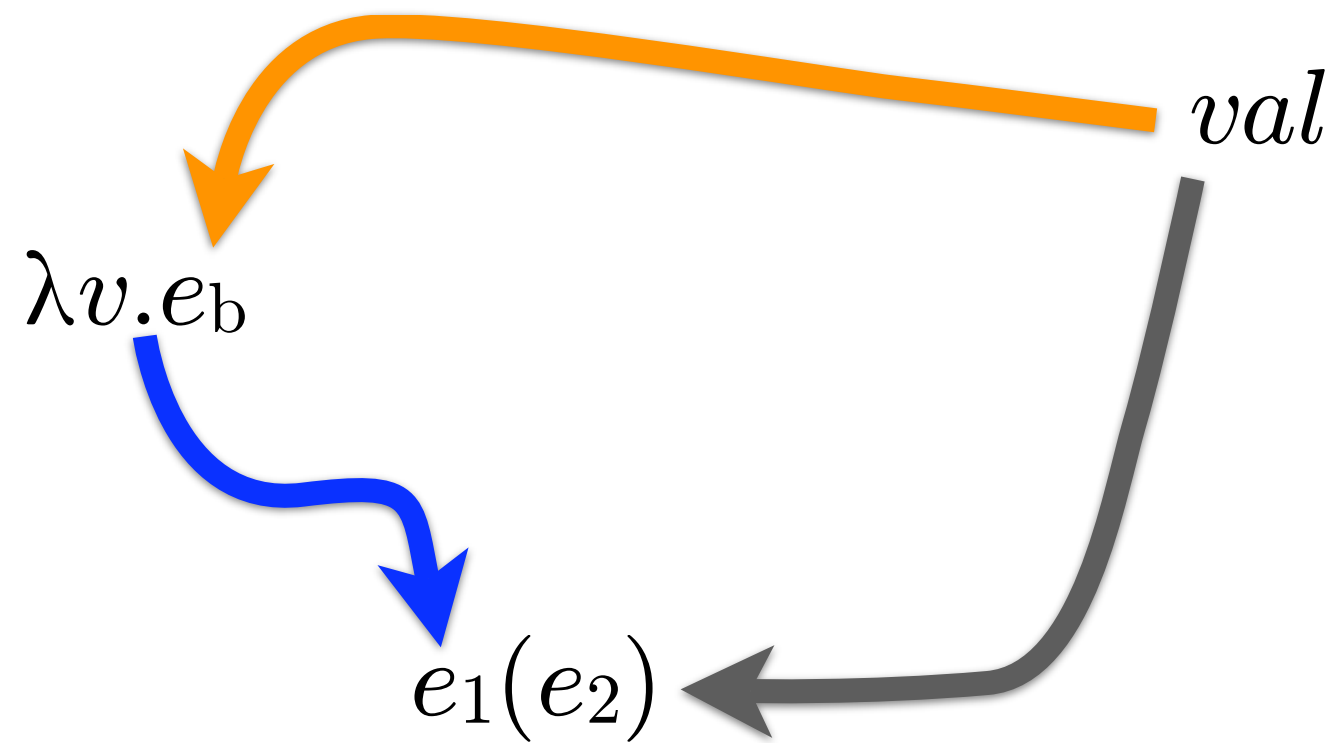


$$\frac{\lambda v.e_b \in \text{FlowsTo}[e_1] \quad \text{and} \quad val \in \text{FlowsTo}[e_2]}{val \in \text{FlowsTo}[v]}$$

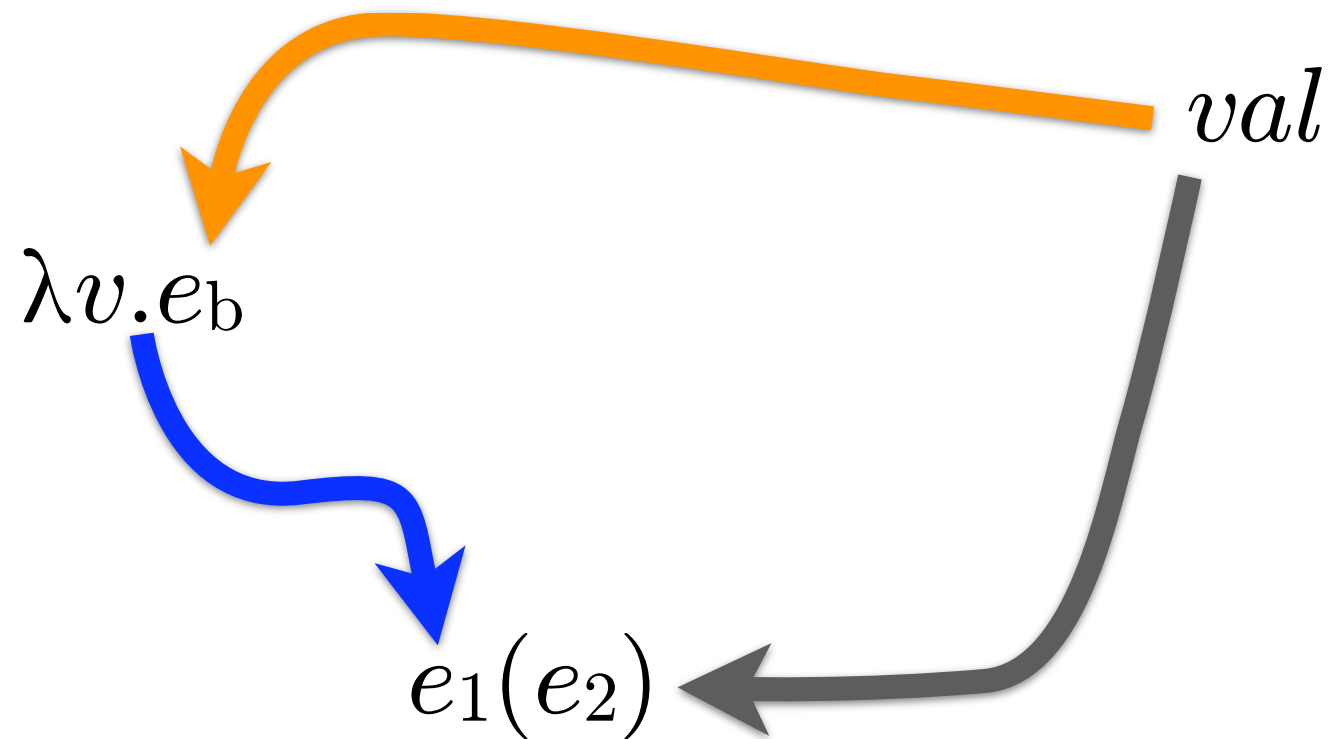
OCFA



OCFA



OCFA



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

OCFA

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

OCFA

$$\lambda v.e_b \in \text{FlowsTo}[\lambda v.e_b]$$

$$\frac{\lambda v.e_b \in \text{FlowsTo}[e_1] \quad \text{and} \quad \textcolor{brown}{val} \in \text{FlowsTo}[e_b]}{\textcolor{gray}{val} \in \text{FlowsTo}[e_1(e_2)]}$$

$$\frac{\lambda v.e_b \in \text{FlowsTo}[e_1] \quad \text{and} \quad \textcolor{green}{val} \in \text{FlowsTo}[e_2]}{\textcolor{red}{val} \in \text{FlowsTo}[v]}$$

OCFA (Palsberg, 1995)

$$\{\lambda v.e_b\} \subseteq \text{FlowsTo}[\lambda v.e_b]$$

$$\frac{\lambda v.e_b \in \text{FlowsTo}[e_1]}{\text{FlowsTo}[e_b] \subseteq \text{FlowsTo}[e_1(e_2)]}$$

$$\frac{\lambda v.e_b \in \text{FlowsTo}[e_1]}{\text{FlowsTo}[e_2] \subseteq \text{FlowsTo}[v]}$$

Computing 0CFA

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

Example: OCFA

```
let id =  $\lambda x.x$   
    v1 = id(3)  
    v2 = id(4)  
in v2
```



Example: OCFA



```
let id =  $\lambda x.x$   
    v1 = id(3)  
    v2 = id(4)  
in v2
```



Example: OCFA

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

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

Example: OCFA

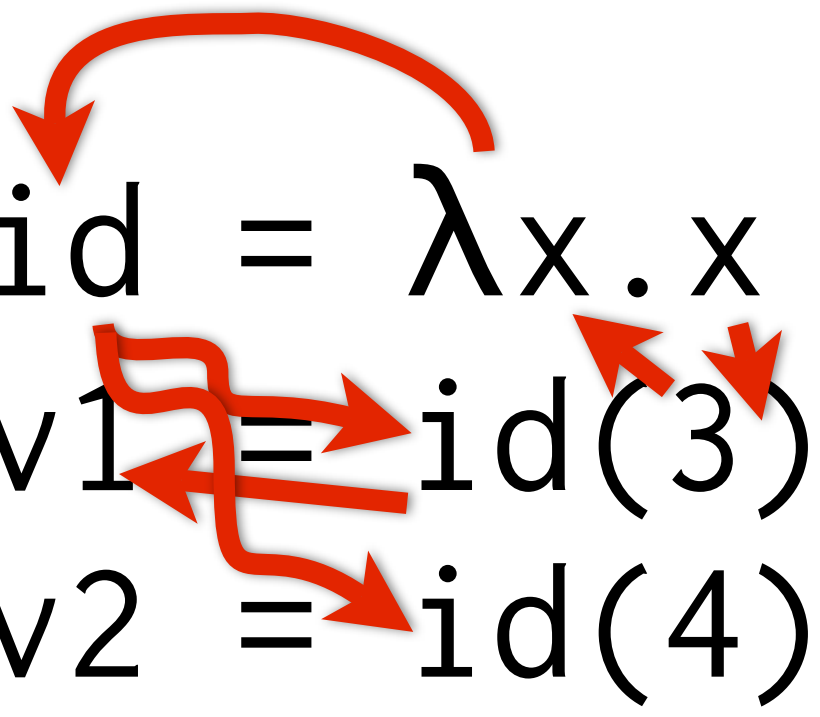
```
let id =  $\lambda x.x$   
    v1 = id(3)  
    v2 = id(4)  
in v2
```

The diagram illustrates the flow of information in the OCFA (Optimistic Call-Frame Analysis) for the provided code. Red arrows indicate the flow of information from the lambda abstraction $\lambda x.x$ to its uses in the function calls $\text{id}(3)$ and $\text{id}(4)$. Specifically, an arrow points from the lambda abstraction to the id variable in both function calls. Another arrow points from the id variable in the first function call to the id variable in the second function call, indicating that the flow set for id is used in the second call. A third arrow points from the id variable in the second function call to the id variable in the let binding, indicating that the flow set for id is used in the binding.

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

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

Example: OCFA



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

The diagram illustrates the flow of information for the OCFA analysis. Red arrows show the following connections:

- A curved arrow from the `id` variable in the `let` binding to the `id` in the `FlowsTo[id]` set.
- A straight arrow from the `x` parameter in the lambda expression `λx.x` to the `x` in the `FlowsTo[x]` set.
- A straight arrow from the `3` argument in `id(3)` to the `3` in the `FlowsTo[id(3)]` set.
- A straight arrow from the `4` argument in `id(4)` to the `3` in the `FlowsTo[id(3)]` set, indicating that the value 4 is mapped to the set {3}.

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

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

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

Example: OCFA

let $id = \lambda x.x$
 $v1 = id(3)$
 $v2 = id(4)$
in $v2$

The diagram illustrates the flow of information in the OCFA (Optimistic Call Frame Analysis) for the given code. Red arrows show the following flow sets:

- $FlowsTo[id] = \{\lambda x.x\}$: The flow set for the variable `id` is the lambda expression $\lambda x.x$.
- $FlowsTo[x] = \{3\}$: The flow set for the parameter `x` is the value 3.
- $FlowsTo[id(3)] = \{3\}$: The flow set for the expression `id(3)` is the value 3.
- $FlowsTo[v1] = \{3\}$: The flow set for the variable `v1` is the value 3.

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

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

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

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

Example: OCFA

```
let id =  $\lambda x.x$   
    v1 = id(3)  
    v2 = id(4)  
in v2
```

The diagram illustrates the OCFA (Optimistic Call Frame Analysis) flow sets for the provided code. Red arrows show the flow of information from the code to the flow set definitions:

- A curved arrow points from the `id` binding in the `let` statement to the `FlowsTo[id]` definition.
- Arrows point from the `id` and `3` in `id(3)` to the `FlowsTo[x]` definition.
- Arrows point from the `id` and `4` in `id(4)` to the `FlowsTo[id(3)]` definition.
- An arrow points from the `v1` binding to the `FlowsTo[v1]` definition.

