Abstract interpreters for free

Matthew Might
University of Utah
matt.might.net
@mattmight

“I replaced myself with a shell script.”

Hilary Mason

My life goal: Replace myself with a $\LaTeX$ macro.

Big
Idea

small-step concrete semantics
=>
small-step abstract semantics

small-step concrete semantics

=>

small-step abstract semantics

(for free)

How do you design an abstract interpreter?

More science; less art?

Yes.

A tale of two machines

$$([\![var := var']\!] : \boldsymbol{stmt}, env, heap) \Rightarrow (\boldsymbol{stmt}, env[var \mapsto env(var')], heap).$$

$$([\![var := var']\!] : \boldsymbol{stmt}, \widehat{env}, \widehat{heap}) \leadsto (\boldsymbol{stmt}, \widehat{env}[var \mapsto \widehat{env}(var')], \widehat{heap}).$$

$$(\llbracket var := var' \rrbracket : \boldsymbol{stmt}, \widehat{env}, \widehat{heap}) \mathrel{\ooalign{$\Rightarrow$\cr$\rightsquigarrow$}} (\boldsymbol{stmt}, \widehat{env}[var \mapsto \widehat{env}(var')], \widehat{heap}).$$

The principle?

Put hats on everything.

Problem: It doesn’t work.

$$([\![*var := var']\!] : \boldsymbol{stmt}, env, heap) \Rightarrow (\boldsymbol{stmt}, env, heap[env(var) \mapsto env(var')]),$$

$$\frac{\hat{a} \in \widehat{env}(var)}{([\![*var := var']\!] : \boldsymbol{stmt}, \widehat{env}, \widehat{heap}) \leadsto (\boldsymbol{stmt}, \widehat{env}, \widehat{heap} \sqcup [\hat{a} \mapsto \widehat{env}(var')]).}$$

$$\frac{\hat{a} \in \widehat{env}(var)}{\begin{gathered}\sout{(\llbracket *var := var' \rrbracket : \boldsymbol{stmt}, env, heap) \Rightarrow (\boldsymbol{stmt}, env, heap[env(var) \mapsto env(var')])},\\ (\llbracket *var := var' \rrbracket : \boldsymbol{stmt}, \widehat{env}, \widehat{heap}) \leadsto (\boldsymbol{stmt}, \widehat{env}, \widehat{heap} \sqcup [\hat{a} \mapsto \widehat{env}(var')]).\end{gathered}}$$

$$(\llbracket *var := var' \rrbracket : \boldsymbol{stmt}, env, heap) \Rightarrow (\boldsymbol{stmt}, env, heap[env(var) \mapsto env(var')]),$$

$$\frac{\hat{a} \in \widehat{env}(var)}{(\llbracket *var := var' \rrbracket : \boldsymbol{stmt}, \widehat{env}, \widehat{heap}) \leadsto (\boldsymbol{stmt}, \widehat{env}, \widehat{heap} \sqcup [\hat{a} \mapsto \widehat{env}(var')]).}$$

Where to add nondeterminism?

Where to add sets?

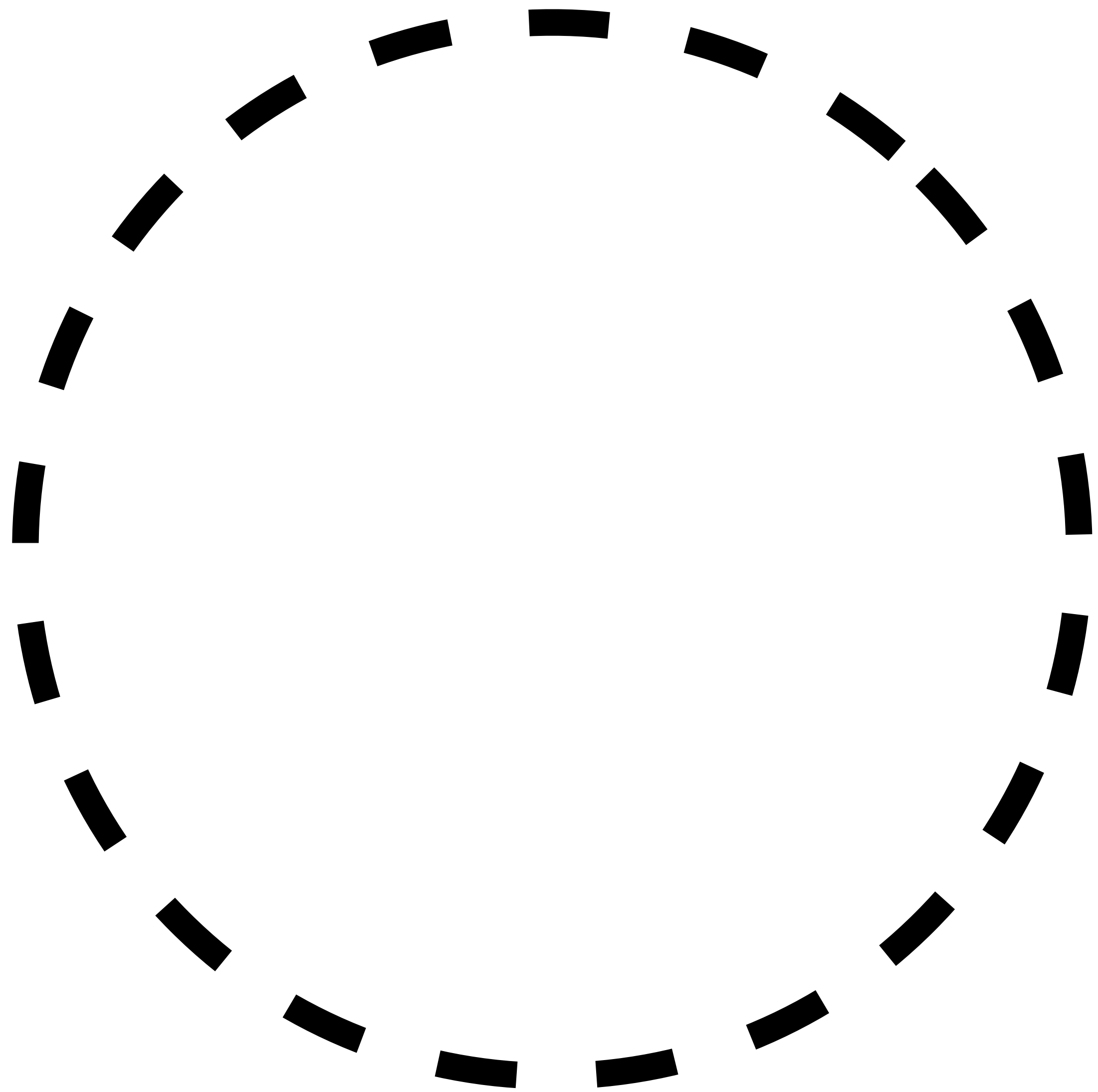
A two-step process.

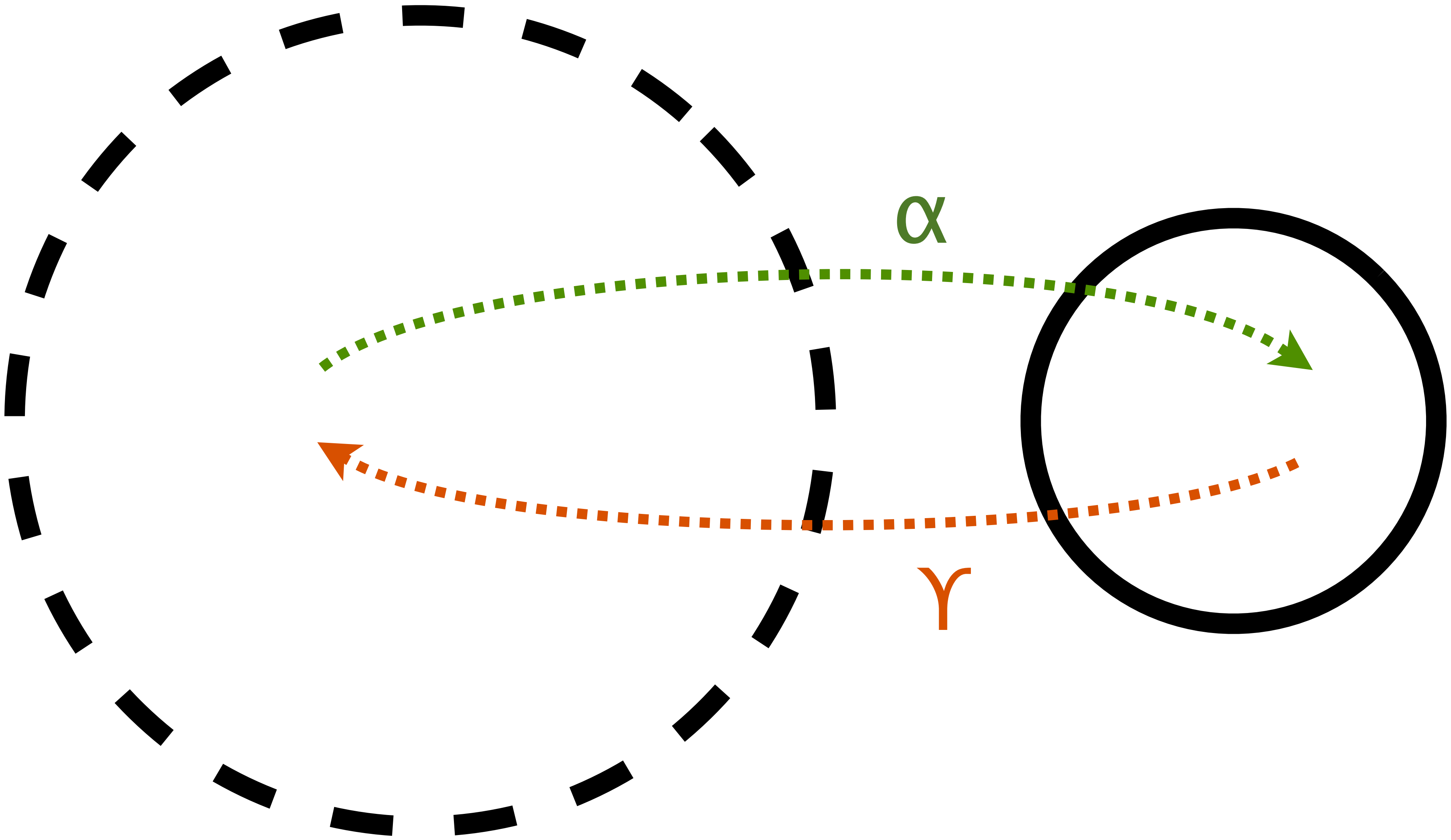
1. Snipping
2. Trickling

Snipping

Why doesn’t putting hats on everything work?

It doesn’t abstract.




α
γ

Where does infinite structure come from?

Recursive definitions.

