Efficient control-flow analysis and beyond

Matt Might
University of Utah
matt.might.net

Clarifications

- *k*-CFA is not sole solution to constraints

For every polynomial-time algorithm you have,
there is an exponential algorithm I would rather run.

Alan Perlis

Objectives

- Two more popular CFAs
- Implementing small-step CFAs
- *k*-CFA for Java
- Abstract garbage collection
- Abstract counting
- Optimizations

Two more CFAs

poly/CFA (Wright *et al.*, 1998)

- Inspired by let-bound polymorphism
- Achieves effect of monomorphization
- Tends to be faster and more precise

poly/CFA (Wright *et al.*, 1998)

$$\widehat{Time} = \mathsf{Call} + \{0\}$$

$$\widehat{tick}(call, \hat{t}) = \begin{cases} call & \text{function is let-bound} \\ 0 & \text{otherwise.} \end{cases}$$

CPA (Agesen, 1995)

- CPA: Cartesian product algorithm
- Time = Product of argument information
- For example: Allocate types as time

CPA (Agesen, 1995)

$$\widehat{Time} = \mathsf{Type}^*$$

$$\widehat{tick}([\![(f\ e_1 : \tau_1 \cdots e_n : \tau_n)]\!], \hat{t}) = \langle \tau_1, \ldots, \tau_n \rangle$$

Implementing small-step 0CFA

Implementation

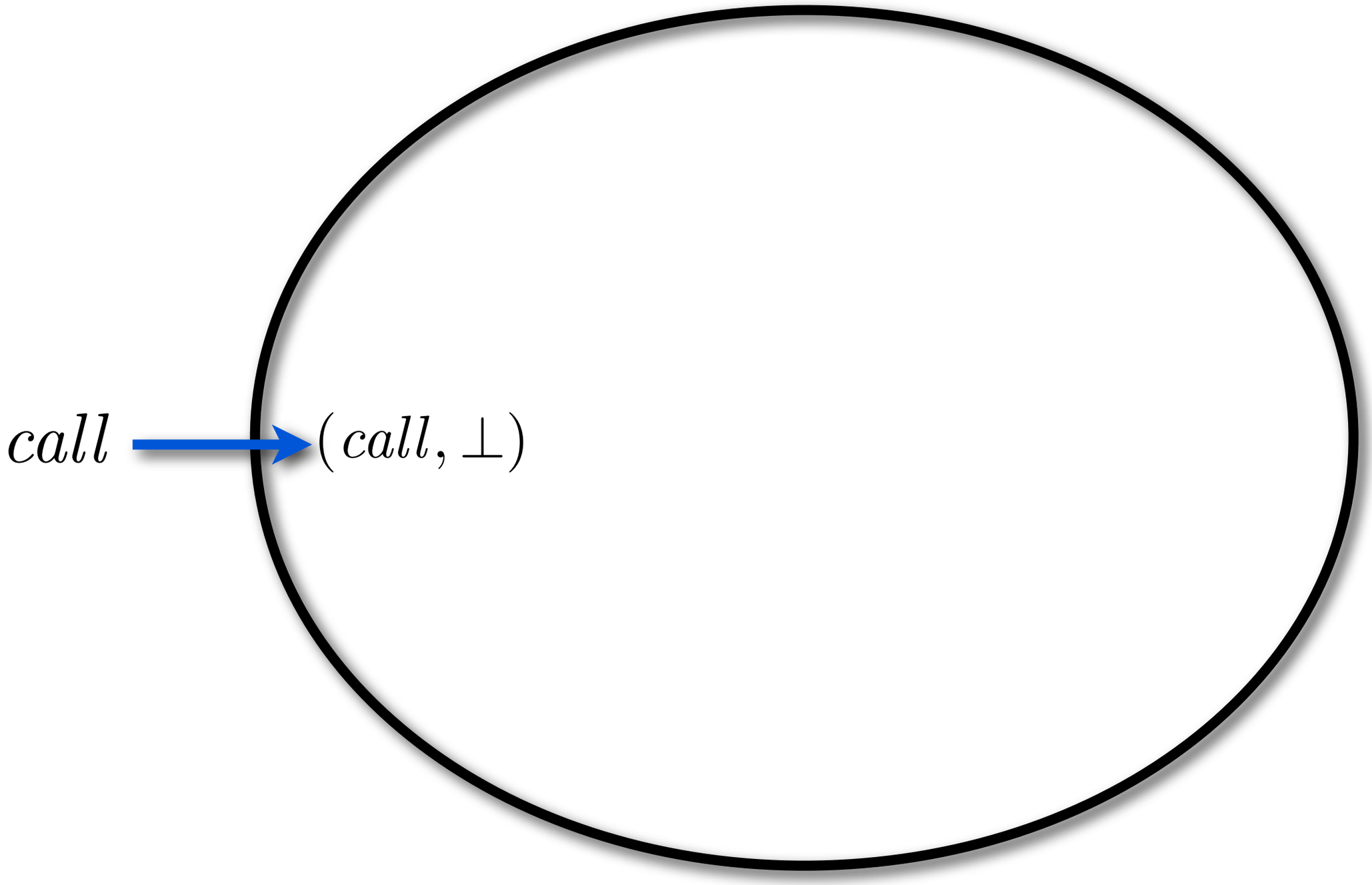
- Naïve implementation is straightforward
- Efficient implementation needs thought