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

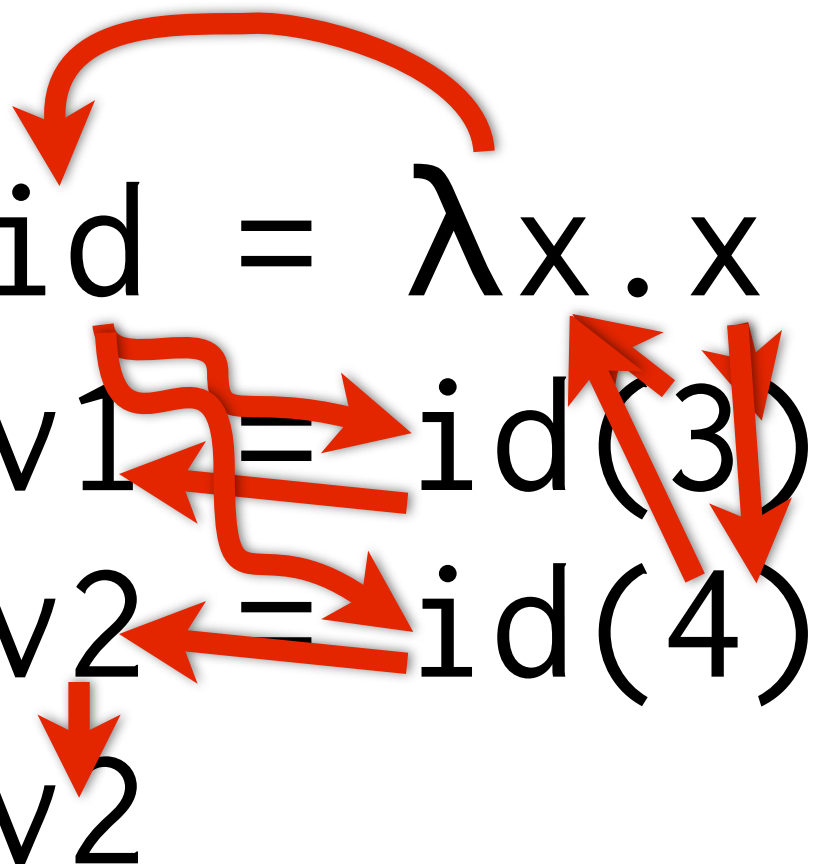
$\text{FlowsTo}[x] = \{3, 4\}$

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

$\text{FlowsTo}[v1] = \{3\}$

Example: OCFA

let $id = \lambda x.x$
 $v1 = id(3)$
 $v2 = id(4)$
in $v2$



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

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

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

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

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

Example: OCFA

let $id = \lambda x.x$
 $v1 = id(3)$
 $v2 = id(4)$
in $v2$

The diagram illustrates the flow of information in the OCFA analysis. Red arrows show the following flow sets:

- id flows to $\lambda x.x$ (curved arrow from id to $\lambda x.x$).
- x flows to 3 and 4 (straight arrows from x to the arguments of id).
- $id(3)$ flows to 3 (straight arrow from $id(3)$ to 3).
- $id(4)$ flows to 3 and 4 (straight arrows from $id(4)$ to 3 and 4).
- $v1$ flows to 3 (straight arrow from $v1$ to 3).
- $v2$ flows to $v2$ (straight arrow from $v2$ to the final $v2$).

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

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

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

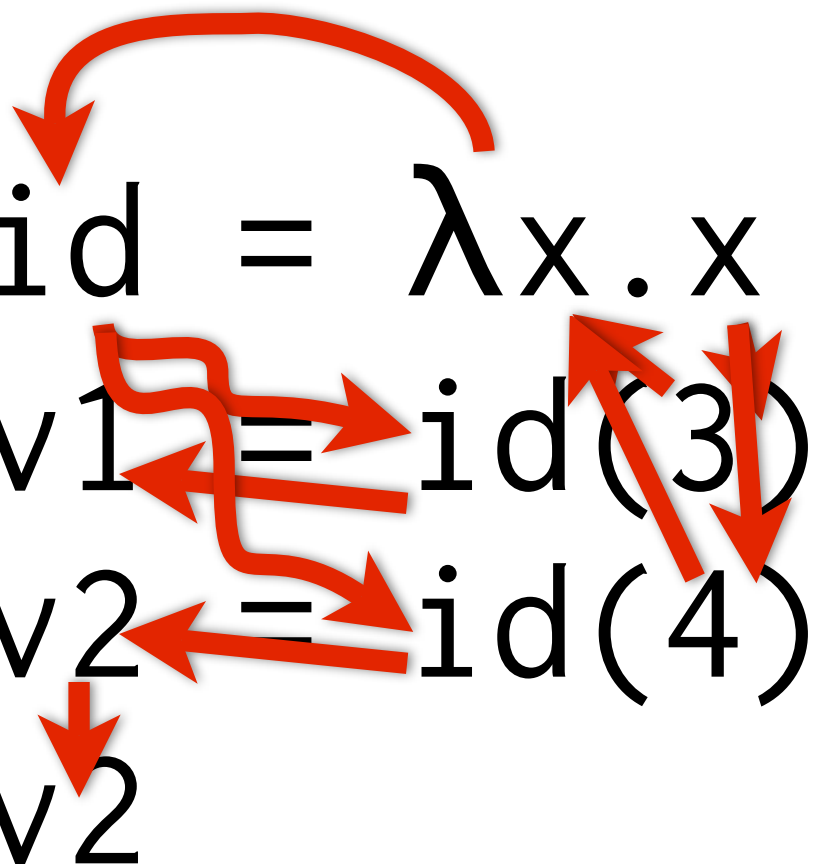
$FlowsTo[v1] = \{3\}$

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

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

Example: OCFA

let $id = \lambda x.x$
 $v1 = id(3)$
 $v2 = id(4)$
in $v2$



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

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

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

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

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

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

Example: OCFA

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

The diagram illustrates the flow of information in the OCFA (Optimistic Call-Frame Analysis) for the given code. Red arrows show the following flows:

- From the lambda abstraction $\lambda x.x$ to the argument 3 in $id(3)$.
- From the lambda abstraction $\lambda x.x$ to the argument 4 in $id(4)$.
- From the argument 3 to the variable $v1$.
- From the argument 4 to the variable $v2$.
- From the variable $v2$ to the final result $v2$ in the `in` block.

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

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

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

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

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

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

Equality-based CFA

OCFA

(Palsberg, 1995)

$$\{\lambda v.e_b\} \subseteq \text{FlowsTo}[\lambda v.e_b]$$

$$\frac{\lambda v.e_b \in \text{FlowsTo}[e_1]}{\text{FlowsTo}[e_b] \subseteq \text{FlowsTo}[e_1(e_2)]}$$

$$\frac{\lambda v.e_b \in \text{FlowsTo}[e_1]}{\text{FlowsTo}[e_2] \subseteq \text{FlowsTo}[v]}$$

Equality CFA (Heinglen, 1992)

$$\{\lambda v.e_b\} \subseteq \text{FlowsTo}[\lambda v.e_b]$$

$$\frac{\lambda v.e_b \in \text{FlowsTo}[e_1]}{\text{FlowsTo}[e_b] = \text{FlowsTo}[e_1(e_2)]}$$

$$\frac{\lambda v.e_b \in \text{FlowsTo}[e_1]}{\text{FlowsTo}[e_2] = \text{FlowsTo}[v]}$$

Computing (=)CFA

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

Example: (=)CFA

```
let id =  $\lambda x. x$   
    v1 = (id 3)  
    v2 = (id 4)  
in v2
```

Example: (=)CFA

```
let id =  $\lambda x. x$   
    v1 = (id 3)  
    v2 = (id 4)  
in v2
```

Type-based OCFA

Typed lambda calculus

$$t ::= e : \tau$$

$$e ::= v$$

$$| \lambda^\ell v. t$$

$$| t \ t'$$

Flow types

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

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

$$\begin{aligned} &((\lambda^A f.(f\ f)) \\ &(\lambda^B f.(f\ f))) \end{aligned}$$

Example: Omega

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

Example: Omega

$$\begin{aligned} &((\lambda^A f.(f : \{B\} \ f : \{B\}) : \{\}) : \{A\}) \\ &(\lambda^B f.(f : \{B\} \ f : \{B\}) : \{\}) : \{B\}) : \{\} \end{aligned}$$