Example: λ-calculus

$$\text{Closure} = \text{Lambda} \times \text{Environment}$$

$$\text{Environment} = \text{Var} \rightarrow \text{Closure}$$

$$\text{Closure} = \text{Lambda} \times \text{Environment}$$

$$\text{Environment} = \text{Var} \rightarrow \text{Closure}$$

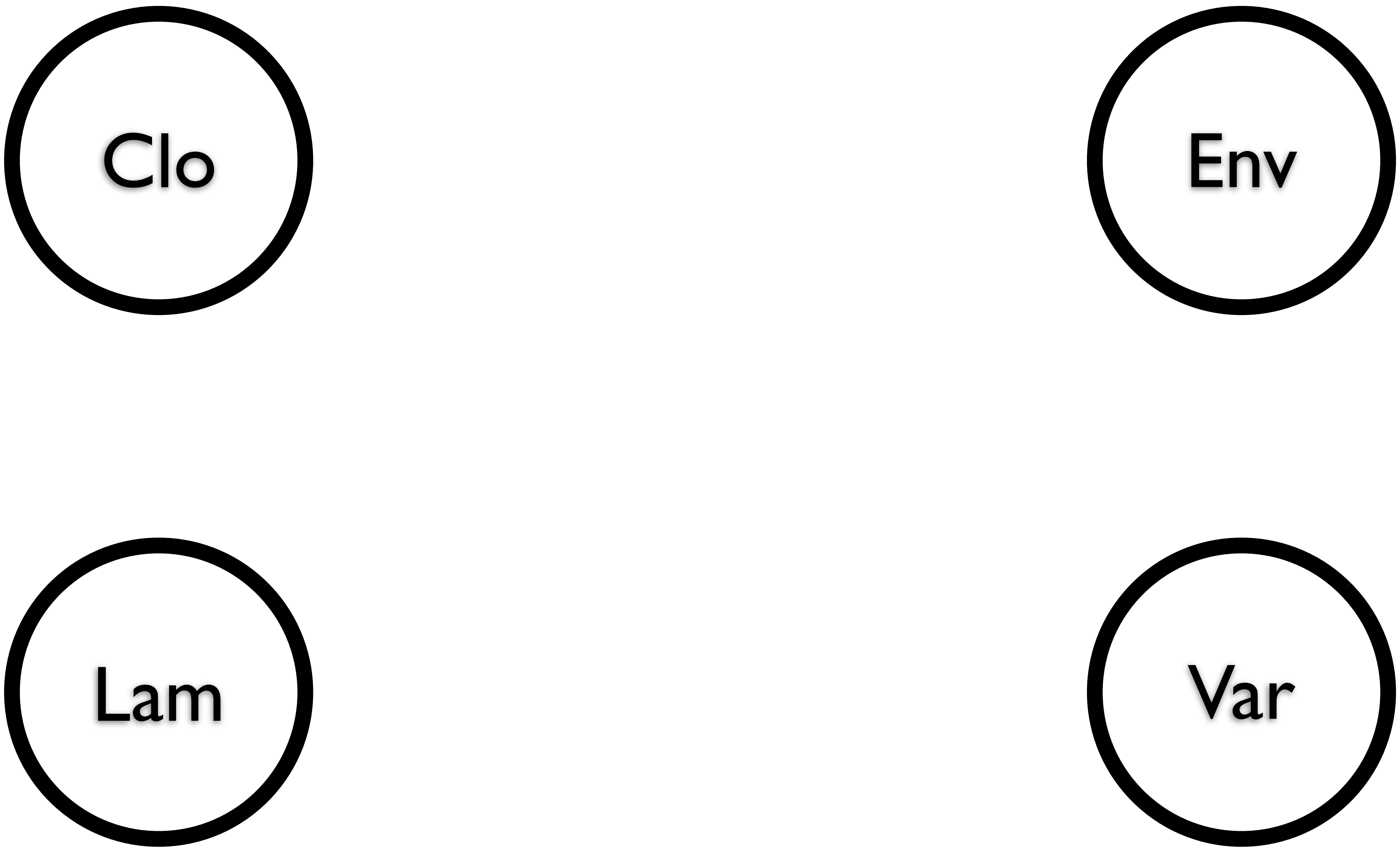
$$\text{Closure} = \text{Lambda} \times \text{Environment}$$

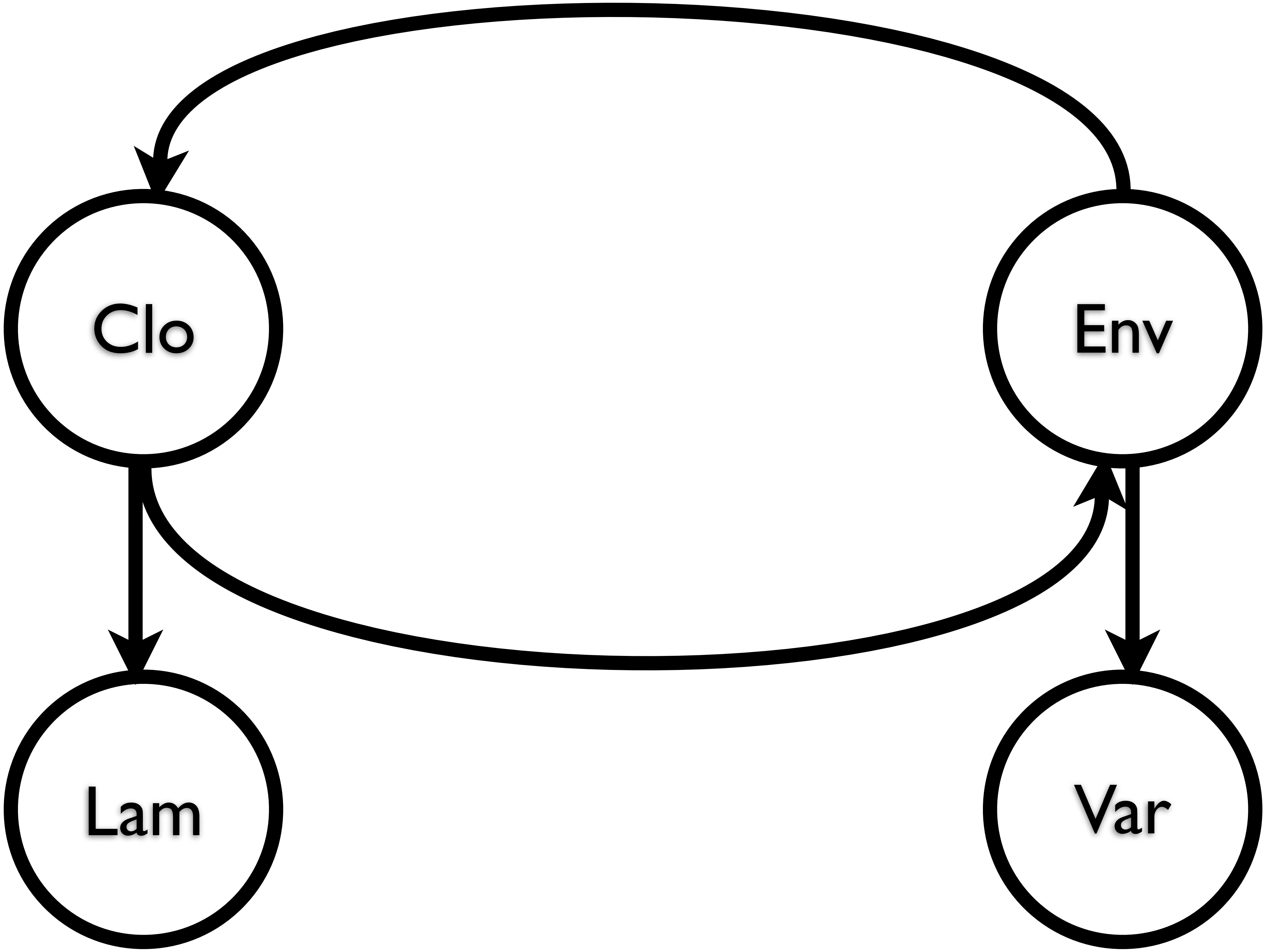
$$\text{Environment} = \text{Var} \rightarrow \text{Closure}$$

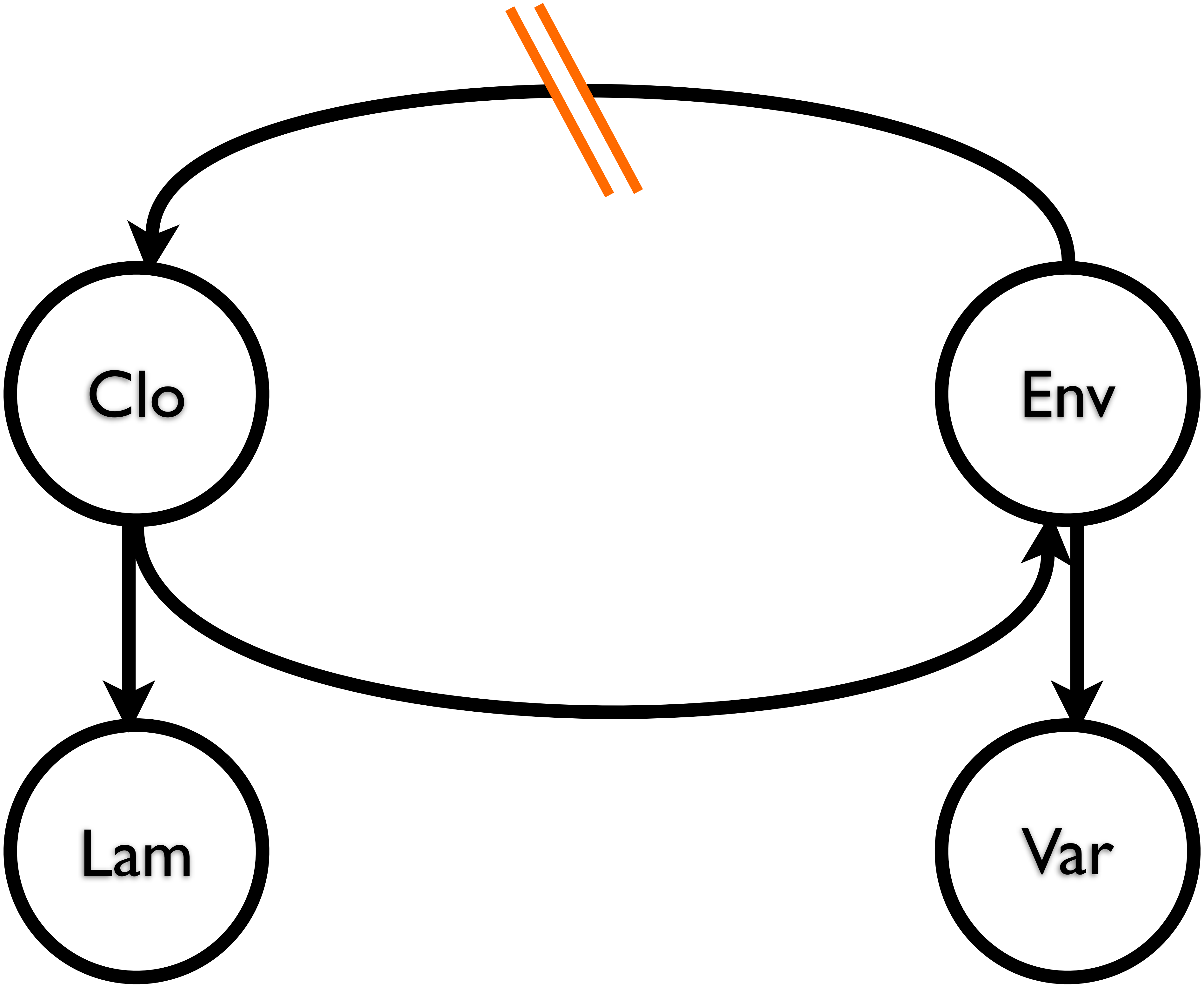
∞

How do we untie this knot?