Running 0CFA

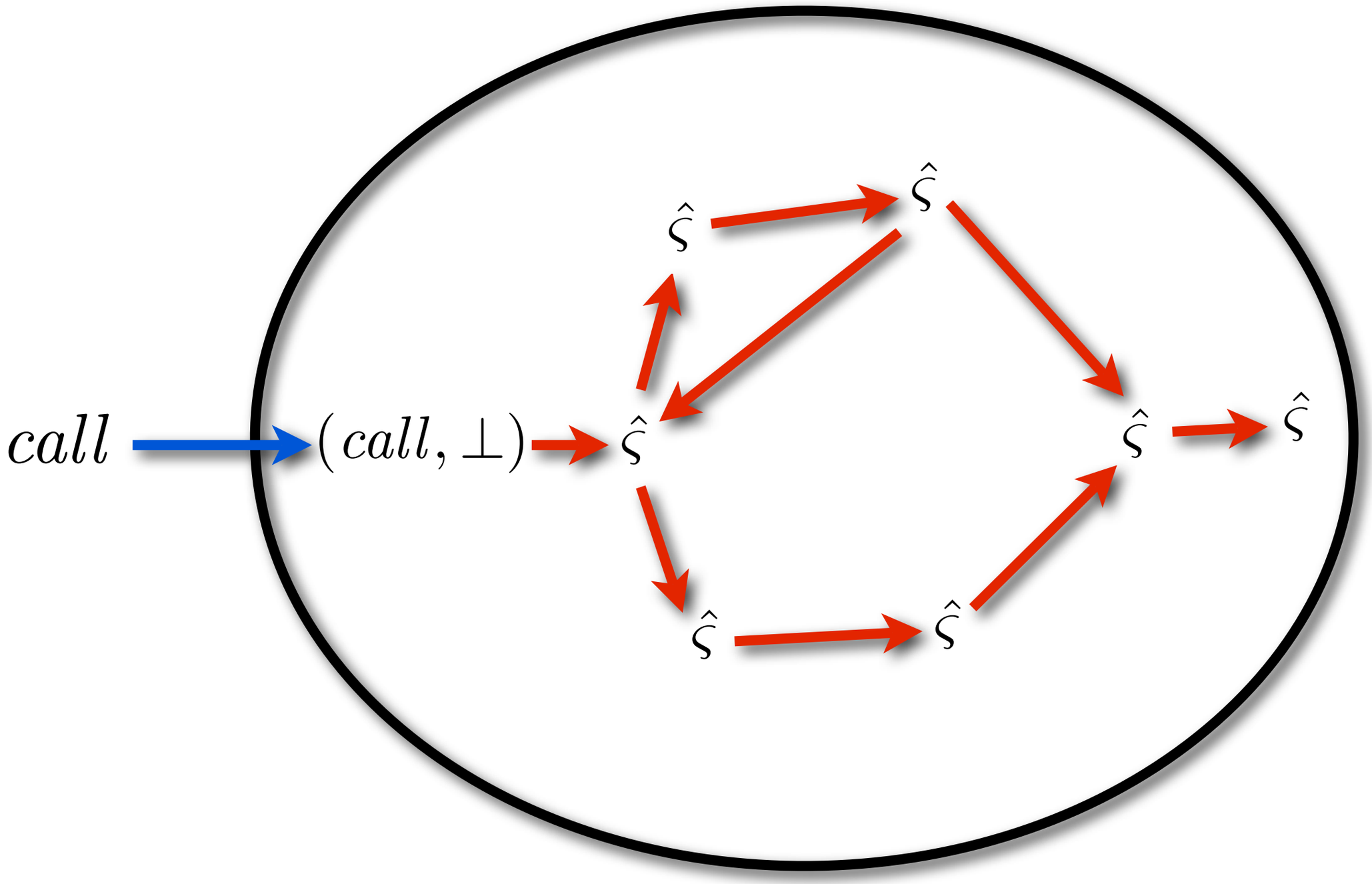
Running 0CFA

$$\hat{\Sigma}$$

Running 0CFA



Running 0CFA



Complexity of small-step 0CFA

Number of states: $|\mathsf{Call}| \cdot \left(2^{|\mathsf{Lam}|}\right)^{|\mathsf{Var}|}$