Flow-typed functions

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

Flow-typed functions

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

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

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

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

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

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

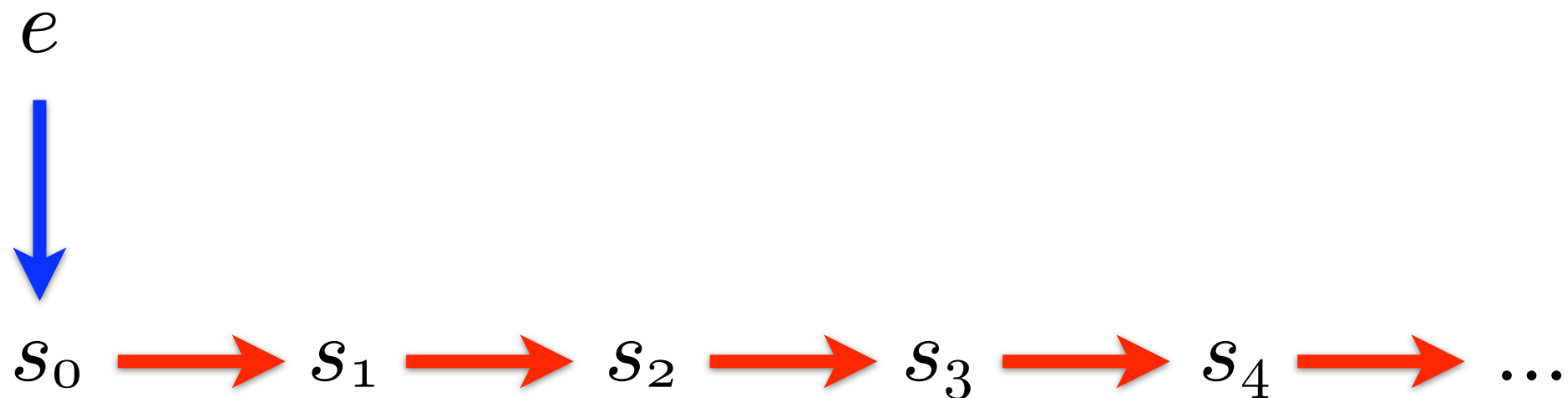
Small-step machine

- Convert program e into machine state s_0

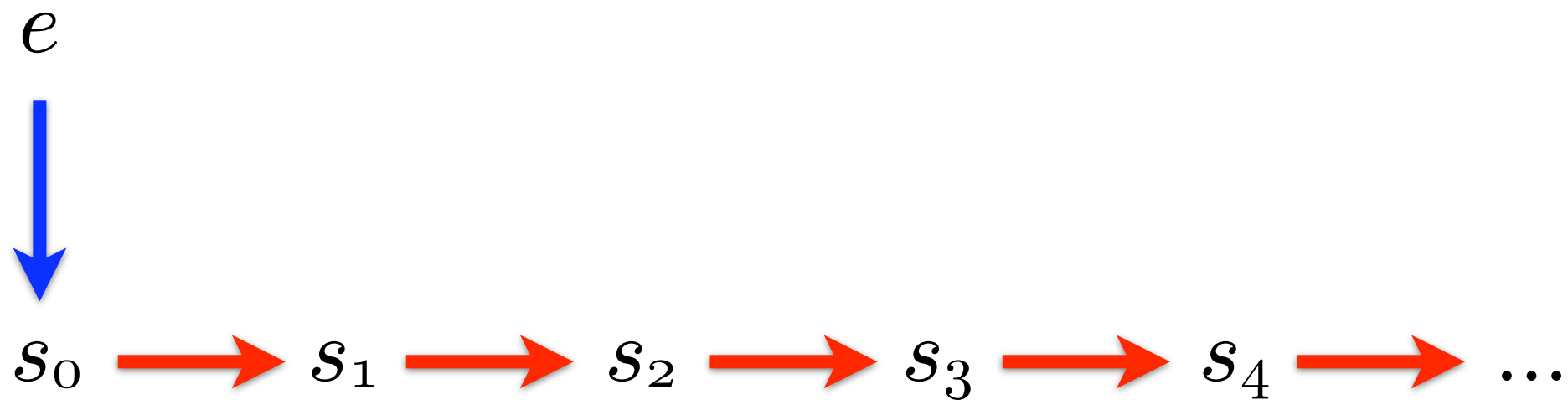


Small-step machine

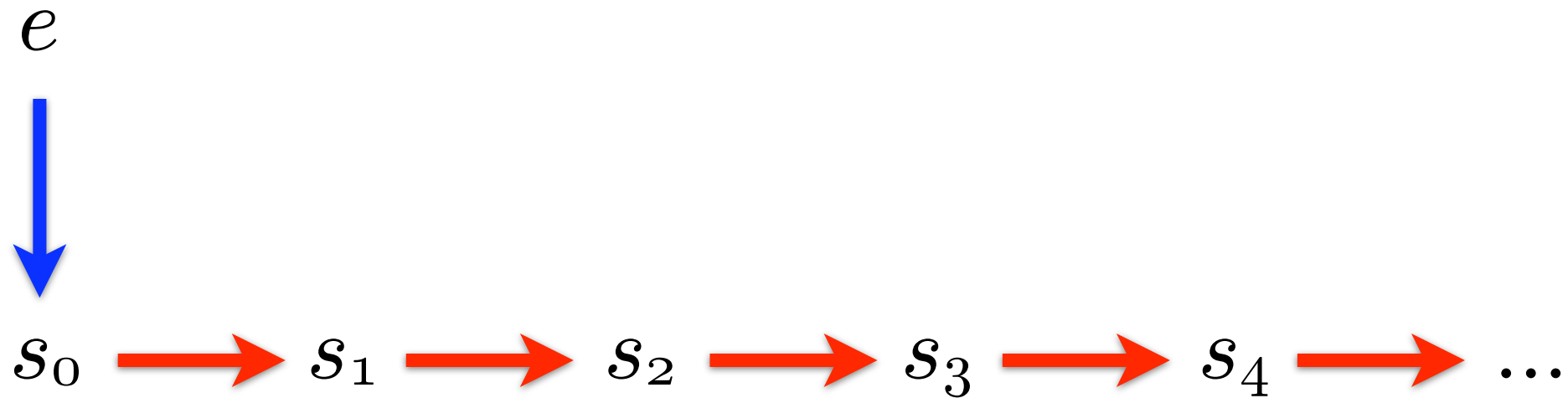
- Convert program e into machine state s_0
- Transition from state s_n to state s_{n+1}