Scott & Strachey, 1966


Clo
Env
Lam
Var


Clo
Env
Lam
Var


Clo
Env
Lam
Var

So, what happens to the semantics?

How do programmers handle recursive structures?

Pointers.

```
struct Clo {
 Lam   lam ;
 Env   env ;
} ;
```

```
struct Env {
 Var   var ;
 Clo   value ;
 Env*  env ;
} ;
```

```
struct Clo {
} ;
```

```
struct Env {
                   ;
  Env*  env ;
} ;
```

error: field ‘clo’ has incomplete type

```
struct Clo {
 Lam   lam ;
 Env   env ;
} ;
```

```
struct Env {
 Var   var ;
 Clo   value ;
 Env*  env ;
} ;
```

```
struct Clo {
 Lam   lam ;
 Env   env ;
} ;
```

```
struct Env {
 Var   var ;
 Clo*  value ;
 Env*  env ;
} ;
```

But, math lacks `malloc()`.

So, we add a store.

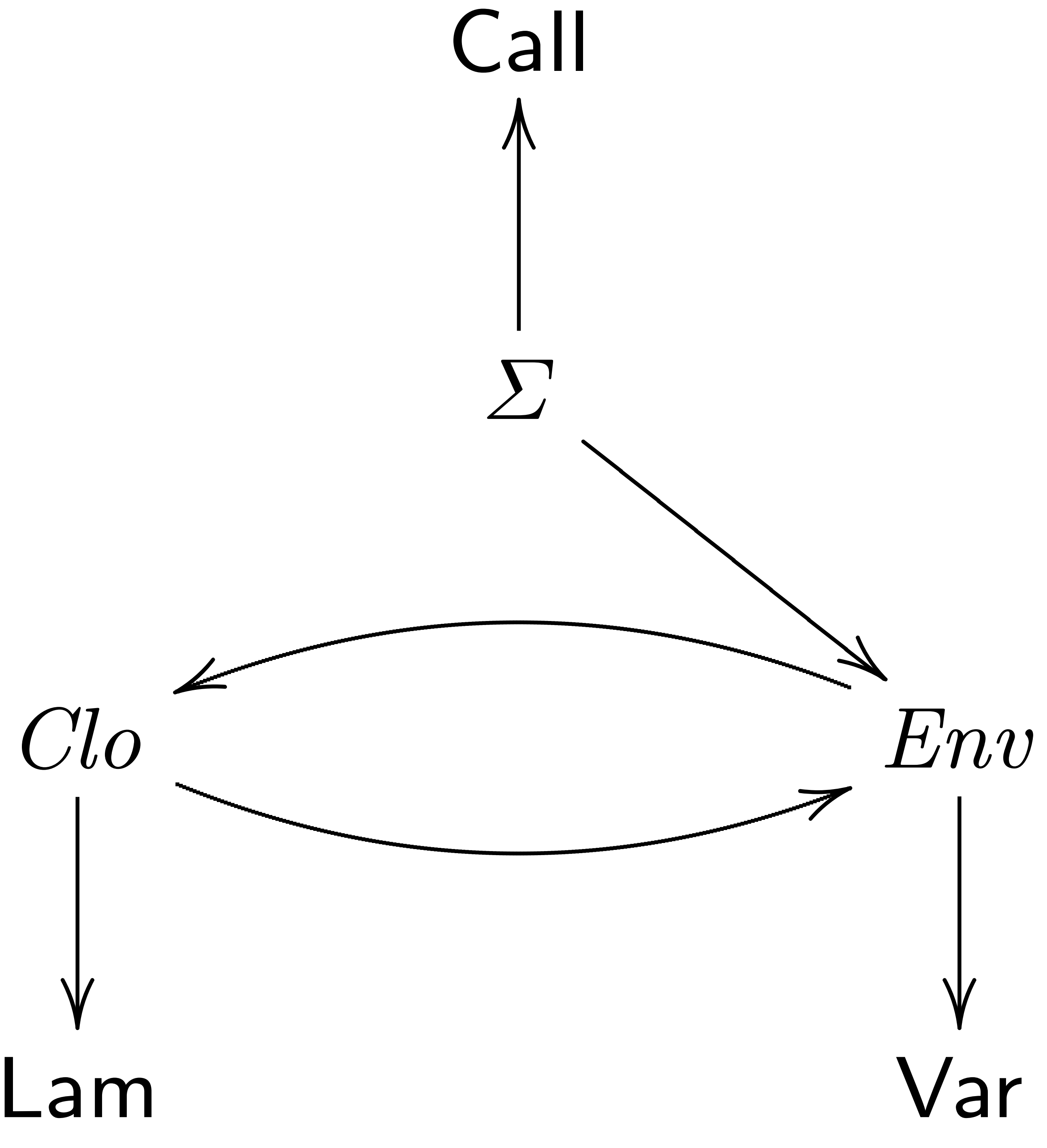
State-space (CPS λ-C)

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

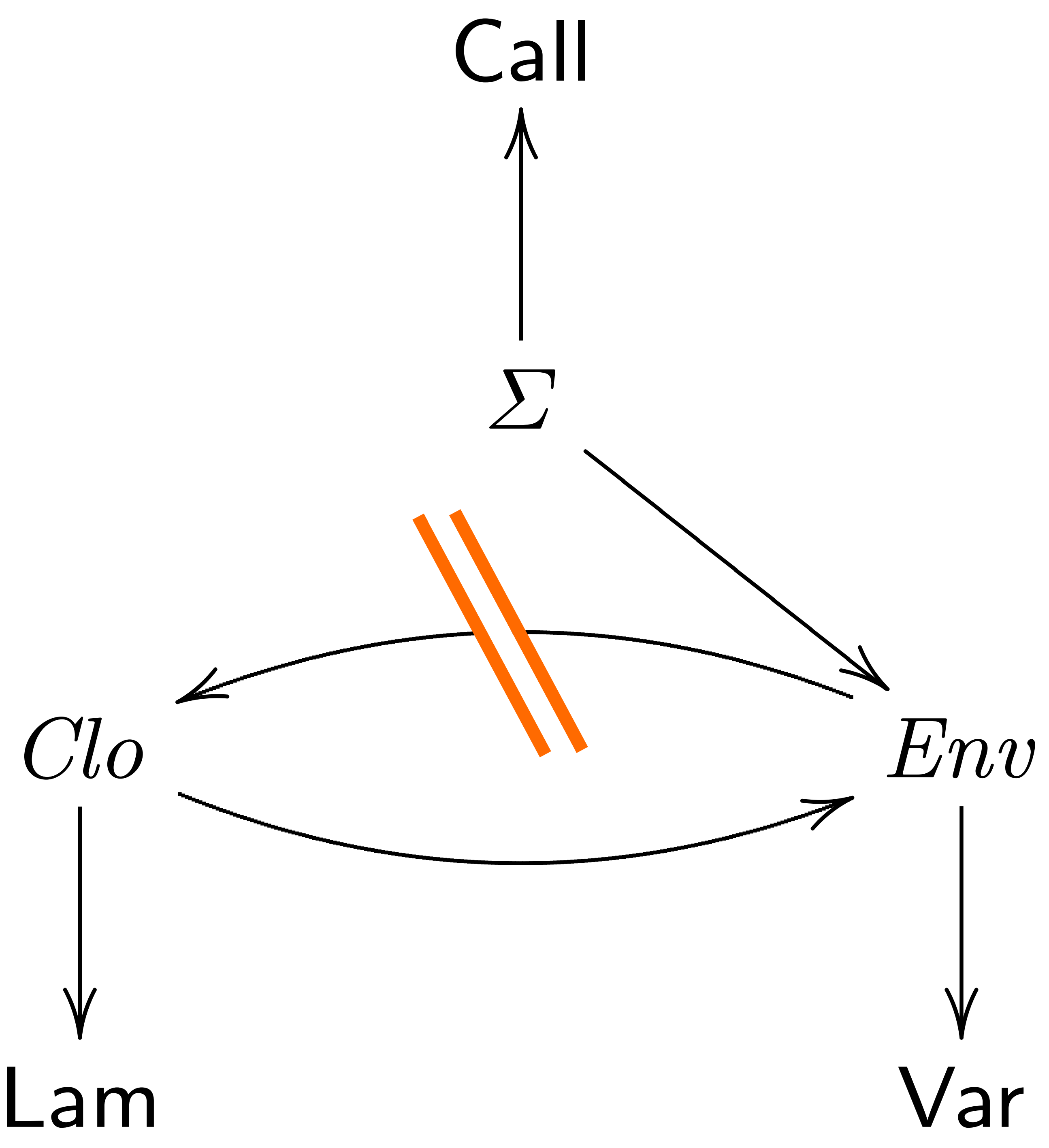
State-space (CPS λ-C)

$$\varsigma \in \Sigma = \mathsf{Call} \times \mathit{Env}$$
$$\rho \in \mathit{Env} = \mathsf{Var} \rightharpoonup \mathit{Clo}$$
$$\mathit{clo} \in \mathit{Clo} = \mathsf{Lam} \times \mathit{Env}.$$