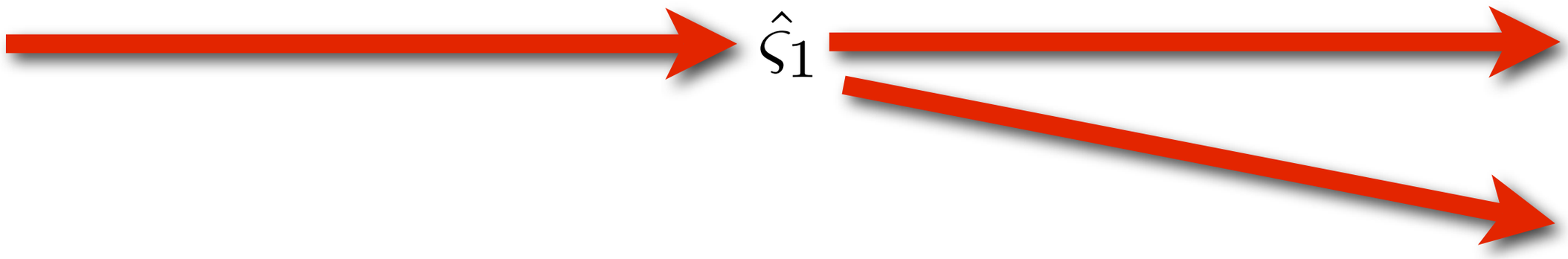
Techniques

- Exploiting monotonicity
- Contracting the state-space

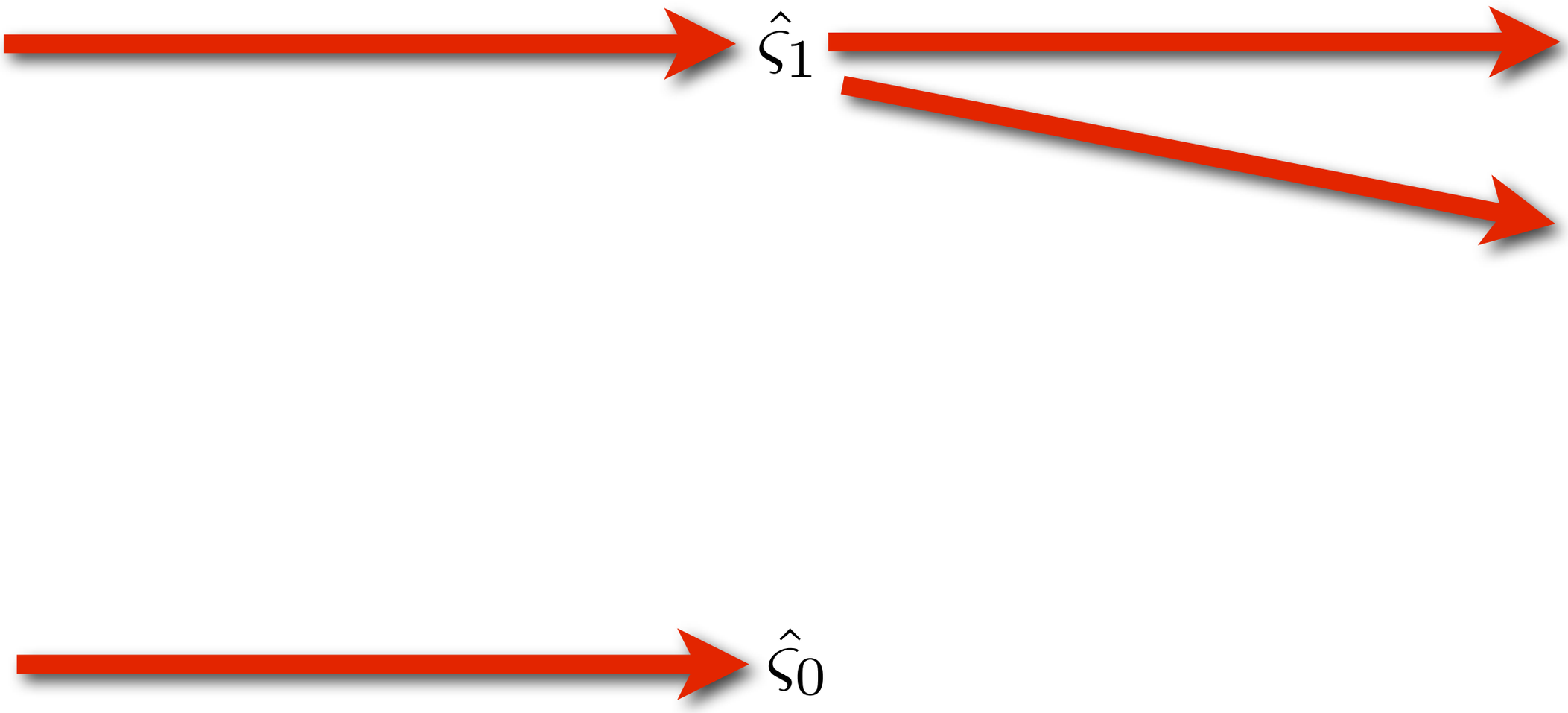
Exploiting monotonicity

$$
\begin{tikzcd}
\hat{s}_0 \arrow[r, "\rightsquigarrow"] \arrow[d, "\sqsubseteq"'] & \hat{s}_0' \arrow[d, "\sqsubseteq"] \\
\hat{s}_1 \arrow[r, dotted, "\rightsquigarrow"'] & \hat{s}_1'
\end{tikzcd}
$$