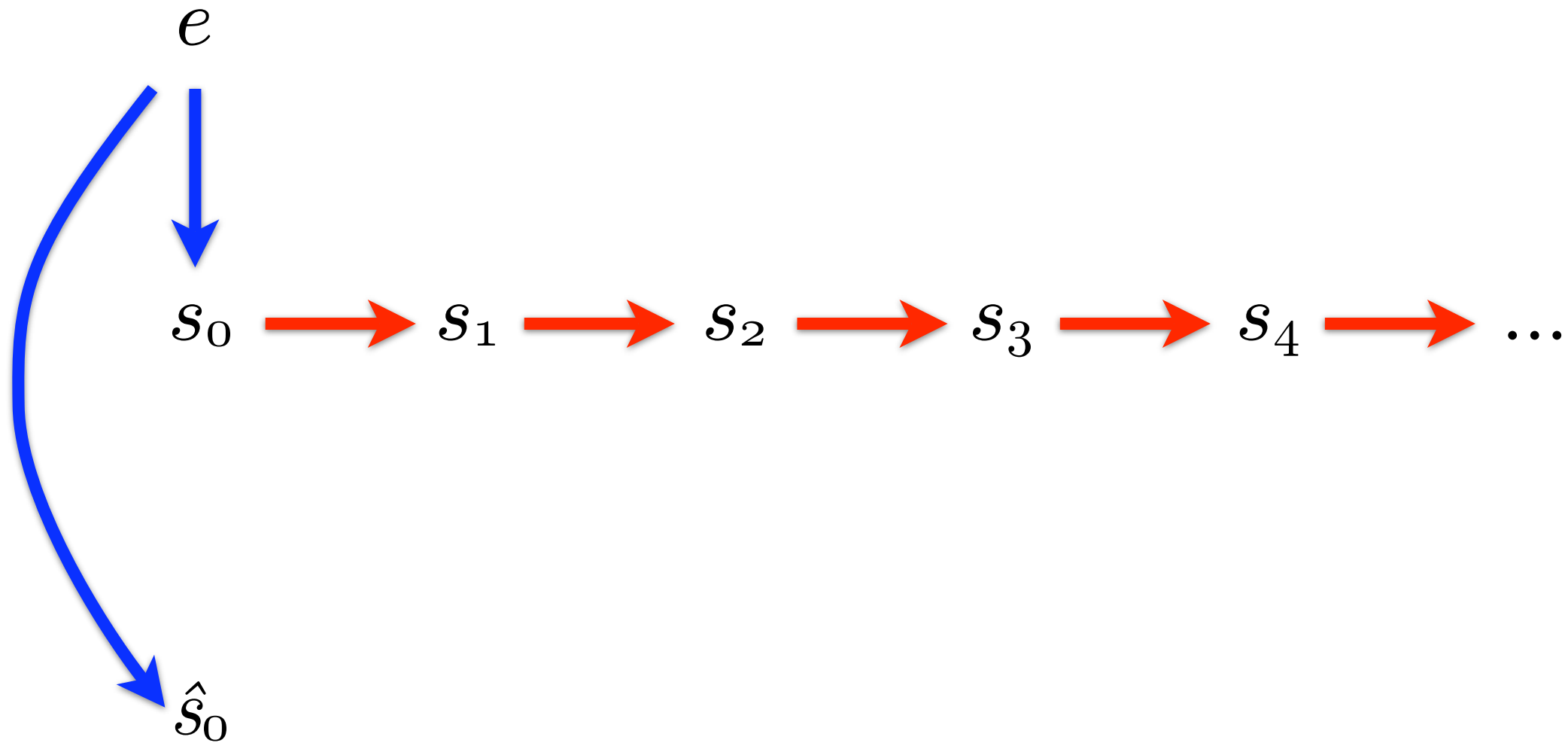
Abstract machine



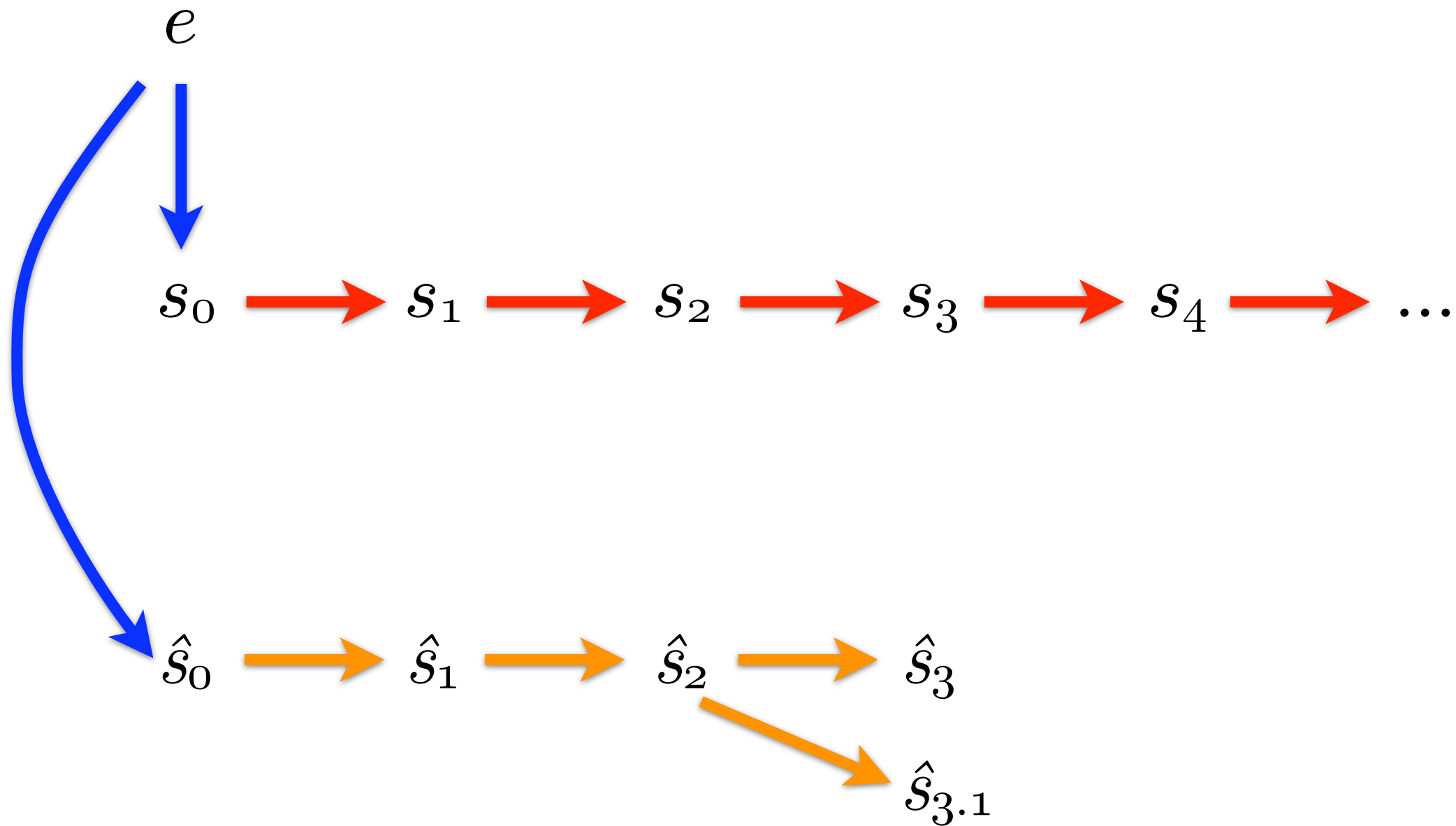
Abstract machine



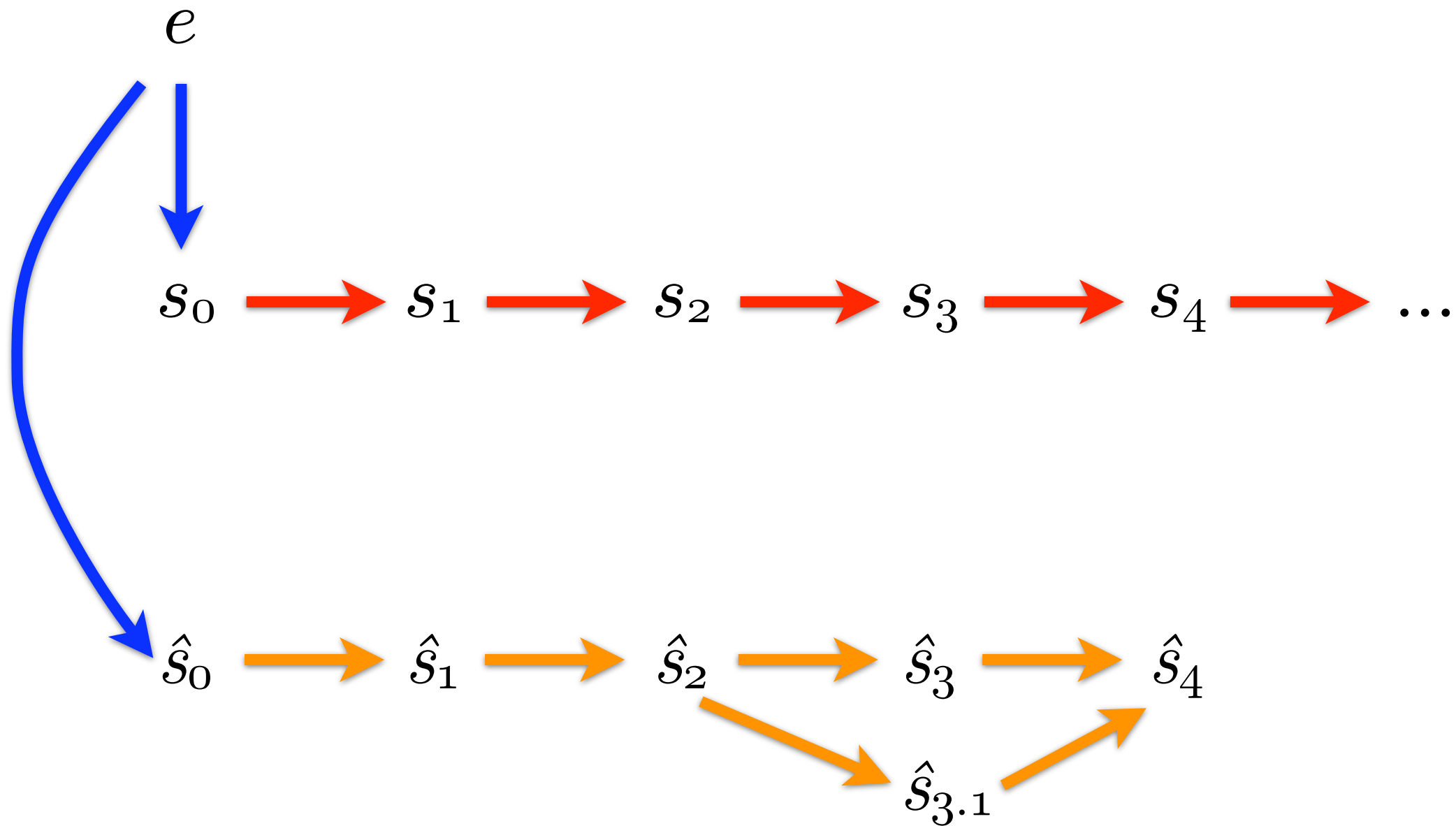
Abstract machine



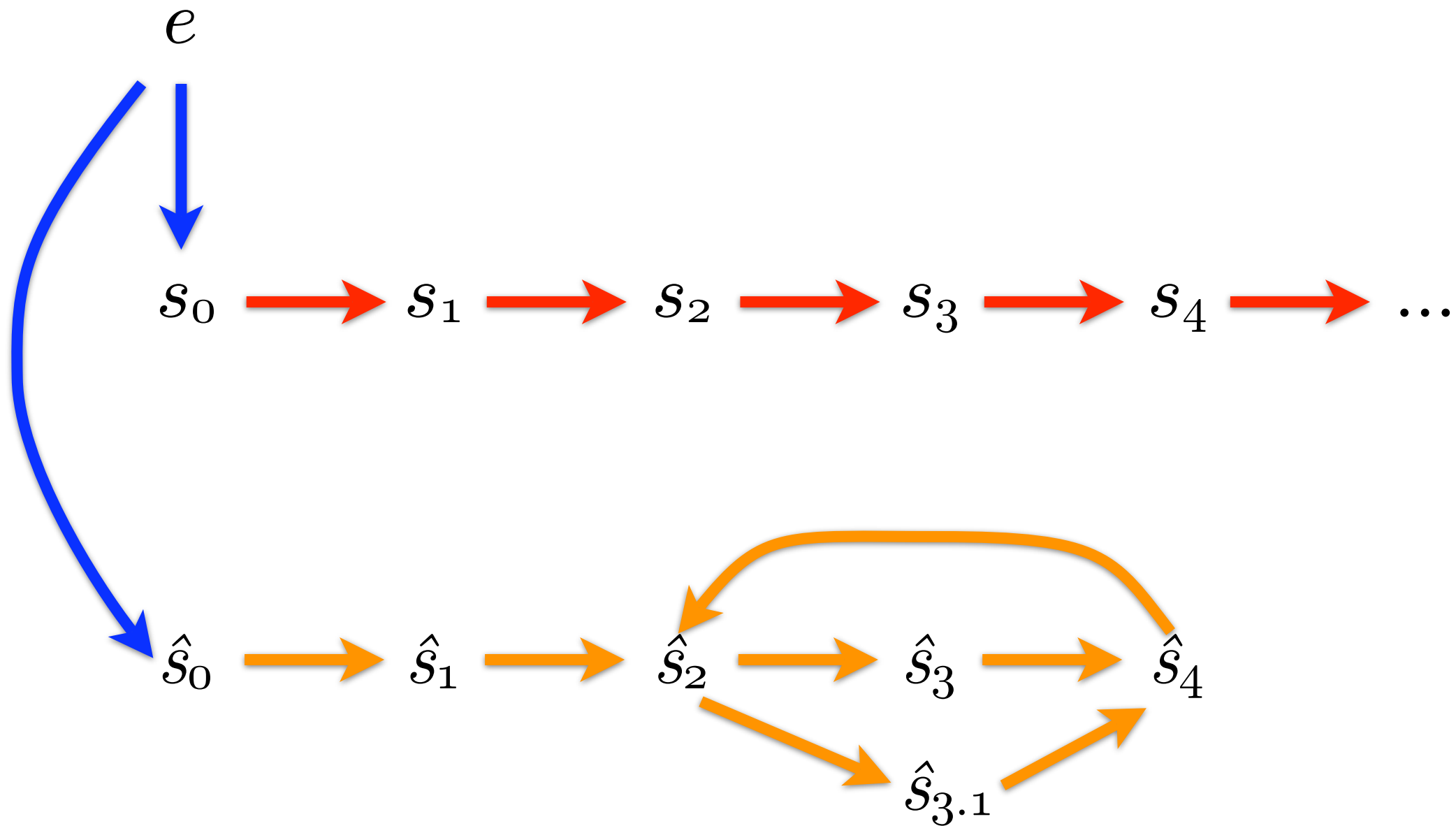
Abstract machine