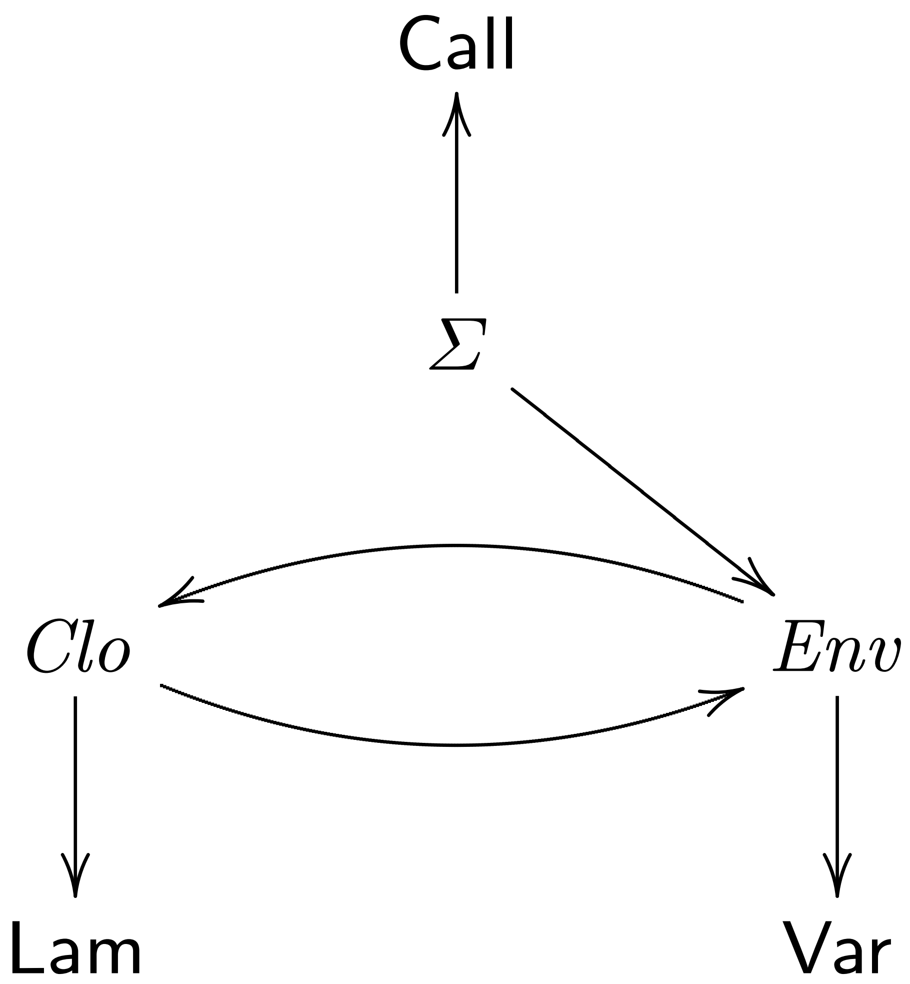
State-space (CPS λ-C)

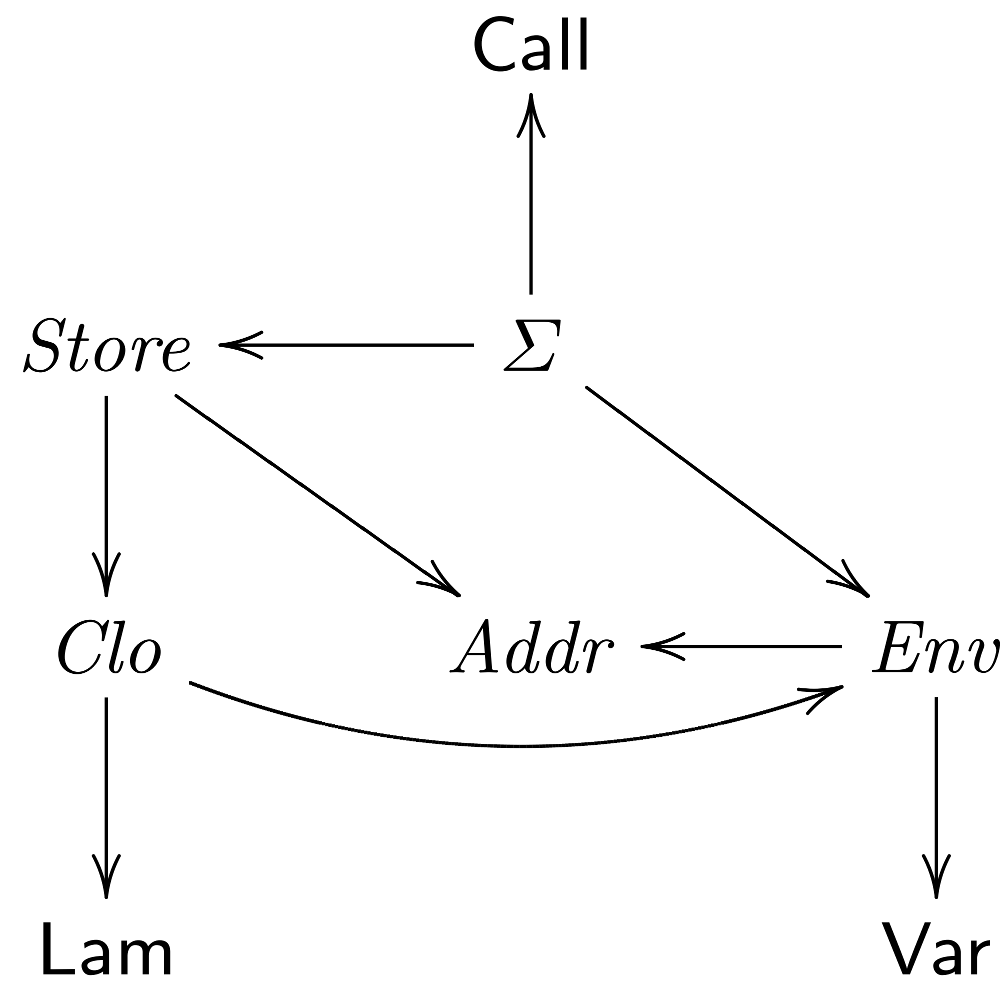


State-space (CPS λ-C)



State-space (CPS λ-C)





State-space (snipped)

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

State-space (snipped)

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

State-space (snipped)

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

$$\sigma \in Store = Addr \rightharpoonup Clo$$

State-space (snipped)

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

$$\rho \in \mathit{Env} = \mathsf{Var} \rightharpoonup \mathit{Addr}$$

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

$$\sigma \in \mathit{Store} = \mathit{Addr} \rightharpoonup \mathit{Clo}$$

State-space (snipped)

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

$$\rho \in \mathit{Env} = \mathsf{Var} \rightharpoonup \mathit{Addr}$$

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

$$\sigma \in \mathit{Store} = \mathit{Addr} \rightharpoonup \mathit{Clo}$$

$$a \in \mathit{Addr} \text{ is an infinite set of addresses}$$

How do transitions change?

Store-passing style.
(Scott & Strachey, 1966)

Before

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

After

$$
\begin{aligned}
(\llbracket(f\ e_1 \ldots e_n)\rrbracket, \rho, \sigma) &\Rightarrow (call, \rho'', \sigma''), \text{ where} \\
((lam, \rho'), \sigma'_0) &= \mathcal{E}((f, \rho), \sigma) \\
lam &= \llbracket(\lambda\ (v_1 \ldots v_n)\ call)\rrbracket \\
a_1, \ldots, a_n &\notin dom(\sigma'_0) \\
\rho'' &= \rho'[v_i \mapsto a_i] \\
(clo_i, \sigma'_i) &= \mathcal{E}((e_i, \rho), \sigma'_{i-1}) \\
\sigma'' &= \sigma'_n[a_i \mapsto clo_i],
\end{aligned}
$$

After (cleaned up)

$$
\begin{aligned}
(\llbracket(f\ e_1 \ldots e_n)\rrbracket, \rho, \sigma) &\Rightarrow (call, \rho'', \sigma'), \text{ where} \\
(lam, \rho') &= \mathcal{E}(f, \rho, \sigma) \\
lam &= \llbracket(\lambda\ (v_1 \ldots v_n)\ call)\rrbracket \\
a_1, \ldots, a_n &\notin dom(\sigma) \\
\rho'' &= \rho'[v_i \mapsto a_i] \\
clo_i &= \mathcal{E}(e_i, \rho, \sigma) \\
\sigma' &= \sigma[a_i \mapsto clo_i],
\end{aligned}
$$

But, the state-space
is still infinite.

So, how do we deal with addresses?

Cousot & Cousot, 1977

State-space (snipped)

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

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

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

$$\sigma \in Store = Addr \rightharpoonup Clo$$

$$a \in Addr \text{ is an infinite set of addresses}$$

State-space (snipped)

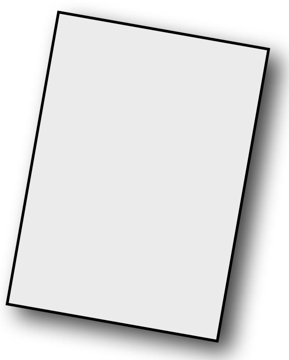
$$\varsigma \in \Sigma = \mathsf{Call} \times \mathit{Env} \times \mathit{Store}$$

$$\rho \in \mathit{Env} = \mathsf{Var} \rightharpoonup \mathit{Addr}$$

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

$$\sigma \in \mathit{Store} = \mathit{Addr} \rightharpoonup \mathit{Clo}$$

$a \in \mathit{Addr}$ is a[...]finite set of addresses

 : infinite set of addresses → finite set of addresses

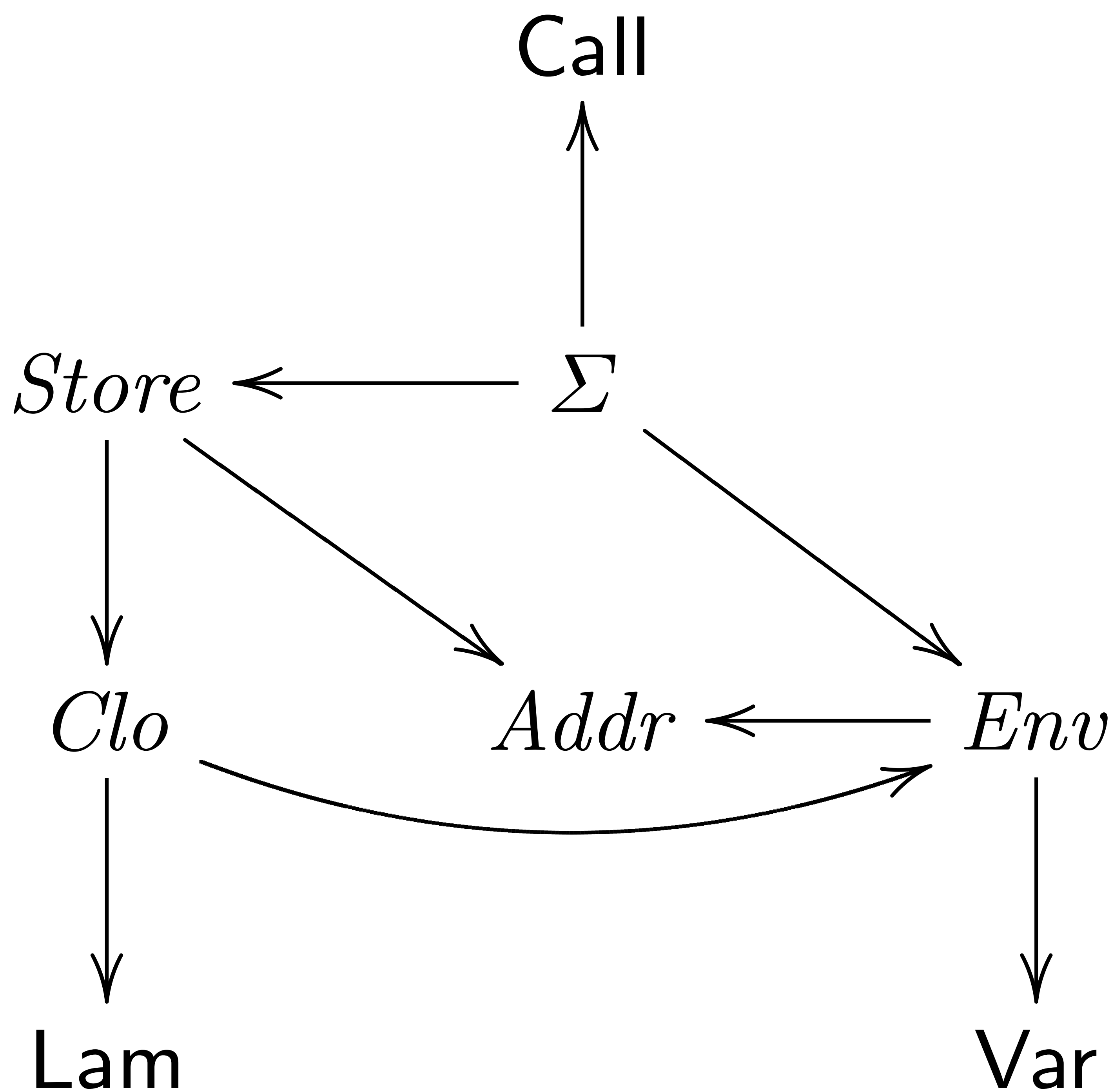
$$\eta : Addr \rightarrow \widehat{Addr}$$