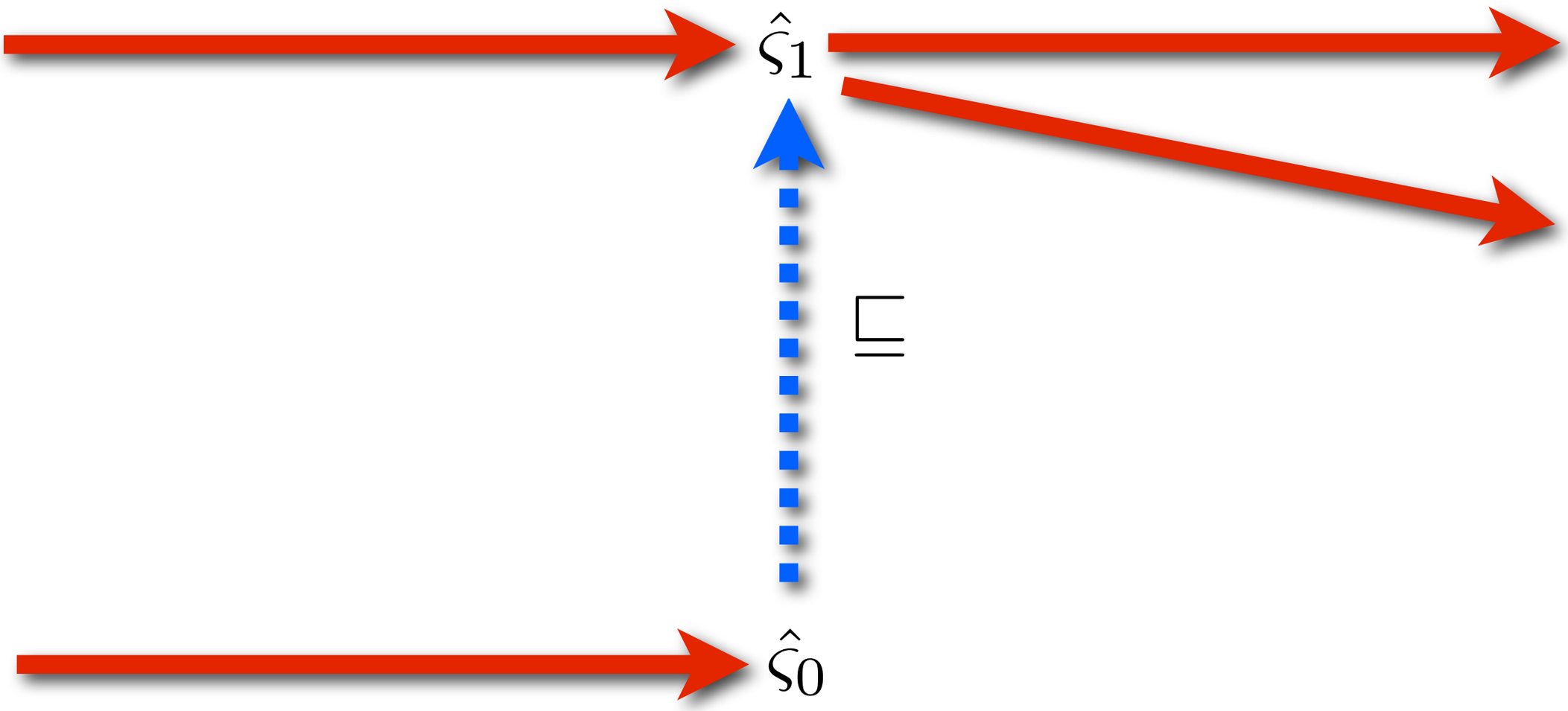
Exploiting monotonicity



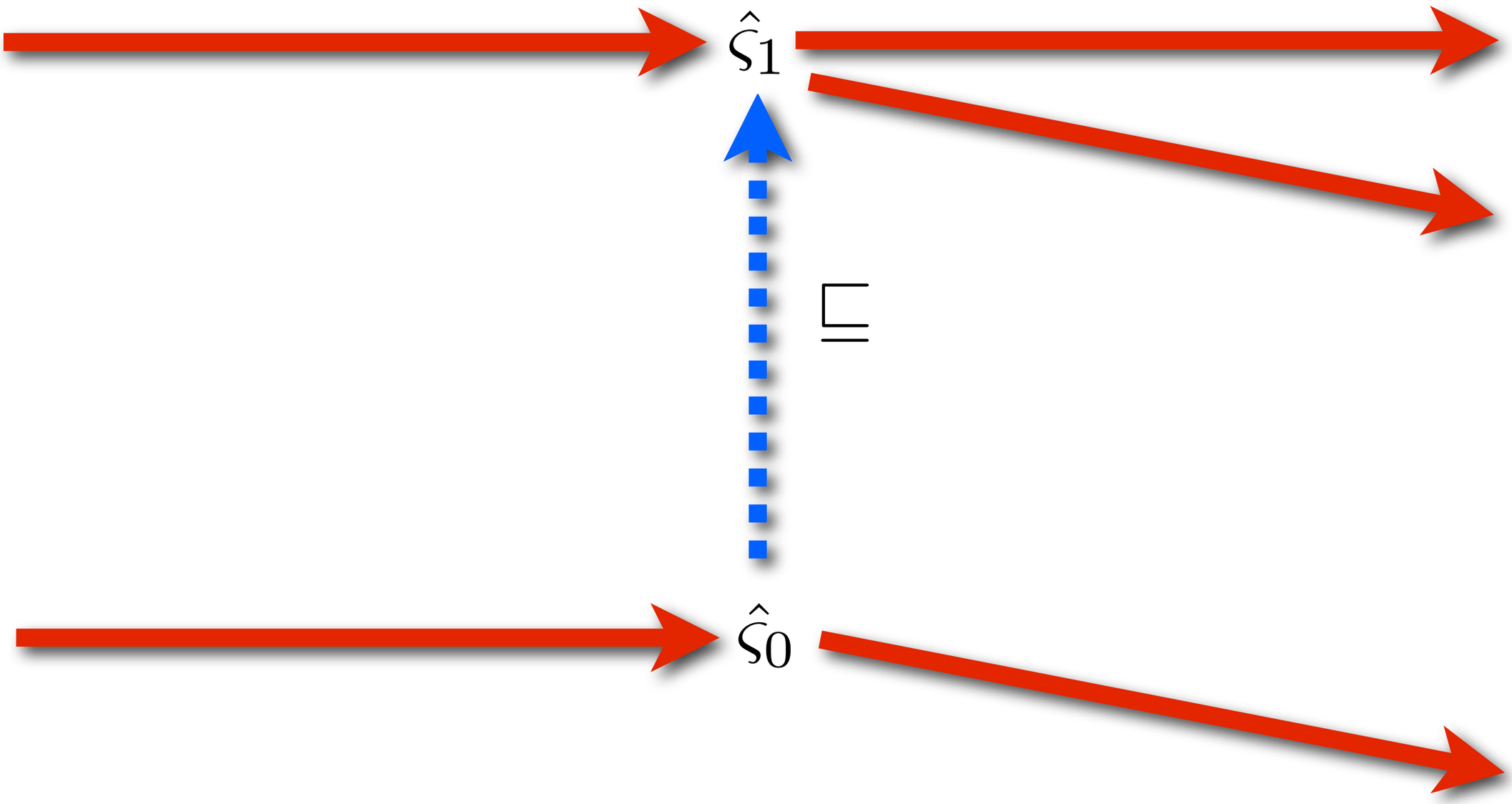
Exploiting monotonicity



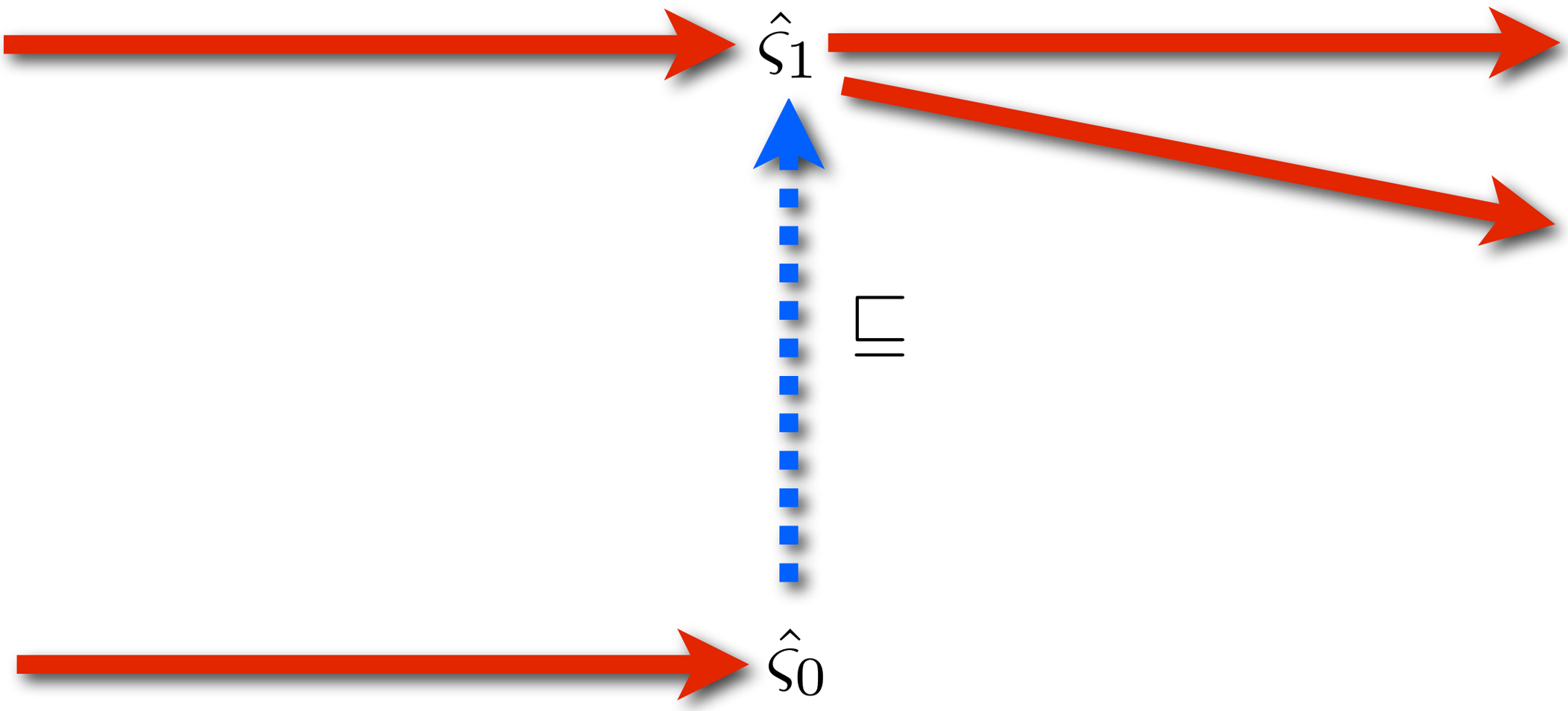
Exploiting monotonicity



Exploiting monotonicity



Exploiting monotonicity



Exploiting monotonicity

$$O\left(|\mathsf{Call}| \cdot \left(2^{|\mathsf{Lam}|}\right)^{|\mathsf{Var}|}\right)$$