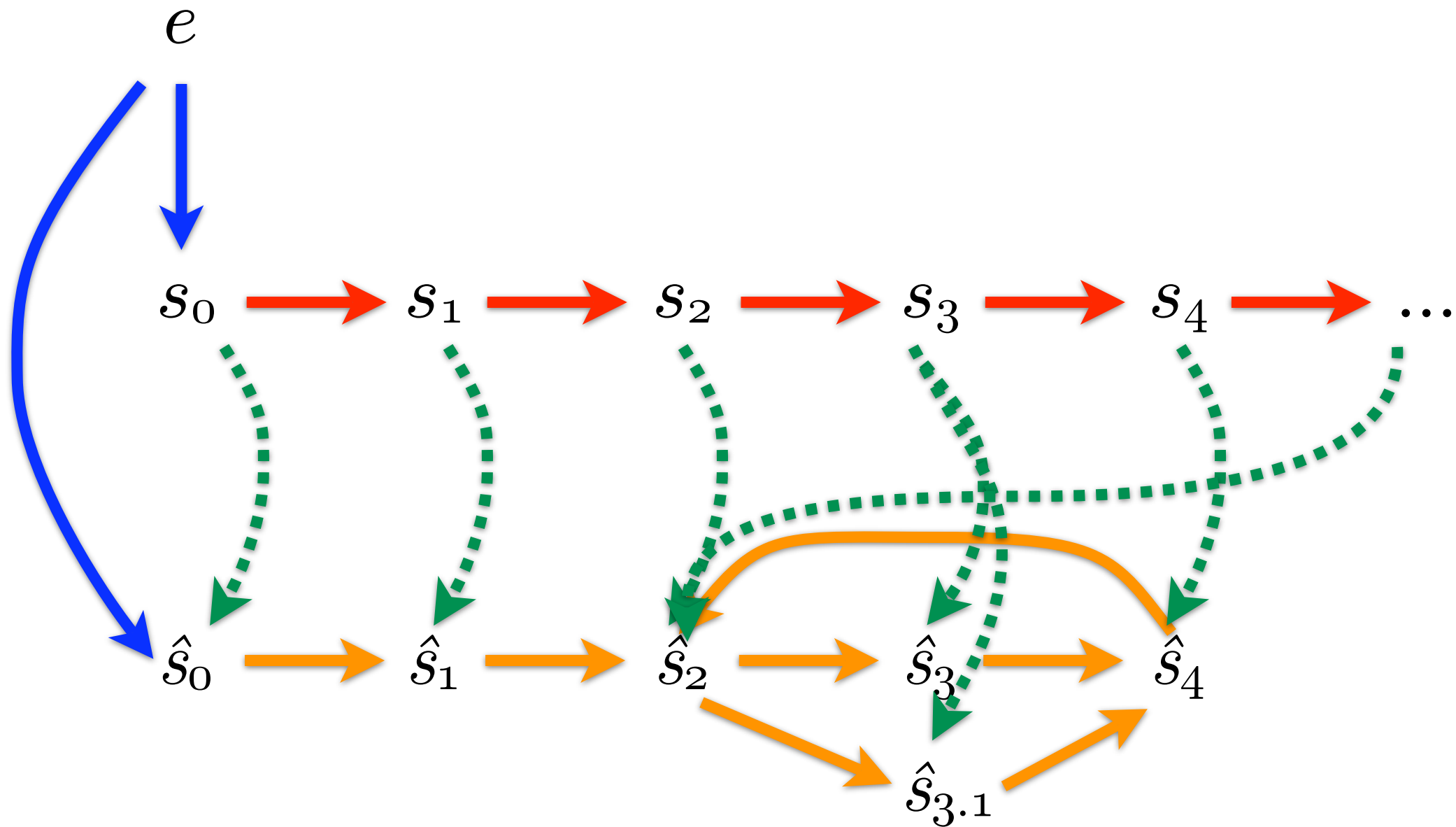
Abstract machine



Abstract machine



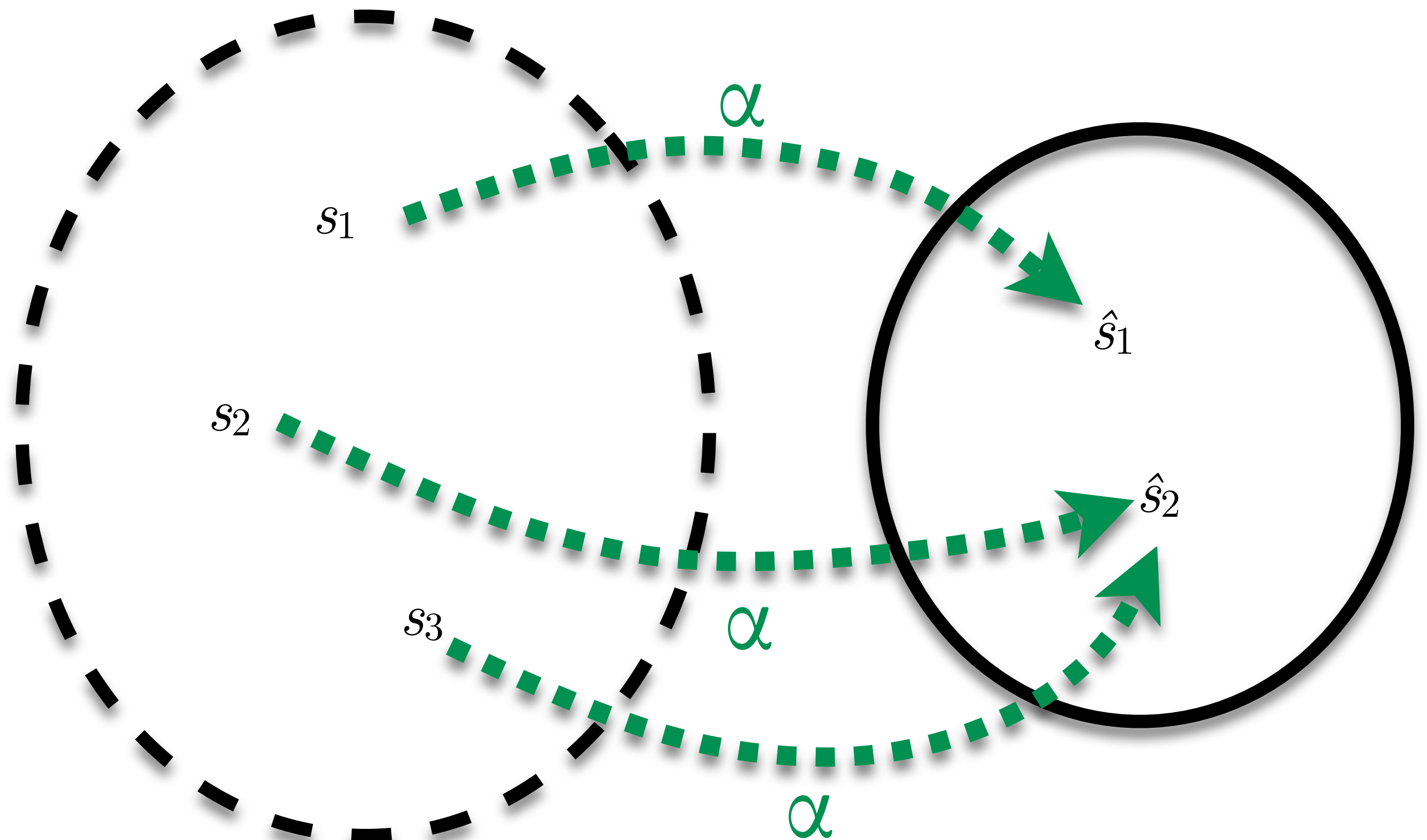
Abstract machine



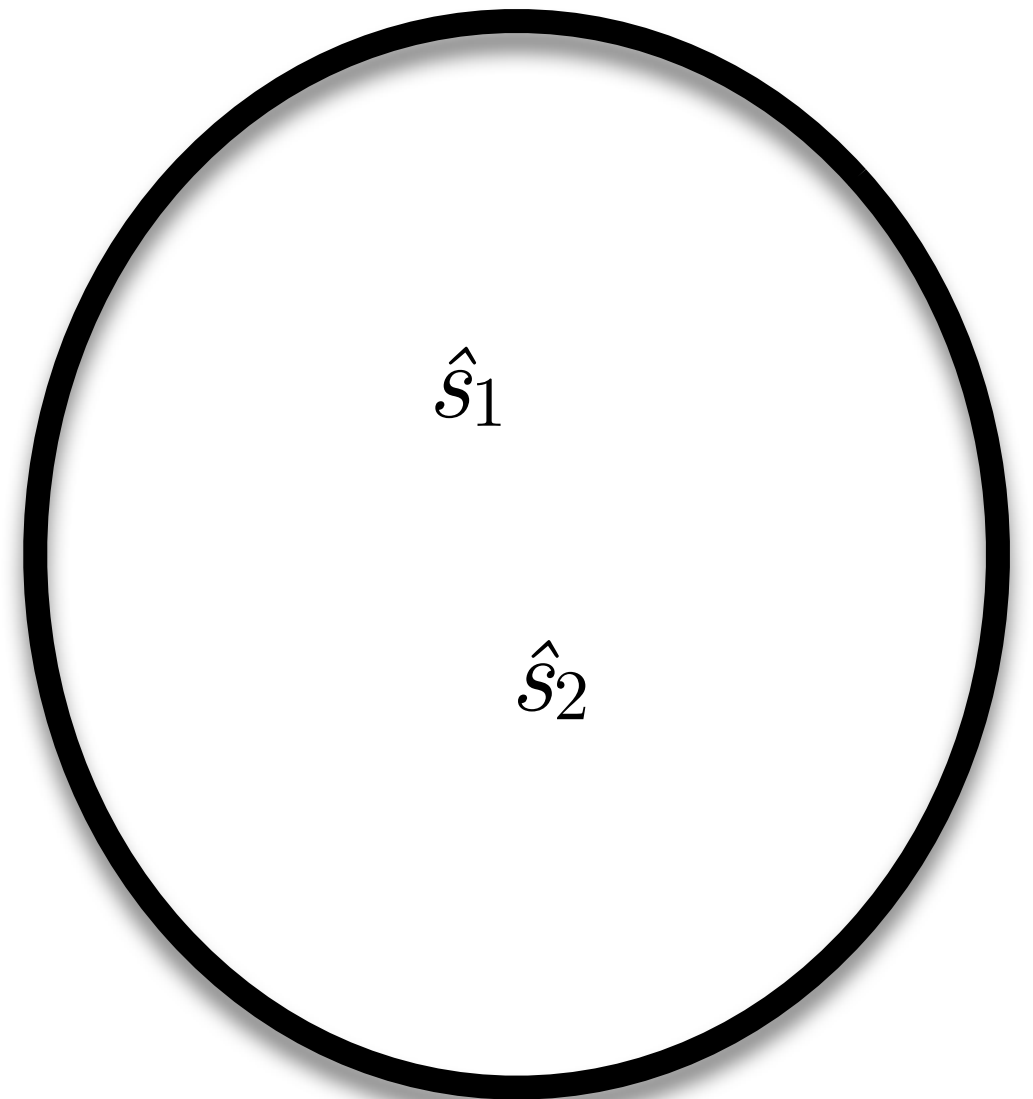
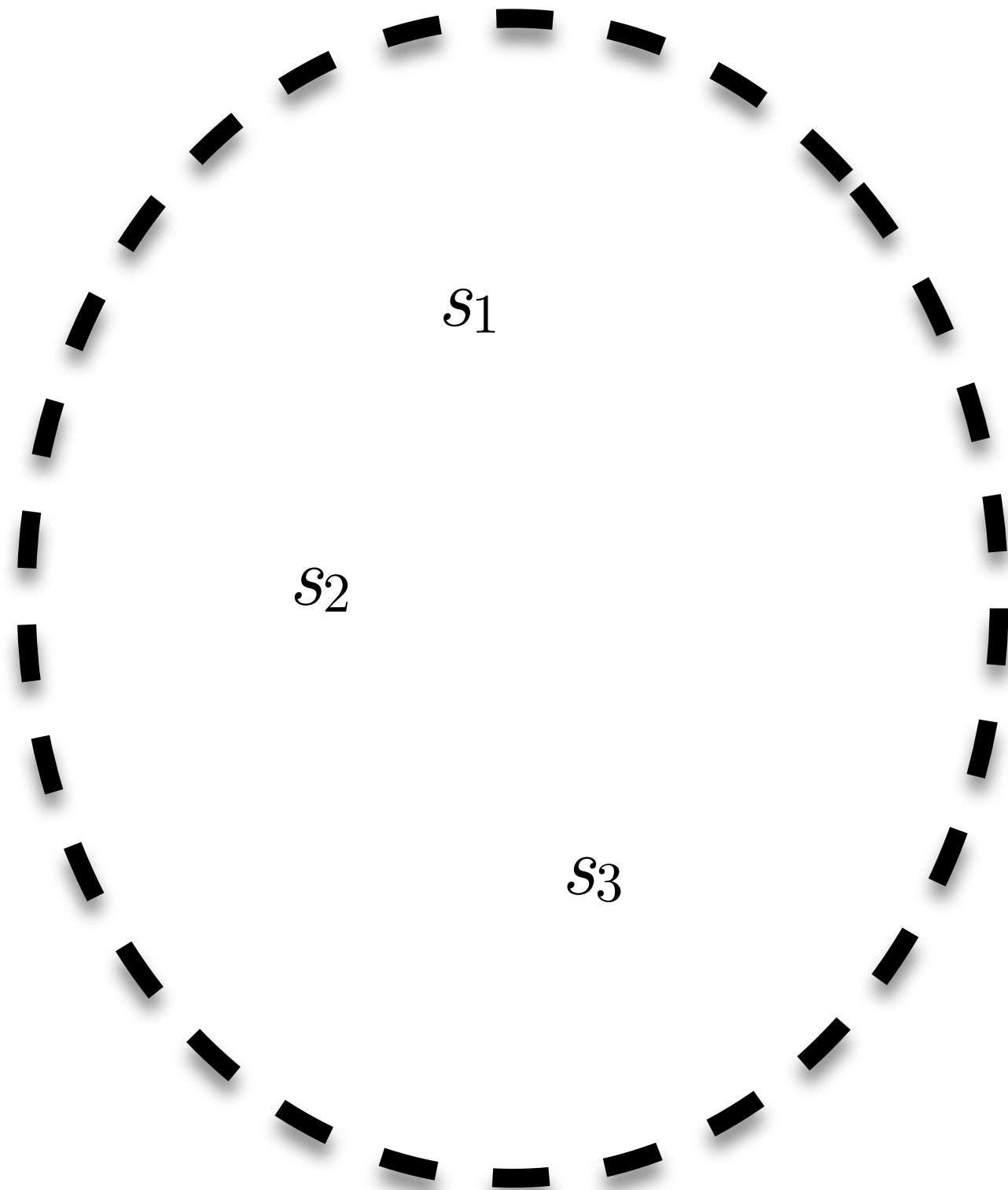
Theorem: The abstract simulates the concrete.