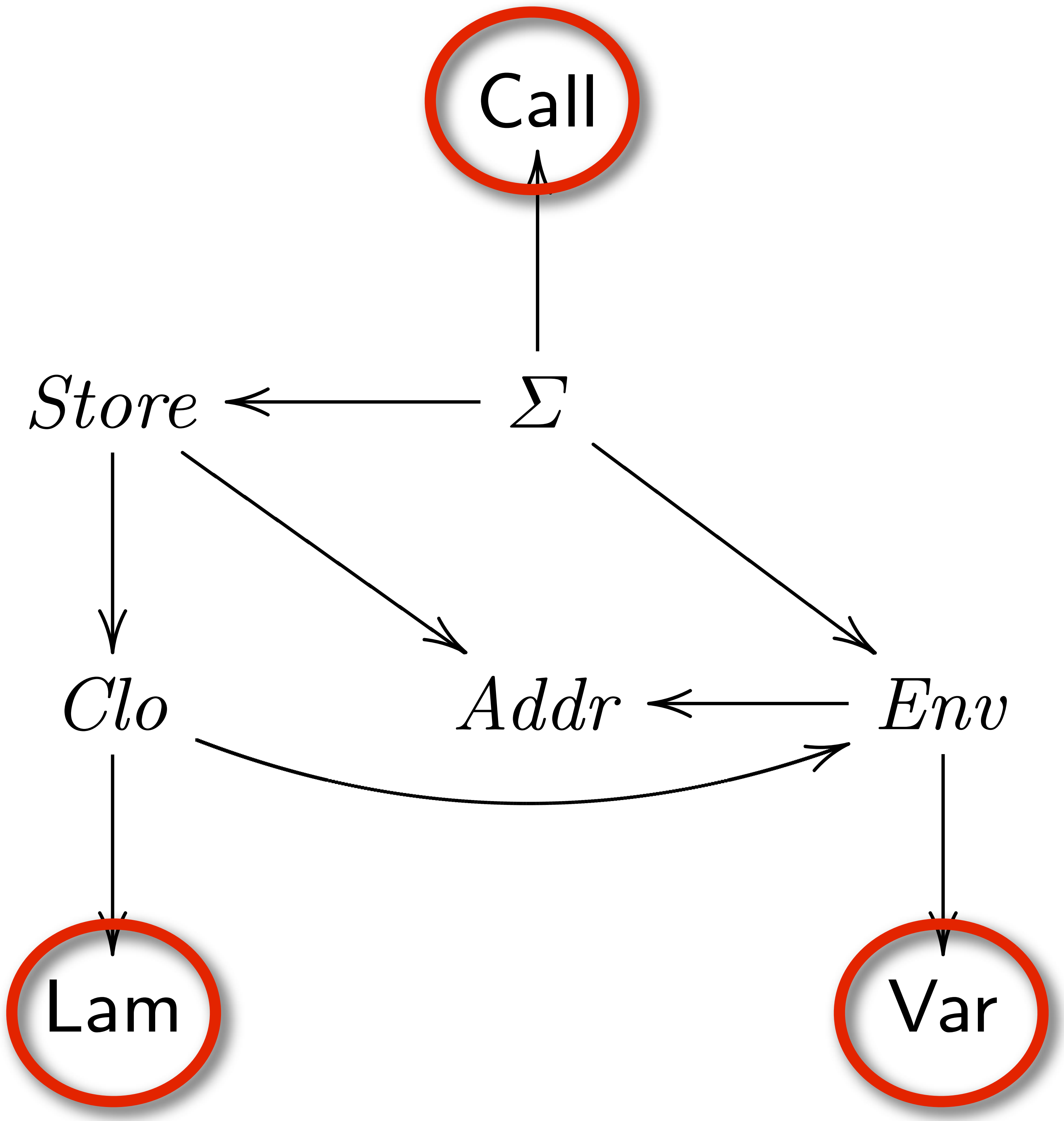
$$\eta : Addr \rightarrow \widehat{Addr}$$