System state-space

$$\Xi = \mathcal{P}(\Sigma)$$

System state-space

$$\Xi = \mathcal{P}(\Sigma)$$

$$f : \Xi \longrightarrow \Xi$$

System transition

$$f(\xi) = \{\varsigma' \mid \varsigma \in \xi \text{ and } \varsigma \Rightarrow \varsigma'\} \cup \{\varsigma_0\}$$

Abstract system

$$\hat{\Xi} = \mathcal{P}(\hat{\Sigma})$$

$$\hat{f} : \hat{\Xi} \longrightarrow \hat{\Xi}$$

Abstract transition

$$\hat{f}(\hat{\xi}) = \{\hat{\varsigma}' \mid \hat{\varsigma} \in \hat{\xi} \text{ and } \hat{\varsigma} \rightsquigarrow \hat{\varsigma}'\} \cup \{\hat{\varsigma}_0\}$$

0CFA system

$$\hat{\Xi} = \mathcal{P}(\mathsf{Call} \times \widehat{Store})$$

0CFA system

$$\hat{\Xi} = \mathcal{P}(\mathsf{Call}) \times \widehat{Store}$$

Monotonic transition

$$\hat{f}(C, \hat{\sigma}) = (C' \cup C, \hat{\sigma}'), where$$

$$S' = \left\{ \hat{\varsigma}' : call \in C \text{ and } (call, \hat{\sigma}) \rightsquigarrow \hat{\varsigma}' \right\}$$