Abstraction

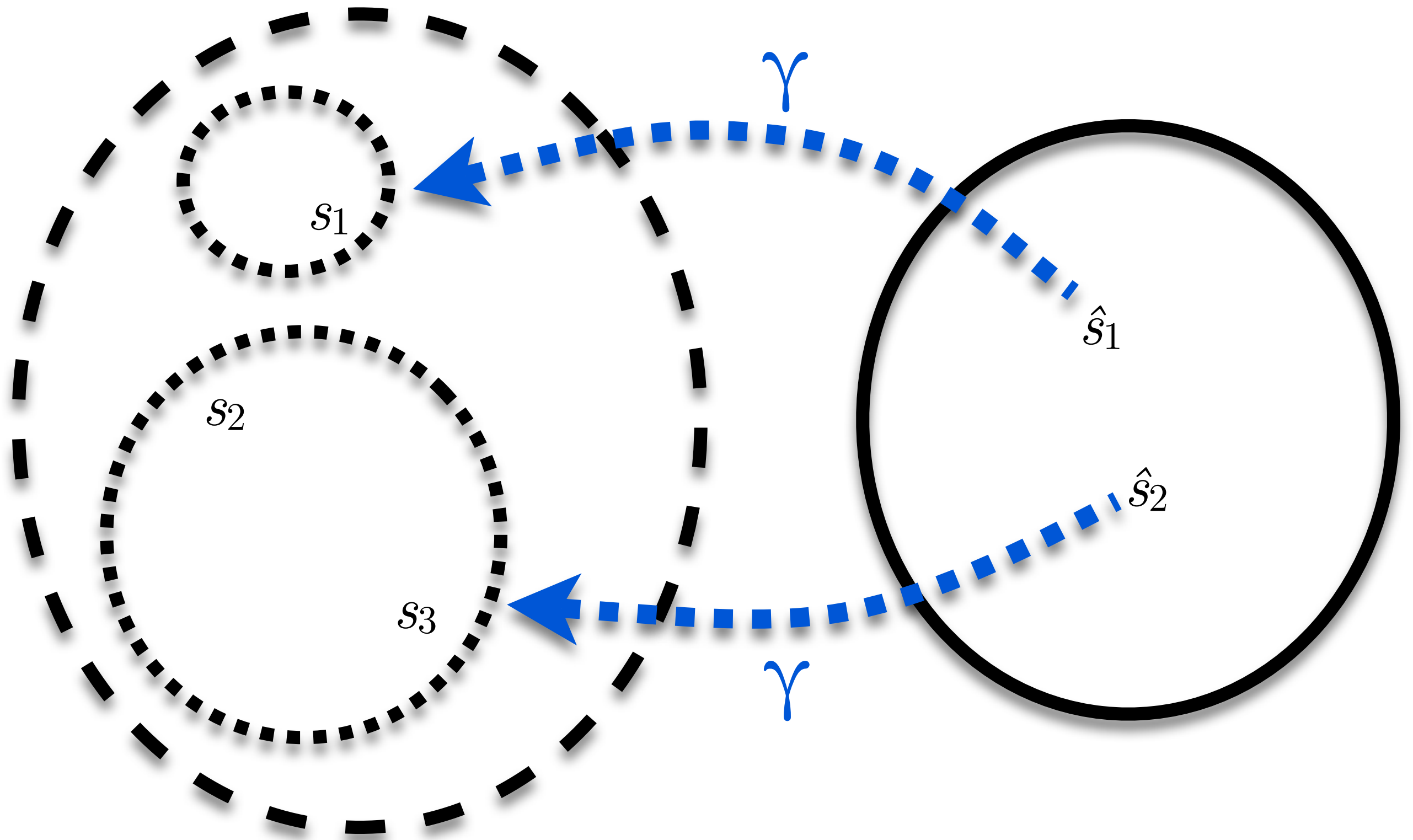
Abstraction



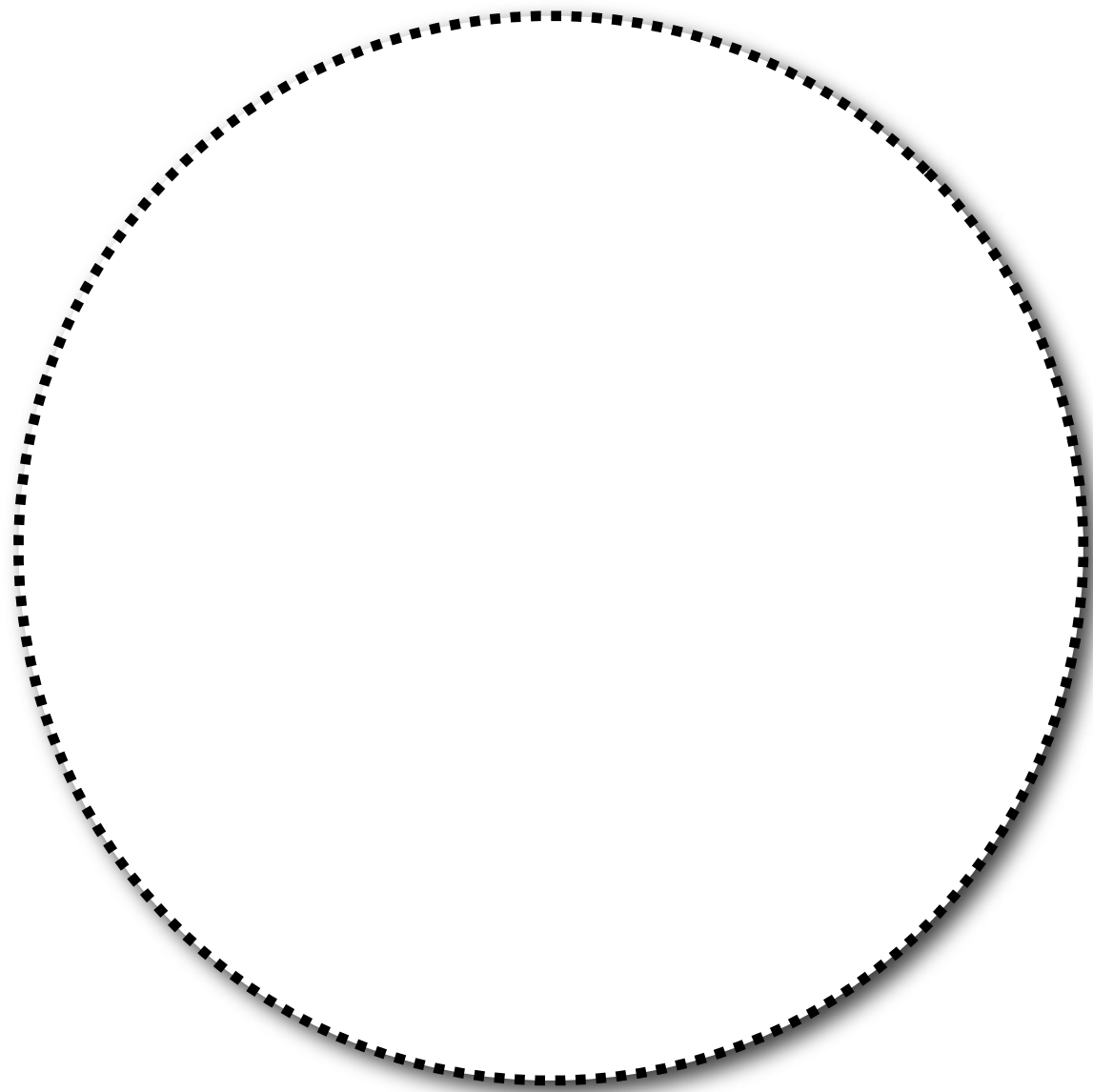
Concretization



Concretization

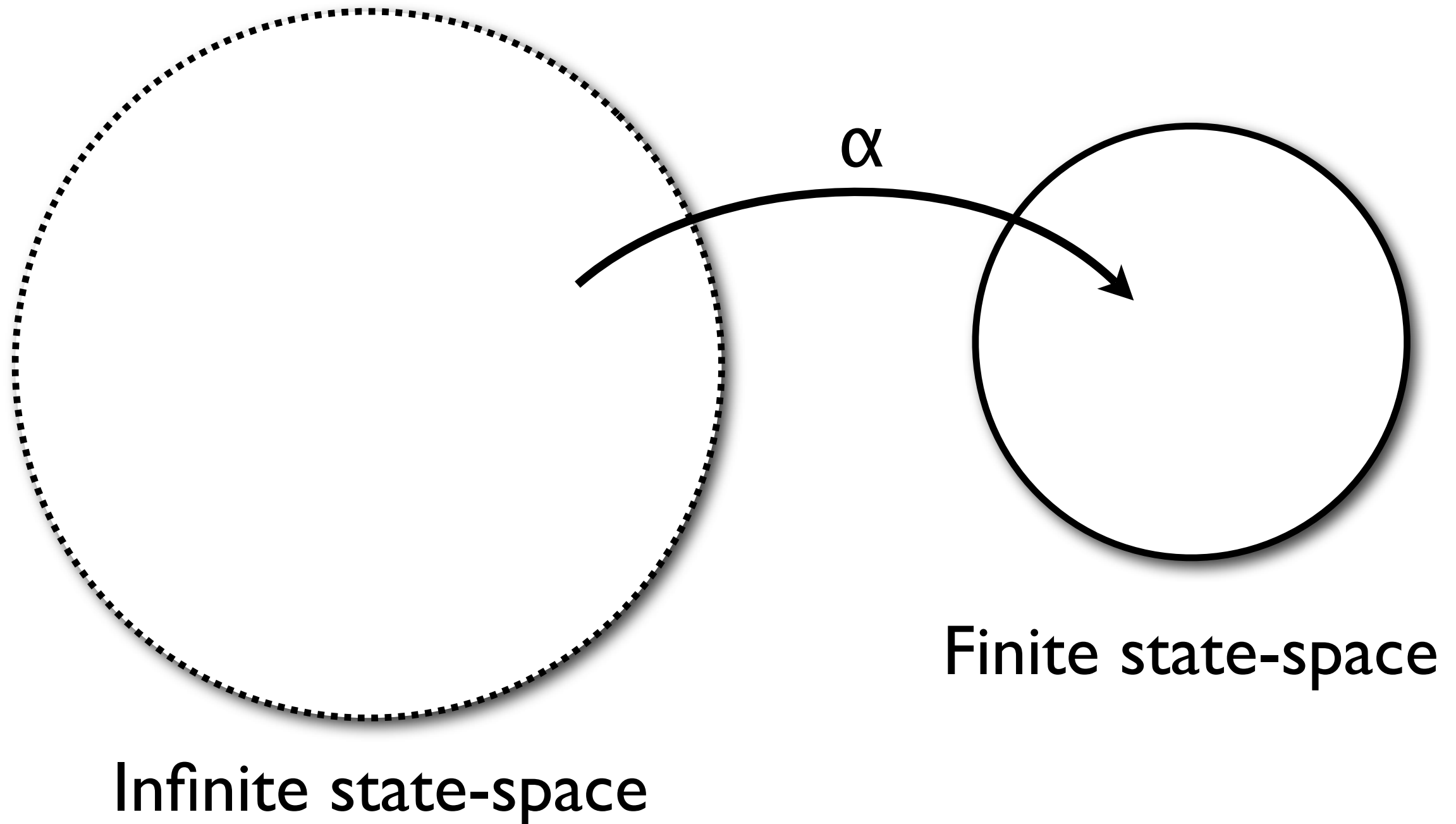


Small-step CFAs...



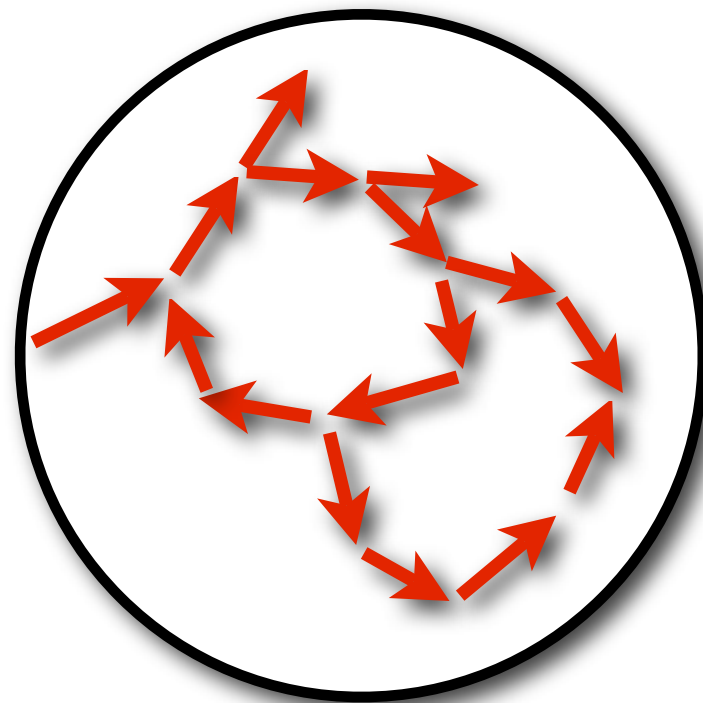
Infinite state-space

Small-step CFAs...



...are finite state machines.

...are finite state machines.

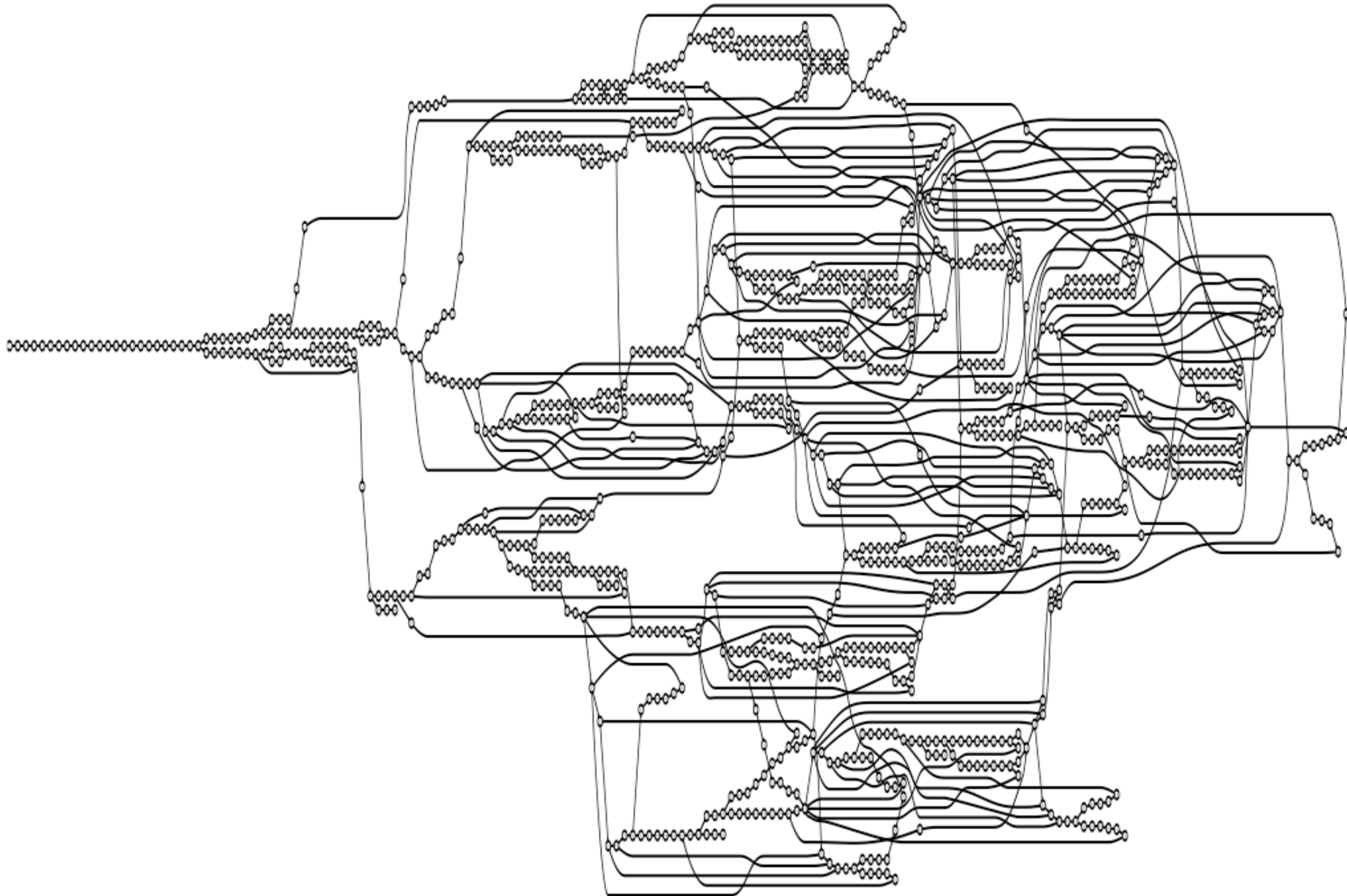


Finite state-space

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 OCFA for CPS

Why use CPS?