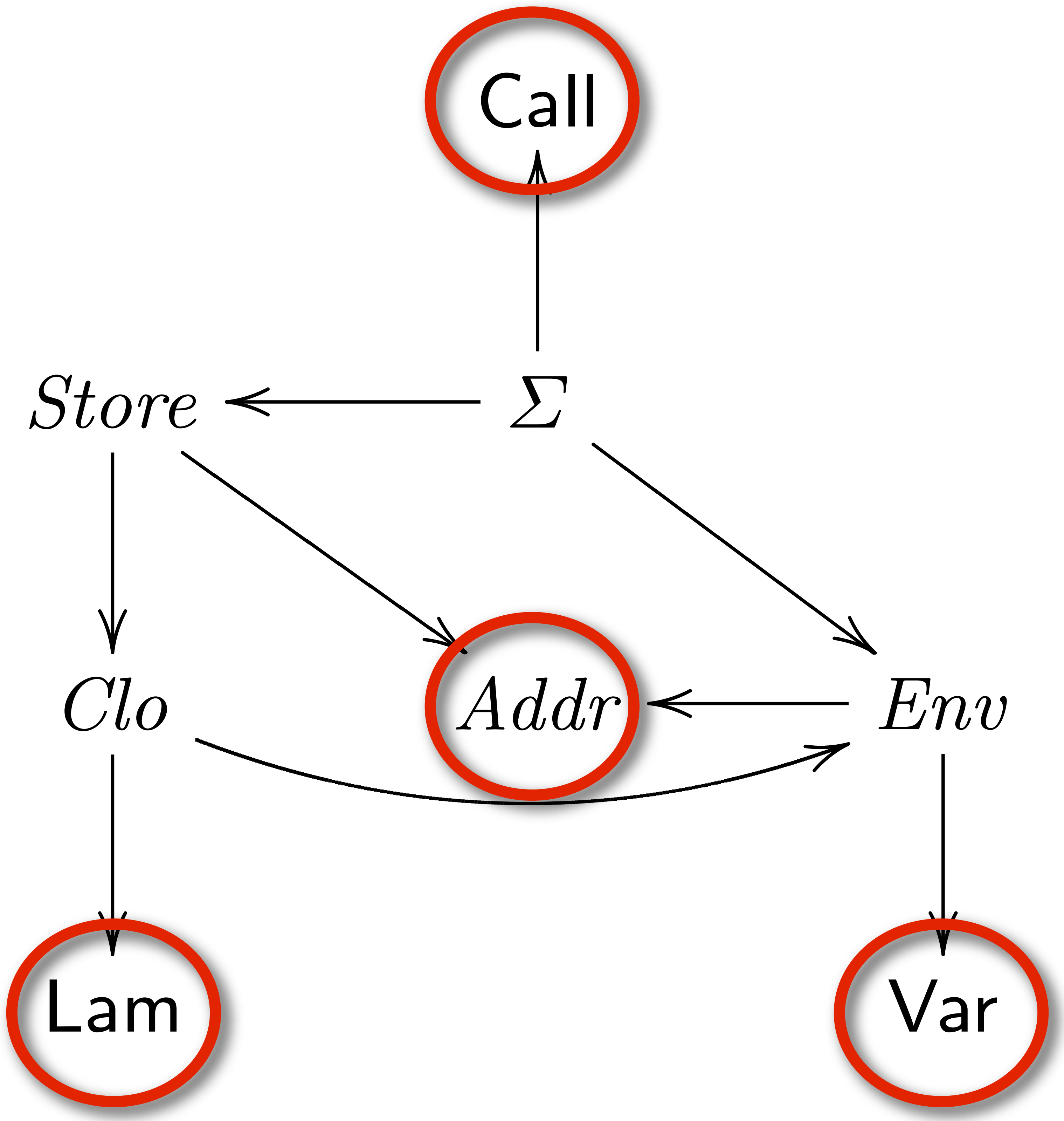
$$\downarrow$$

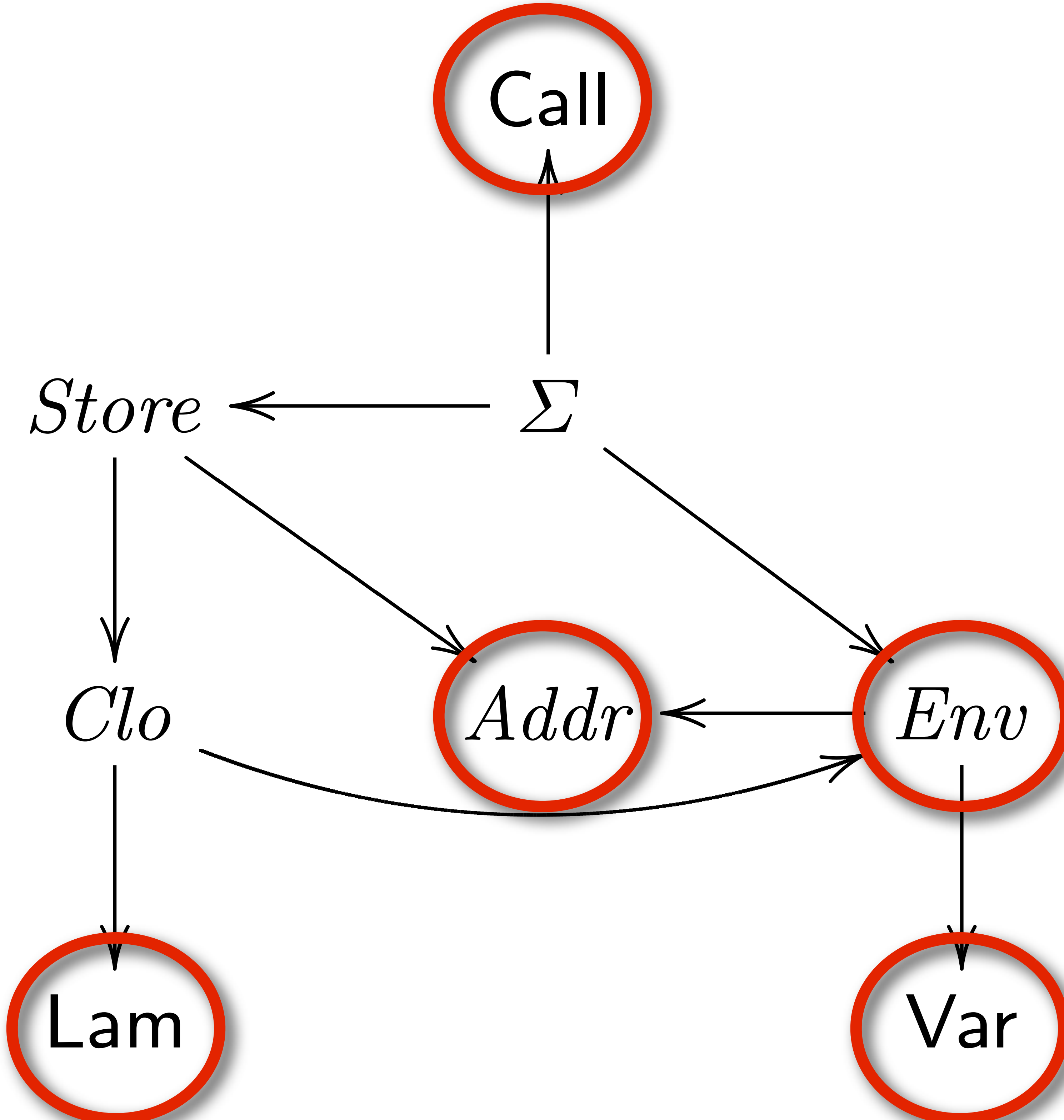
$$(\mathcal{P}(\Sigma), \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (\mathcal{P}(\hat{\Sigma}), \sqsubseteq_{\mathcal{P}(\hat{\Sigma})})$$