$$C' = \left\{ call : (call, _) \in S' \right\}$$

$$\hat{\sigma}' = \bigsqcup_{(_, \hat{\sigma}) \in S'} \hat{\sigma}$$

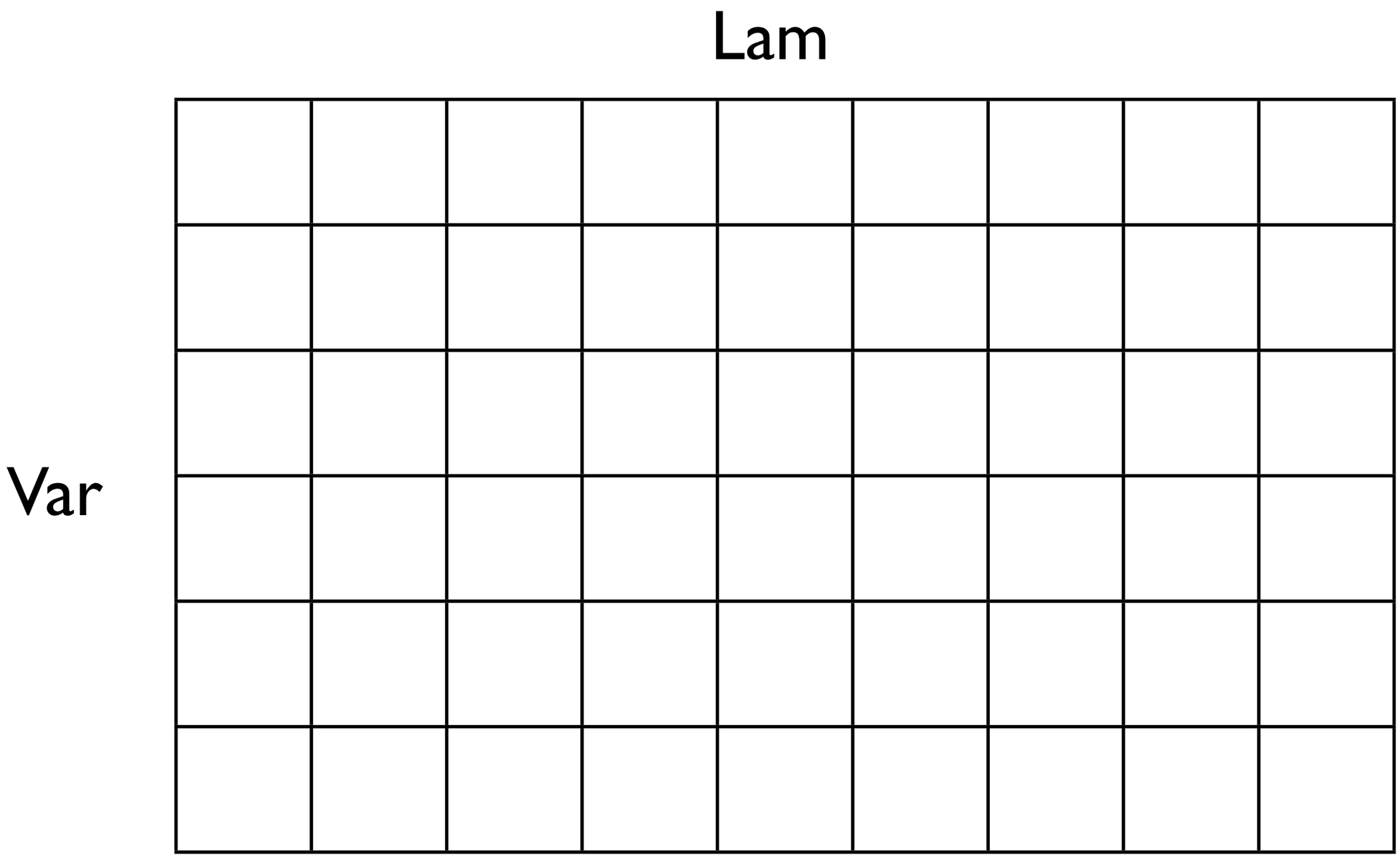
Flow-sensitive 0CFA

- Fixed point in $O(|\mathsf{Call}| \cdot |\mathsf{Var}| \cdot |\mathsf{Lam}|)$
- Analysis is flow-sensitive

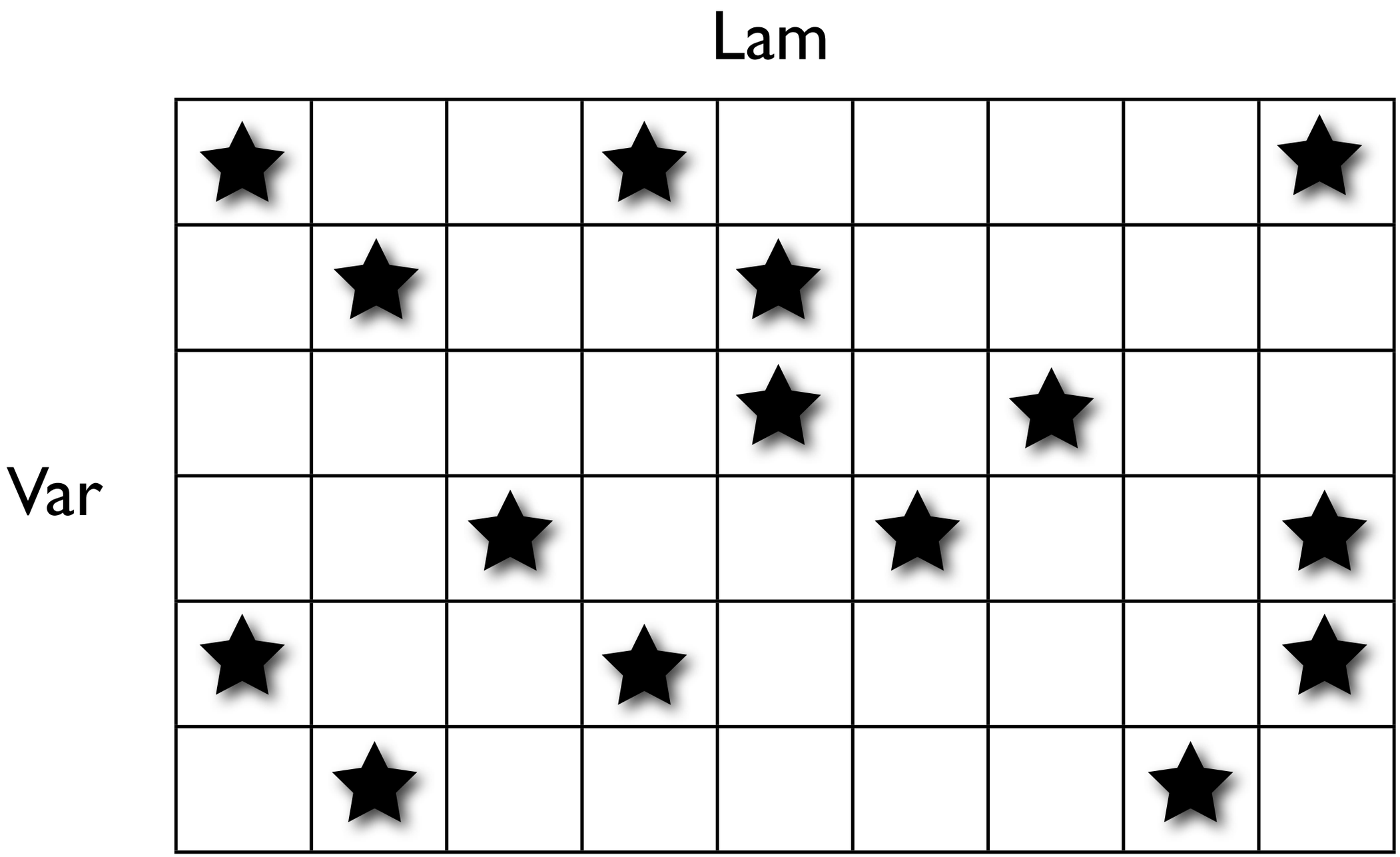
Flow-insensitive 0CFA

- Start with state $\hat{\xi}_0 = (\text{Call}, \perp)$
- Total complexity is cubic

Cubic 0CFA visualized



Cubic 0CFA visualized



Per-point 0CFA

- Use abstract system $\hat{\Xi} = \text{Call} \rightarrow \widehat{Store}$
- Total complexity is quartic
- Often faster than cubic 0CFA
- Much more precise than cubic 0CFA

Theory and practice

"Expensive" 0CFA often faster than "efficient" 0CFA.