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

Continuation-passing style

$$v \in \text{Var}$$

$$f, e \in \text{Exp} = \text{Var} + \text{Lam}$$

$$\textit{lam} \in \text{Lam} ::= (\lambda (v_1 \dots v_n) \textit{call})$$

$$\textit{call} \in \text{Call} ::= (f \ e_1 \dots e_n)$$

Concrete state-space I

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

$$\rho \in \text{Env} = \text{Var} \rightarrow \text{Clo}$$

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

Concrete injector

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

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

Concrete semantics I

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

$$\mathcal{E} : \text{Exp} \times \textit{Env} \longrightarrow \textit{Clo}$$

$$\mathcal{E} : \text{Exp} \times \text{Env} \rightarrow \text{Clo}$$

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

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

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

$$(\llbracket (f \ e_1 \dots e_n) \rrbracket, \rho) \Rightarrow (call, \rho''), \text{ where}$$

$$(\llbracket (\lambda \ (v_1 \dots v_n) \ call) \rrbracket, \rho') = \mathcal{E}(f, \rho)$$

$$clo_i = \mathcal{E}(e_i, \rho)$$

$$\rho'' = \rho'[v_i \mapsto clo_i]$$

Abstracting to 0CFA

- Abstract closures into lambdas
- Merge all environments together

Abstract state-space I

$$\hat{\zeta} \in \hat{\Sigma} = \text{Call} \times \widehat{Env}$$

$$\hat{\rho} \in \widehat{Env} = \text{Var} \rightarrow \mathcal{P}(\widehat{Clo})$$

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

Abstraction I

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

Abstraction I

$$\alpha(\textit{call}, \rho) = (\textit{call}, \alpha(\rho))$$

Abstraction I

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

Abstract injector

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

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

Abstract semantics I

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

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

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

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

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

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

$$(\llbracket (f \ e_1 \dots e_n) \rrbracket, \hat{\rho}) \rightsquigarrow (call, \hat{\rho}'), \text{ where}$$

$$\llbracket (\lambda \ (v_1 \dots v_n) \ call) \rrbracket \in \hat{\mathcal{E}}(f, \hat{\rho})$$

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

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

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

$(\llbracket (f \ e_1 \dots e_n) \rrbracket, \hat{\rho}) \rightsquigarrow (call, \hat{\rho}')$, where

$$\llbracket (\lambda \ (v_1 \dots v_n) \ call) \rrbracket \in \hat{\mathcal{E}}(f, \hat{\rho})$$

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

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

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

$$(\llbracket (f \ e_1 \dots e_n) \rrbracket, \hat{\rho}) \rightsquigarrow (call, \hat{\rho}'), \text{ where}$$

$$\llbracket (\lambda \ (v_1 \dots v_n) \ call) \rrbracket \in \hat{\mathcal{E}}(f, \hat{\rho})$$

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

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

$$(\llbracket (f \ e_1 \dots e_n) \rrbracket, \rho) \Rightarrow (call, \rho''), \text{ where}$$

$$(\llbracket (\lambda \ (v_1 \dots v_n) \ call) \rrbracket, \rho') = \mathcal{E}(f, \rho)$$

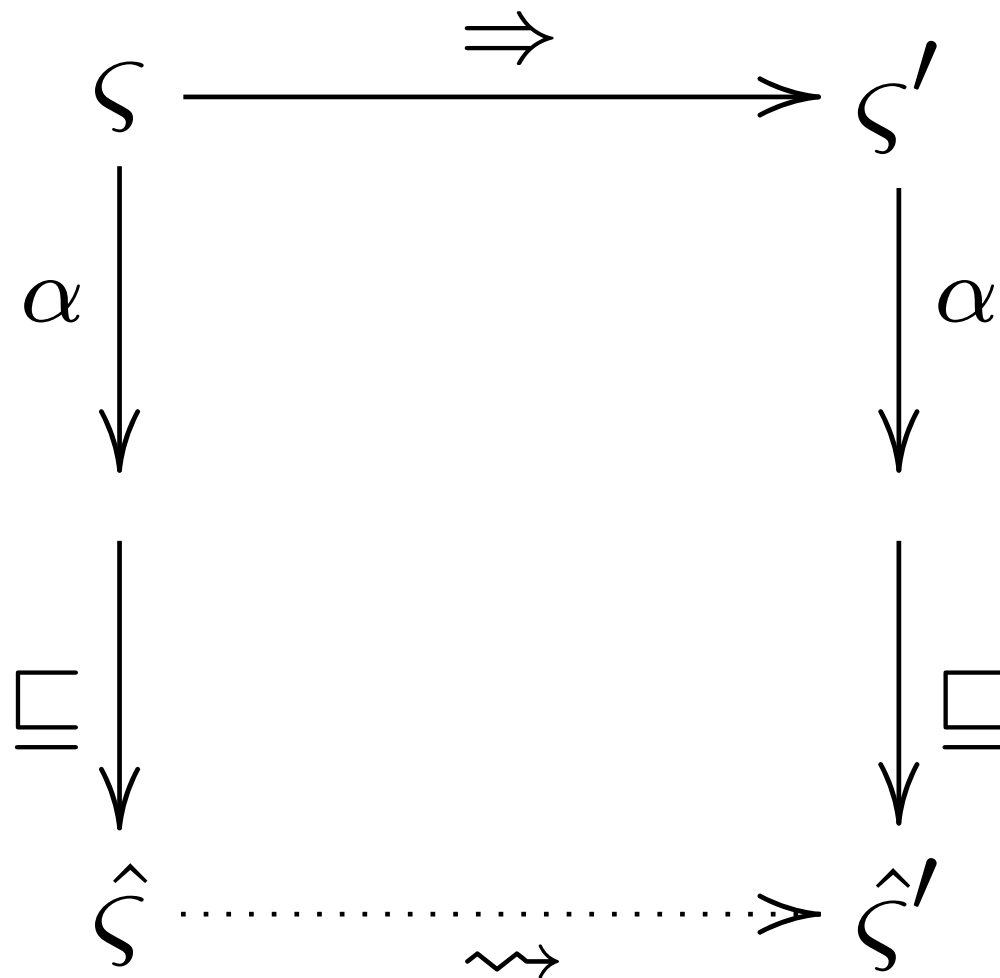
$$clo_i = \mathcal{E}(e_i, \rho)$$

$$\rho'' = \rho'[v_i \mapsto clo_i]$$

Join operation

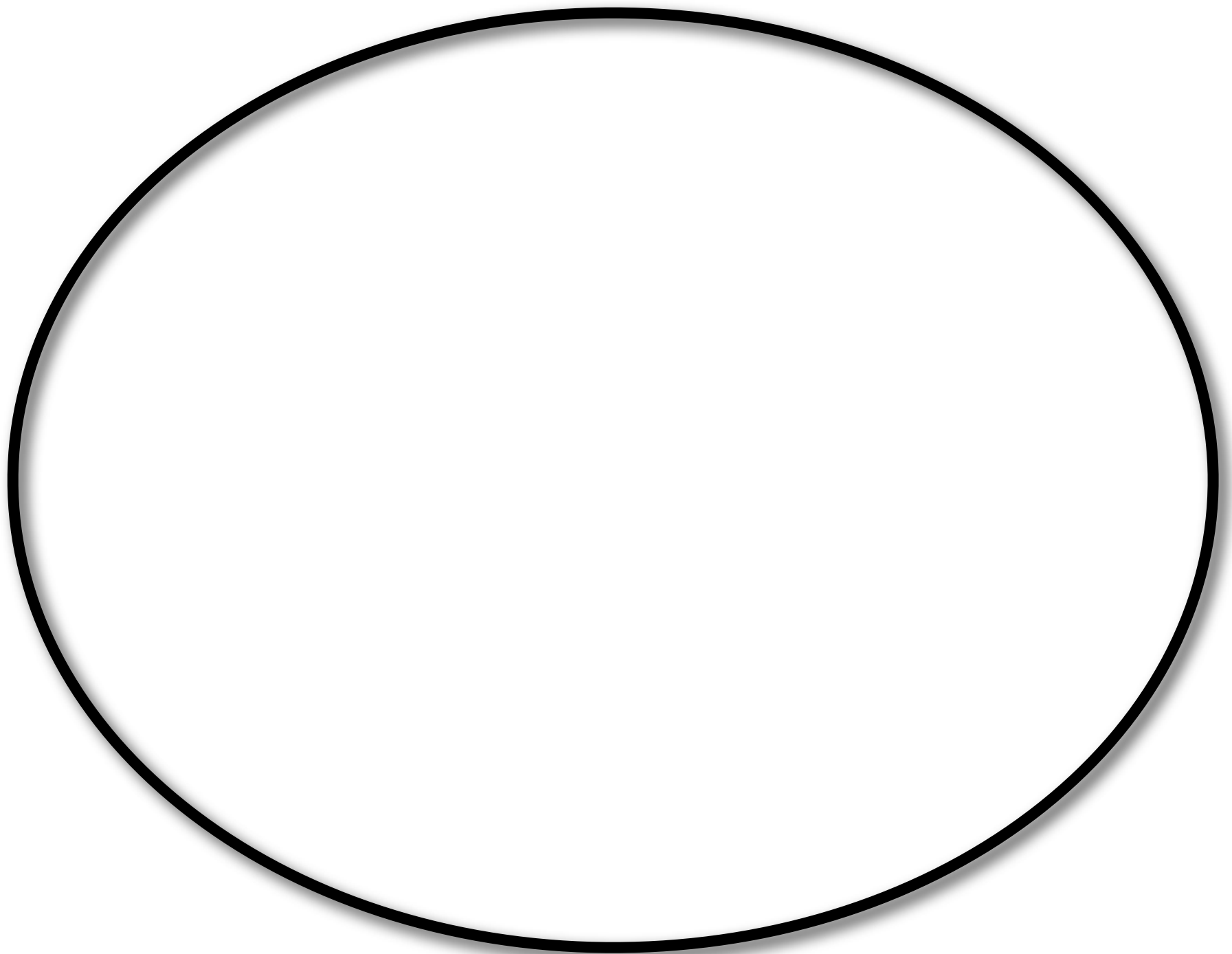
$$(\hat{\rho} \sqcup \hat{\rho}') = \lambda v. (\hat{\rho}(v) \cup \hat{\rho}'(v))$$