Trickling

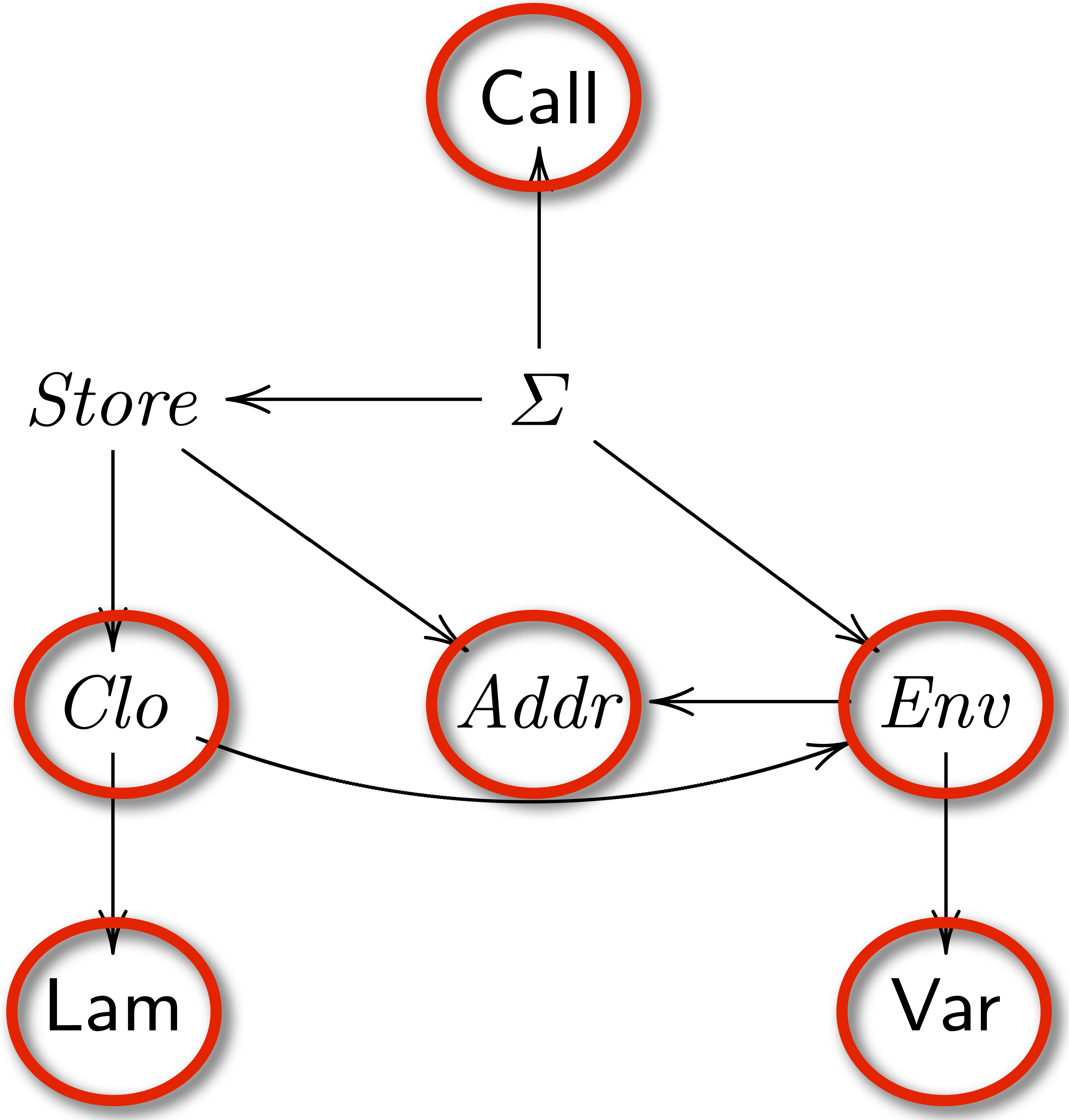

Call
$Store$
Σ
Clo
$Addr$
Env
Lam
Var


Call
Store
Σ
Clo
Addr
Env
Lam
Var

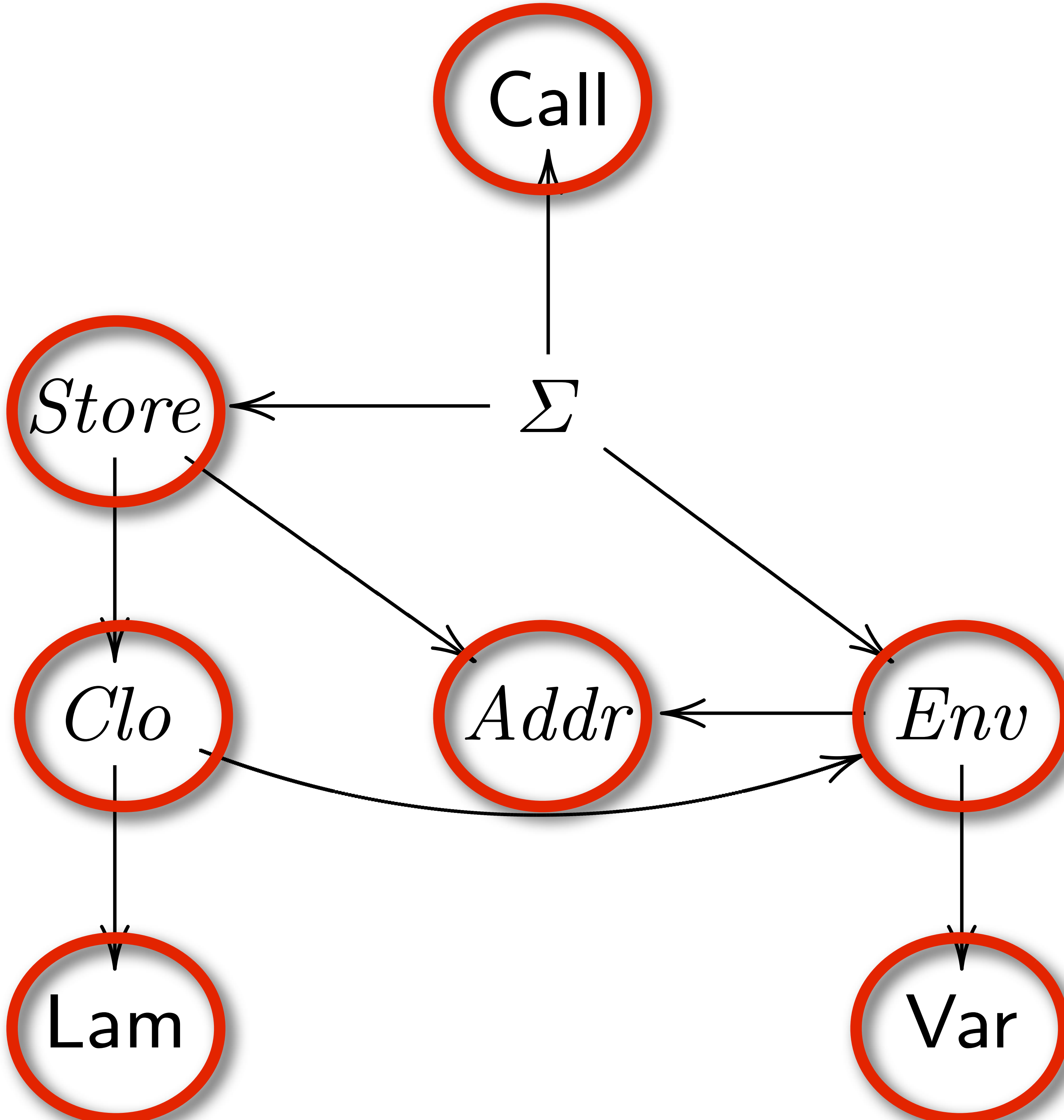

Call
Store
Σ
Clo
Addr
Env
Lam
Var

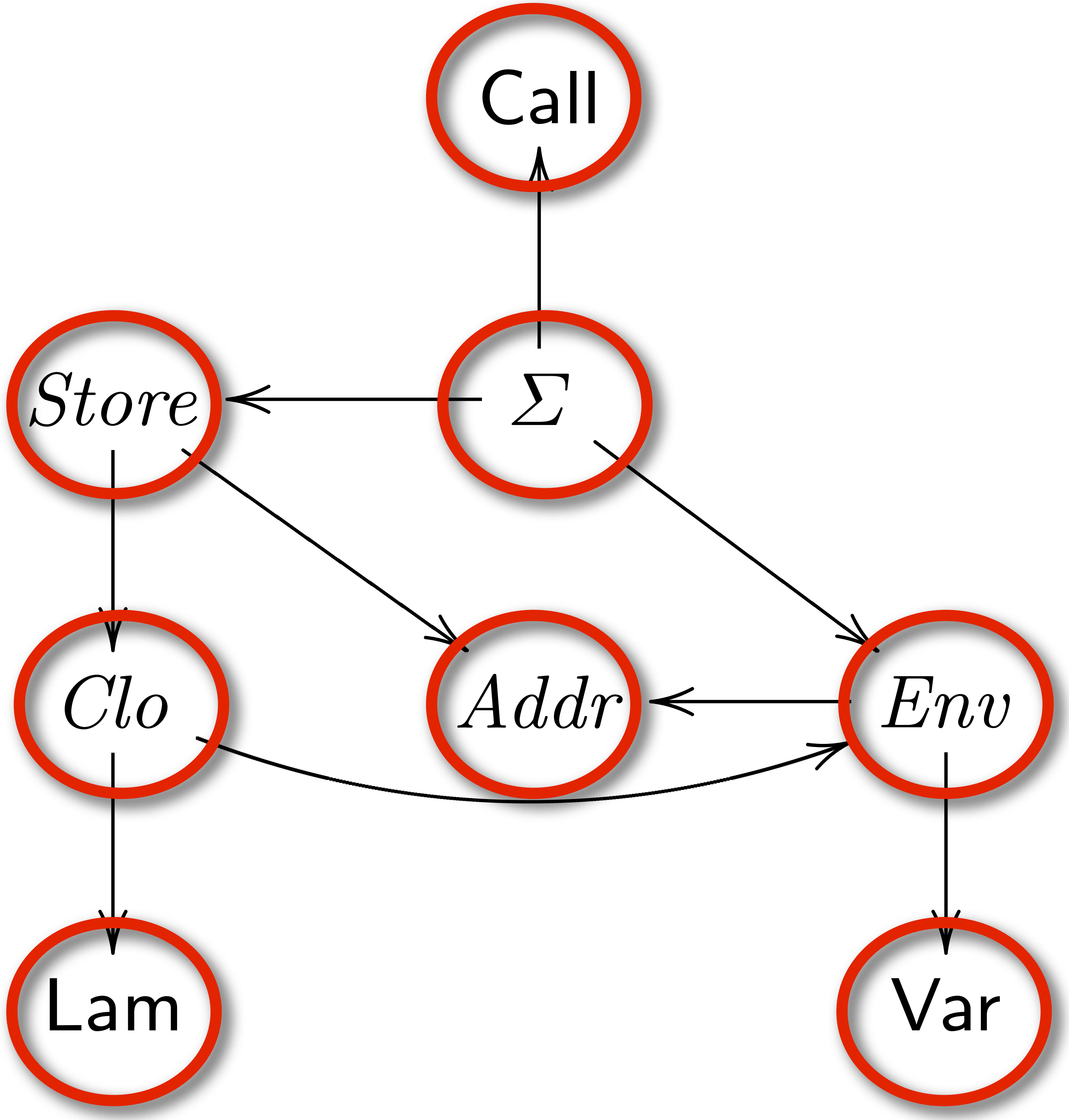


Call
Store
Σ
Clo
Addr
Env
Lam
Var



Call
Store
Σ
Clo
Addr
Env
Lam
Var


Call
Store
Σ
Clo
Addr
Env
Lam
Var



Call
$Store$
Σ
Clo
$Addr$
Env
Lam
Var

If structures $X_1, X_2, \ldots, X_n$ are Galois connections, then $F(X_1, X_2, \ldots, X_3)$ is also a Galois connection.

$$\frac{X_i \text{ is a Galois connection}}{F(X_1, X_2, \ldots, X_3) \text{ is a Galois connection}}$$

Some inference rules

$$(\mathcal{P}(A), \sqsubseteq_1) \underset{\lambda S.S}{\overset{\lambda S.S}{\leftrightarrows}} (\mathcal{P}(A), \sqsubseteq_1) \qquad \text{(power identity)}$$

$$\frac{(\mathcal{P}(A), \sqsubseteq_1) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (\mathcal{P}(\hat{A}), \sqsubseteq_2) \qquad (\mathcal{P}(B), \sqsubseteq_1') \underset{\alpha'}{\overset{\gamma'}{\leftrightarrows}} (\mathcal{P}(\hat{B}), \sqsubseteq_2')}{(\mathcal{P}(A \times B), \sqsubseteq_1'') \underset{\alpha''}{\overset{\gamma''}{\leftrightarrows}} (\mathcal{P}(\hat{A} \times \hat{B}), \sqsubseteq_2'')} \qquad \text{(power product)}$$

$$\frac{(\mathcal{P}(Y), \sqsubseteq_1) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (\mathcal{P}(\hat{Y}), \sqsubseteq_2)}{(\mathcal{P}(X \to Y), \sqsubseteq_1'') \underset{\alpha'}{\overset{\gamma'}{\leftrightarrows}} (\mathcal{P}(X \to \hat{Y}), \sqsubseteq_2'')} \qquad \text{(image)}$$

$$\frac{(\mathcal{P}(X), \sqsubseteq_1) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (\hat{X}, \sqsubseteq_2)}{(\mathcal{P}(X), \sqsubseteq_1) \underset{\alpha'}{\overset{\gamma'}{\leftrightarrows}} (\mathcal{P}(\hat{X}), \sqsubseteq_2')} \qquad \text{(power lift)}$$

$$\frac{(\mathcal{P}(X), \sqsubseteq_1) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (\mathcal{P}(\hat{X}), \sqsubseteq_2) \qquad (\mathcal{P}(Y), \sqsubseteq_1') \underset{\alpha'}{\overset{\gamma'}{\leftrightarrows}} (\mathcal{P}(\hat{Y}), \sqsubseteq_2')}{(\mathcal{P}(X \to Y), \sqsubseteq_1'') \underset{\alpha''}{\overset{\gamma''}{\leftrightarrows}} (\mathcal{P}(\hat{X} \to \hat{Y}), \sqsubseteq_2'')} \qquad \text{(function)}$$

$$\left(\mathcal{P}\left(\Sigma\right), \subseteq\right) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \left(\mathcal{P}(\hat{\Sigma}), \sqsubseteq_{\mathcal{P}(\hat{\Sigma})}\right)$$

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

$$\left(\mathcal{P}\left(\Sigma\right), \subseteq\right) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \left(\mathcal{P}(\hat{\Sigma}), \sqsubseteq_{\mathcal{P}\left(\hat{\Sigma}\right)}\right)$$

$$\Big\downarrow \; ?$$

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

Cousot & Cousot, 1979

$$\left(\mathcal{P}\left(\Sigma\right), \subseteq\right) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \left(\mathcal{P}(\hat{\Sigma}), \sqsubseteq_{\mathcal{P}(\hat{\Sigma})}\right)$$

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

$$(\mathcal{P}\left(\Sigma\right), \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (\mathcal{P}(\hat{\Sigma}), \sqsubseteq_{\mathcal{P}(\hat{\Sigma})})$$

$$\Big\downarrow$$

(Cousot^2, 1979)

$$(\leadsto) = \alpha \circ (\Rightarrow) \circ \gamma$$

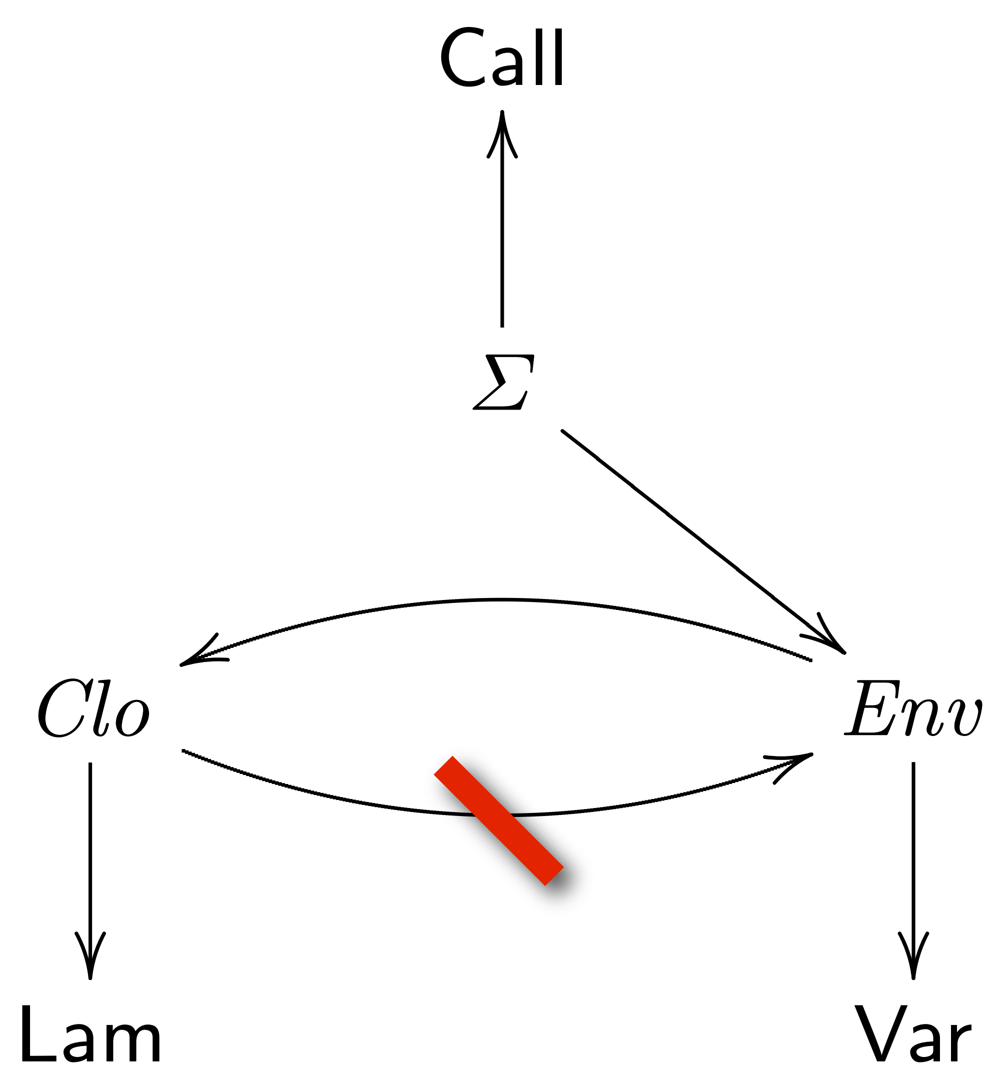
k-CFA (Shivers, 1991)

$$\overbrace{(\llbracket(f\ e_1 \ldots e_n)\rrbracket, \hat{\rho}, \hat{\sigma})}^{\hat{\varsigma}} \leadsto \overbrace{(call, \hat{\rho}'', \hat{\sigma}')}^{\hat{\varsigma}'}, \text{ where}$$

$$\begin{aligned}
(lam, \hat{\rho}') &\in \hat{\mathcal{E}}(f, \hat{\rho}, \hat{\sigma}) \\
lam &= \llbracket(\lambda\ (v_1 \ldots v_n)\ call)\rrbracket \\
\hat{a}_i &= \widehat{alloc}(v_i, \hat{\varsigma}) \\
\hat{\rho}'' &= \hat{\rho}'[v_i \mapsto \hat{a}_i] \\
\hat{\sigma}' &= \hat{\sigma} \sqcup [\hat{a}_i \mapsto \hat{\mathcal{E}}(e_i, \hat{\rho}, \hat{\sigma})],
\end{aligned}$$