Soundness



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

Running 0CFA

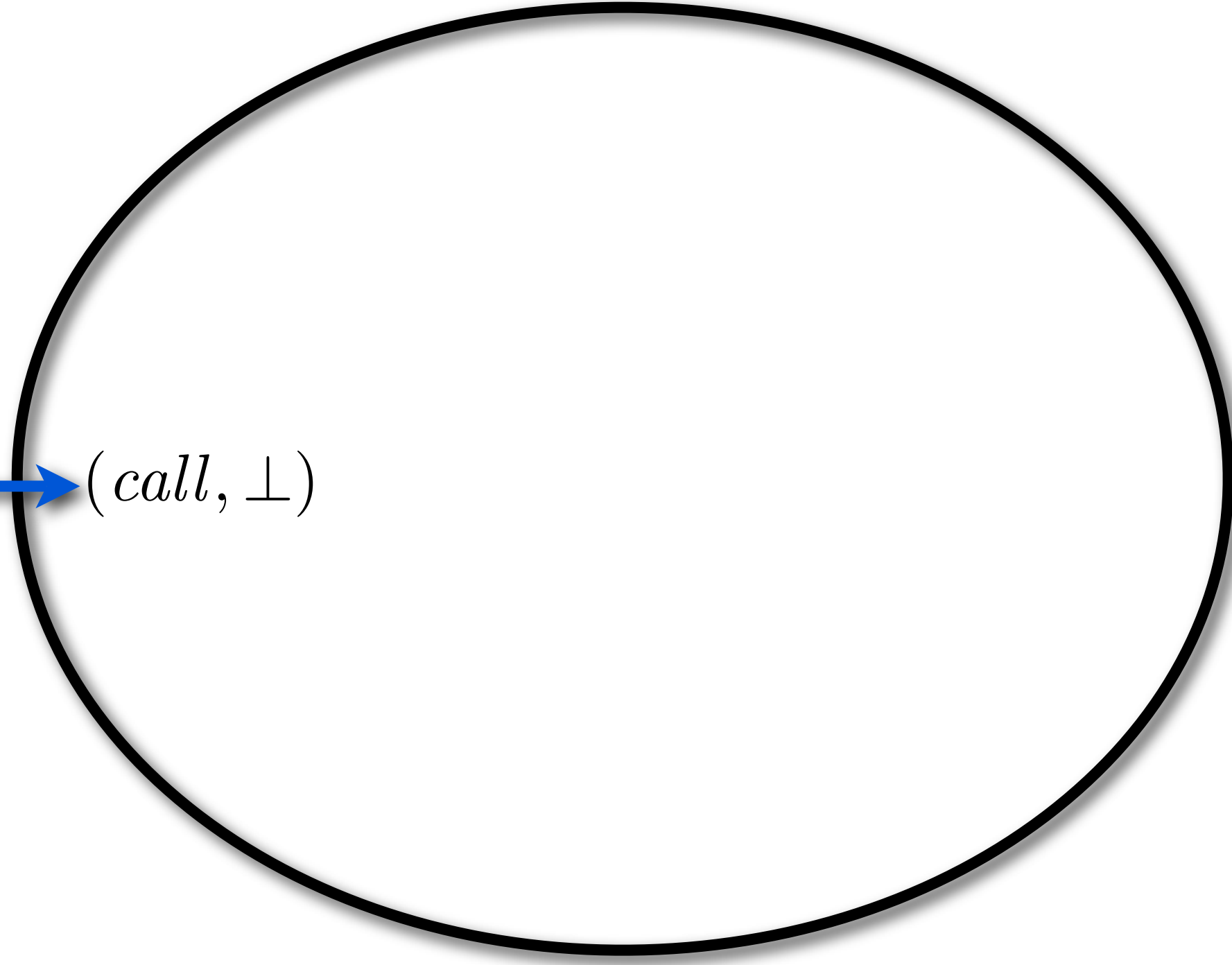


Running 0CFA

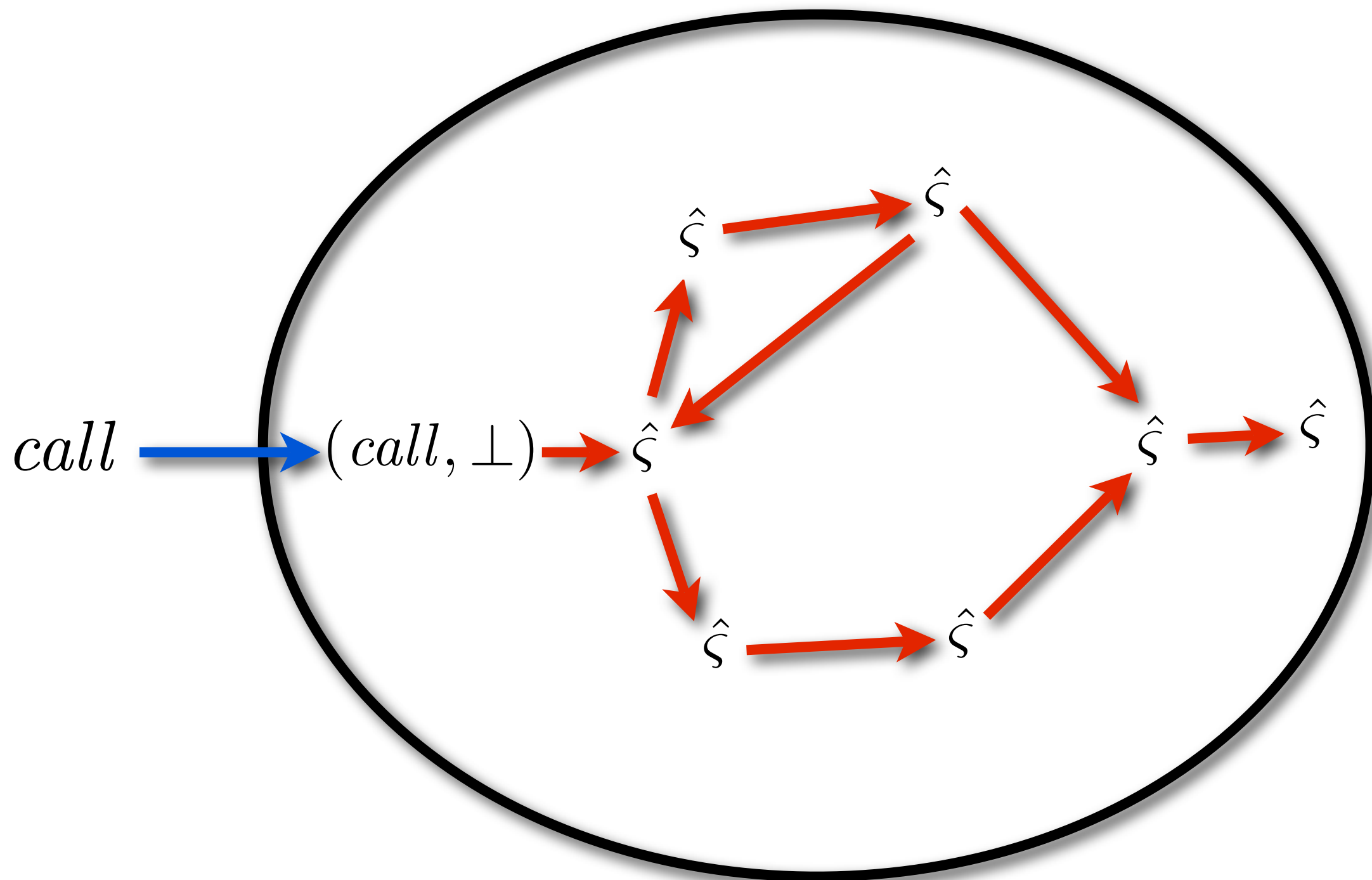
$$\hat{\Sigma}$$

Running 0CFA

call → $(call, \perp)$

A large, empty oval with a thick black border is centered on the slide. A blue arrow points from the left towards the left edge of the oval. The arrow starts at the text 'call' and ends at the text '(call, ⊥)' which is positioned just inside the oval's boundary.

Running 0CFA



Computing flow sets

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

Example: Omega

$lam = (\lambda \ (f) \ (f \ f))$

Example: Omega

$lam = (\lambda (f) (f f))$

$call = (lam lam)$

Example: Omega

$$lam = (\lambda \ (f) \ (f \ f))$$

$$call = (lam \ lam)$$

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

Example: Omega

$$lam = (\lambda \ (f) \ (f \ f))$$

$$call = (lam \ lam)$$

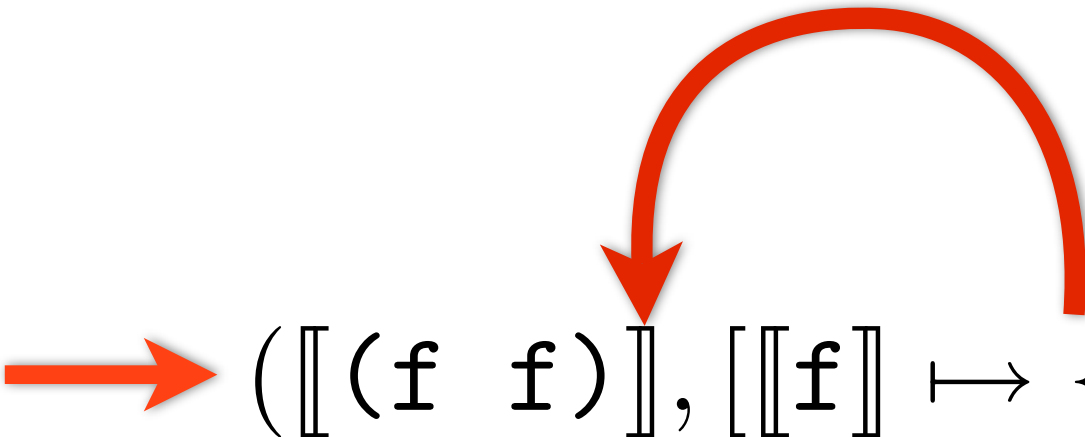
$$\hat{\mathcal{I}}(call) = (call, \perp) \longrightarrow ([[f \ f]], [[f]] \mapsto \{lam\})$$

Example: Omega

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

$call = (lam \ lam)$

$\hat{\mathcal{I}}(call) = (call, \perp) \longrightarrow ([\text{f f}], [[\text{f}] \mapsto \{lam\}])$



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

CPS with a let form

$$v \in \text{Var}$$
$$f, e \in \text{Exp} = \text{Var} + \text{Lam}$$
$$lam \in \mathbf{Lam} ::= (\lambda \ (v_1 \dots v_n) \ call)$$
$$\begin{array}{l} call \in \mathbf{Call} ::= (f \ e_1 \ \dots \ e_n) \\ \quad \quad \quad | \quad (\mathbf{let} \ ((v \ e)) \ call) \end{array}$$

Let-transition rule

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

Let-transition rule

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

Example

$call_1 = (\text{let } ((\text{id } (\lambda (x \ q) \ (q \ x))))$

$call_2 = \quad (\text{id } 3 \ (\lambda \ (v1)$

$call_3 = \quad (\text{id } 4 \ (\lambda \ (v2)$

$call_4 = \quad (\text{halt } v2))))$

Example

$call_1 = (\text{let } ((\text{id } (\lambda (x \ q) \ (q \ x))))$

$call_2 = \quad (\text{id } 3 \ (\lambda \ (v1)$

$call_3 = \quad \quad (\text{id } 4 \ (\lambda \ (v2)$

$call_4 = \quad \quad \quad (\text{halt } v2))))$

Example

$call_1 = (\text{let } ((\text{id } (\lambda (x \ q) \ (q \ x))))$

$call_2 = \quad (\text{id } 3 \ (\lambda \ (v1)$

$call_3 = \quad \quad (\text{id } 4 \ (\lambda \ (v2)$

$call_4 = \quad \quad \quad (\text{halt } v2))))$

$(call_1, \perp)$

Example

$call_1 = (\text{let } ((\text{id } (\lambda (x \ q) \ (q \ x))))$

$call_2 = \quad (\text{id } 3 \ (\lambda \ (v1)$

$call_3 = \quad \quad (\text{id } 4 \ (\lambda \ (v2)$

$call_4 = \quad \quad \quad (\text{halt } v2))))$

$(call_1, \perp)$ 
 $(call_2, [\text{id} \mapsto \{\lambda_1\}])$

Example

$call_1 = (\text{let } ((\text{id } (\lambda (x \ q) \ (q \ x))))$

$call_2 = \quad (\text{id } 3 \ (\lambda (v1)$

$call_3 = \quad (\text{id } 4 \ (\lambda (v2)$

$call_4 = \quad (\text{halt } v2))))$

$(call_1, \perp)$



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



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

Example

$call_1 = (\text{let } ((\text{id } (\lambda (x \ q) \ (q \ x))))$

$call_2 = \quad (\text{id } 3 \ (\lambda \ (v1)$

$call_3 = \quad \quad (\text{id } 4 \ (\lambda \ (v2)$

$call_4 = \quad \quad \quad (\text{halt } v2))))$

$(call_1, \perp)$

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

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

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

Example

$call_1 = (\text{let } ((\text{id } (\lambda (x \ q) \ (q \ x))))$

$call_2 = \quad (\text{id } 3 \ (\lambda (v1)$

$call_3 = \quad (\text{id } 4 \ (\lambda (v2)$

$call_4 = \quad (\text{halt } v2))))$

$(call_1, \perp)$

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

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

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

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

Example

$call_1 = (\text{let } ((\text{id } (\lambda (x \ q) \ (q \ x))))$

$call_2 = \quad (\text{id } 3 \ (\lambda (v1)$

$call_3 = \quad \quad (\text{id } 4 \ (\lambda (v2)$

$call_4 = \quad \quad \quad (\text{halt } v2))))$

$(call_1, \perp)$

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

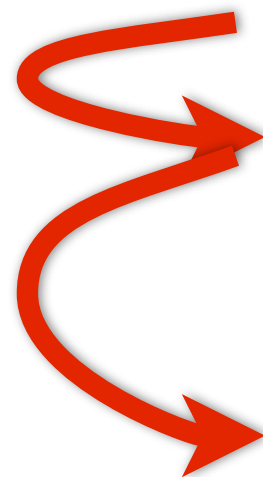
$((q \ x), [\dots, q \mapsto \{\lambda_2\}, x \mapsto \{3\}])$

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

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

$((\text{halt } v2), [\dots, v2 \mapsto \{3, 4\}])$

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



Example

$call_1 = (\text{let } ((\text{id } (\lambda (x \ q) \ (q \ x))))$

$call_2 = \quad (\text{id } 3 \ (\lambda (v1)$

$call_3 = \quad \quad (\text{id } 4 \ (\lambda (v2)$

$call_4 = \quad \quad \quad (\text{halt } v2))))$

$(call_1, \perp)$

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

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

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

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

$((\text{halt } v2), [\dots, v2 \mapsto \{3, 4\}])$

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

$((q \ x), [\dots])$

Example

$call_1 = (\text{let } ((\text{id } (\lambda (x \ q) \ (q \ x))))$

$call_2 = \quad (\text{id } 3 \ (\lambda (v1)$

$call_3 = \quad \quad (\text{id } 4 \ (\lambda (v2)$

$call_4 = \quad \quad \quad (\text{halt } v2))))$

$(call_1, \perp)$

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

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

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

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

$((\text{halt } v2), [\dots, v2 \mapsto \{3, 4\}])$

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

$((q \ x), [\dots])$

$((\text{halt } v2), [\dots])$

Results

$$\text{FlowsTo}[\mathbf{id}] = \{\lambda_1\}$$

$$\text{FlowsTo}[\mathbf{q}] = \{\lambda_2, \lambda_3\}$$

$$\text{FlowsTo}[\mathbf{x}] = \{3, 4\}$$

$$\text{FlowsTo}[\mathbf{v1}] = \{3, 4\}$$

$$\text{FlowsTo}[\mathbf{v2}] = \{3, 4\}$$

Questions?