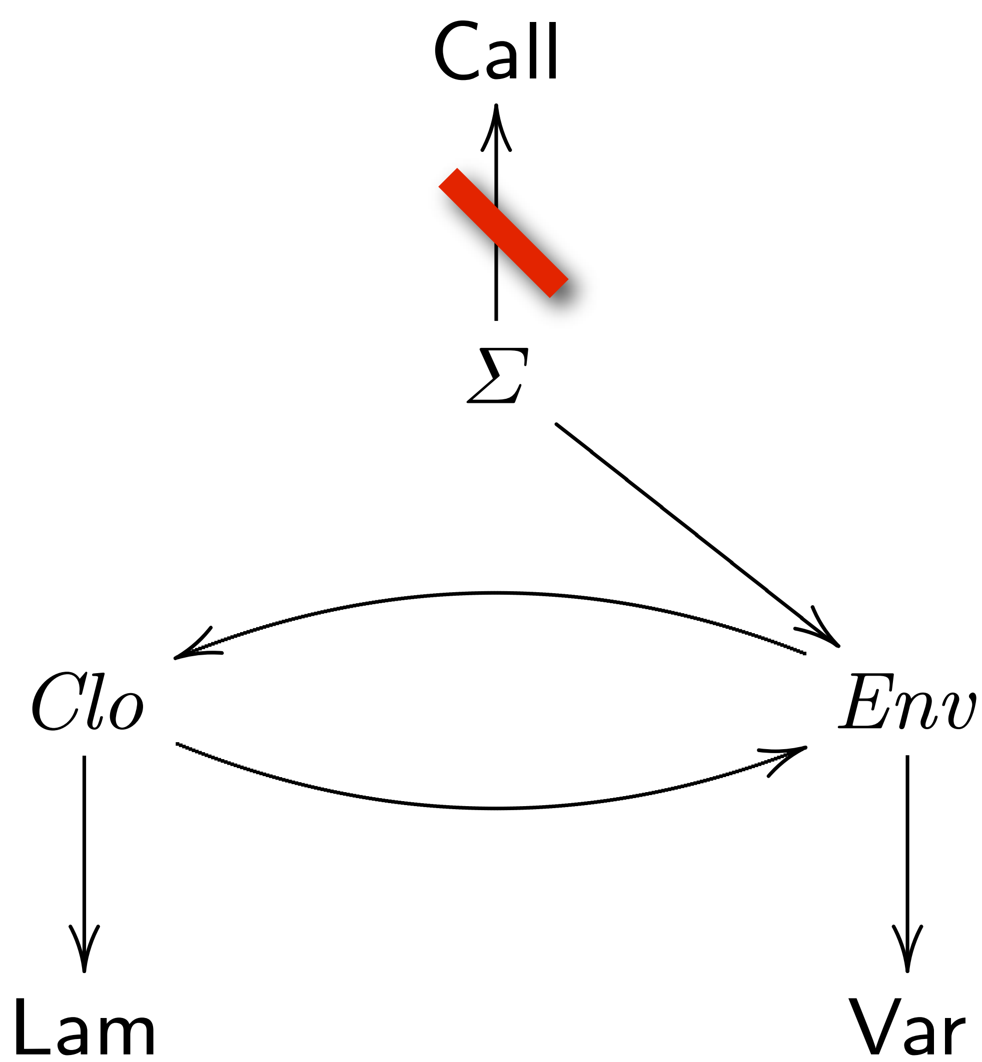
What if we snip a different edge?

Snip Clo-to-Env



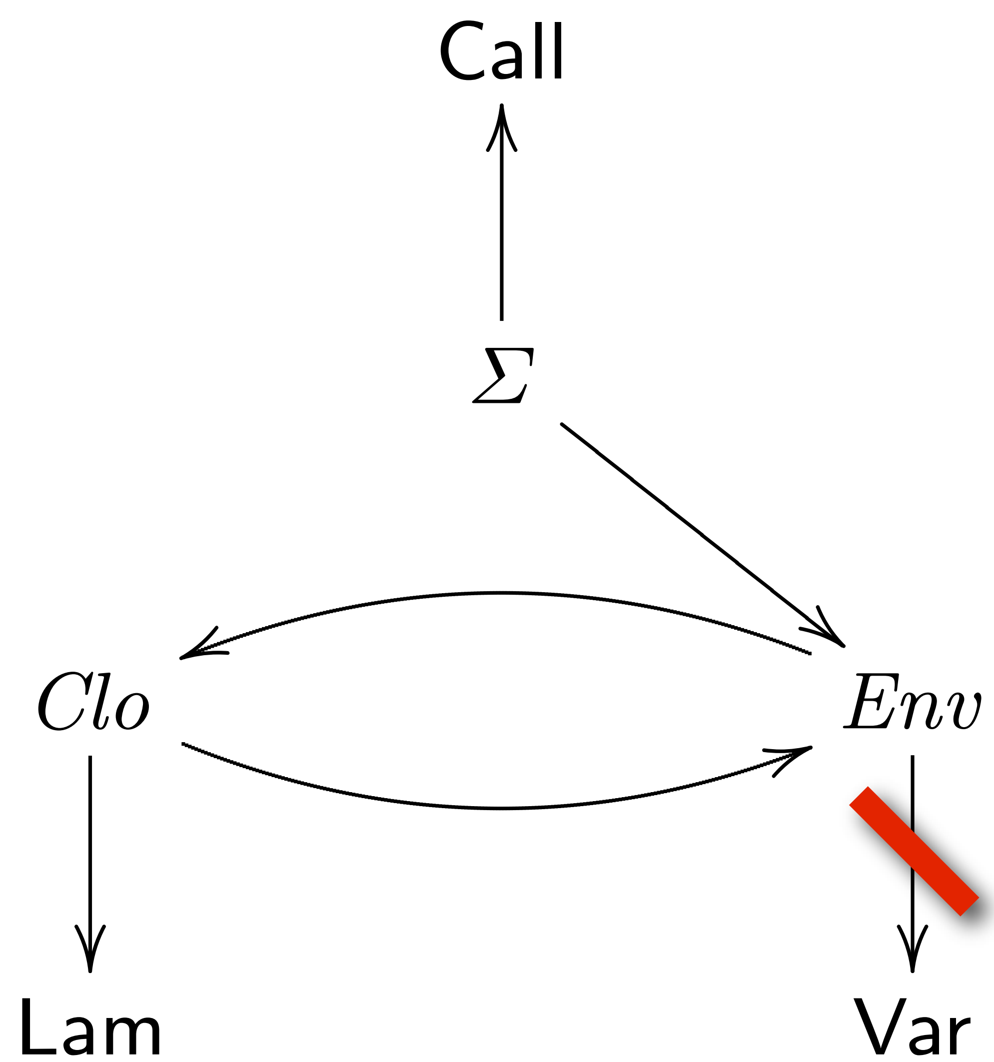
Environment-flow analysis

Snip Σ-to-Call?



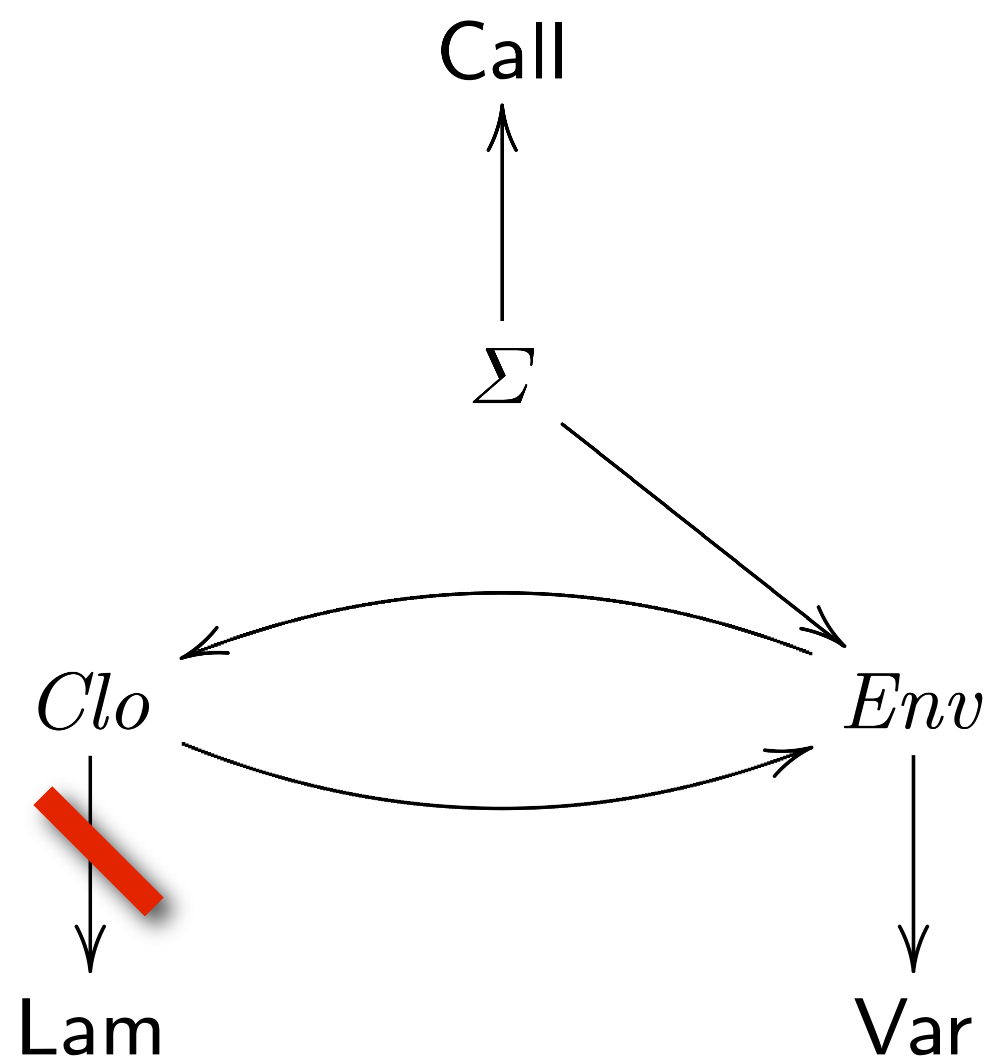
Controls flow-sensitivity.

Snip Env-to-Var?



Controls field-sensitivity.

Snip Clo-to-Lam?



No word to describe it.

Doggie bag

matt.might.net

@mattmight

Doggie bag

matt.might.net

@mattmight

Scott & Strachey, 1966

Doggie bag

matt.might.net
@mattmight

Scott & Strachey, 1966

Cousot & Cousot, 1977

Doggie bag

matt.might.net

@mattmight

Scott & Strachey, 1966

Cousot & Cousot, 1977

Cousot & Cousot, 1979

Doggie bag

matt.might.net

@mattmight

Scott & Strachey, 1966

Cousot & Cousot, 1977

Cousot & Cousot, 1979