Theory and practice

"Expensive" 0CFA often faster than "efficient" 0CFA.

Why?

Theory and practice

"Expensive" 0CFA often faster than "efficient" 0CFA.

Why?

Precise flow graphs are very sparse.

Implementing 1CFA

System-space for 1CFA

$$\hat{\Xi} = \mathcal{P}(\mathsf{Call} \times \widehat{BEnv} \times \widehat{Store})$$

System-space for 1CFA

$$\hat{\Xi} = \mathcal{P}(\mathsf{Call} \times \widehat{BEnv}) \times \widehat{Store}$$

System-space for 1CFA

$\hat{\Xi} = \mathsf{Call} \to \mathcal{P}(\widehat{} \times \widehat{})$

System-space for 1CFA

$\widehat{BEnv}$

Call

$\widehat{Clo}$

$\widehat{Addr}$

System-space for 1CFA

$$\mathsf{Call} \times \widehat{BEnv} + \widehat{Addr} \times \widehat{Clo}$$

$$= |\mathsf{Call}| \times (|\mathsf{Var}| \times |\mathsf{Call}|)^{|\mathsf{Var}|} + \left(|\mathsf{Lam}| \times (|\mathsf{Var}| \times |\mathsf{Call}|)^{|\mathsf{Var}|}\right)^{|\mathsf{Var}| \times |\mathsf{Call}|}$$

Excursion:

k-CFA for Java

Featherweight Java (Igarashi et al., 1999)

$$\texttt{L} ::= \texttt{class C extends C \{}\overline{\texttt{C}}\ \overline{\texttt{f}}\texttt{; K }\overline{\texttt{M}}\texttt{\}}$$

$$\texttt{K} ::= \texttt{C(}\overline{\texttt{C}}\ \overline{\texttt{f}}\texttt{)\{super(}\overline{\texttt{f}}\texttt{); this.}\overline{\texttt{f}}\texttt{=}\overline{\texttt{f}}\texttt{;\}}$$

$$\texttt{M} ::= \texttt{C m(}\overline{\texttt{C}}\ \overline{\texttt{x}}\texttt{)\{ return e; \}}$$

$$\texttt{e} ::= \texttt{x} \mid \texttt{e.f} \mid \texttt{e.m(}\overline{\texttt{e}}\texttt{)} \mid \texttt{new C(}\overline{\texttt{e}}\texttt{)} \mid \texttt{(C)e}$$

Abstract state-space

$$\hat{\varsigma} \in \hat{\Sigma} = \widehat{\mathsf{VExp}} \times \widehat{BEnv} \times \widehat{Store} \times \widehat{KontPtr} \times \widehat{Time}$$

$$\hat{\beta} \in \widehat{BEnv} = \mathsf{Var} \rightharpoonup \widehat{Addr}$$

$$\hat{\sigma} \in \widehat{Store} = \widehat{Addr} \to \hat{D}$$

$$\hat{d} \in \hat{D} = \mathcal{P}\left(\widehat{Val}\right)$$

$$\widehat{val} \in \widehat{Val} = \widehat{Obj} + \widehat{Kont}$$

$$\hat{o} \in \widehat{Obj} = \mathsf{Class} \times \widehat{BEnv}$$

$$\hat{\kappa} \in \widehat{Kont} = \widehat{\mathsf{Ctxt}} \times \widehat{BEnv} \times \widehat{KontPtr}$$

$\hat{a} \in \widehat{Addr}$ is a **finite** set of addresses

$\hat{t} \in \widehat{Time}$ is a **finite** set of time-stamps

$$\hat{p}^{\hat{\kappa}} \in \widehat{KontPtr} \subseteq \widehat{Addr}$$

Abstract state-space

$$\hat{\varsigma} \in \hat{\Sigma} = \mathsf{Body} \times \widehat{BEnv} \times \widehat{Store} \times \widehat{KontPtr} \times \widehat{Time}$$

$$\hat{\beta} \in \widehat{BEnv} = \mathsf{Var} \rightharpoonup \widehat{Addr}$$

$$\hat{\sigma} \in \widehat{Store} = \widehat{Addr} \rightarrow \hat{\square}$$

$$\hat{\square} \in \hat{\square} = \mathcal{P}\left(\widehat{Val}\right)$$

$$\widehat{val} \in \widehat{Val} = \widehat{Clo} + \widehat{Kont}$$

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

$$\hat{\kappa} \in \widehat{Kont} = \mathsf{Var} \times \mathsf{Body} \times \widehat{Env} \times \widehat{KontPtr}$$

$\square \in \widehat{Addr}$ is a **finite** set of addresses

$\hat{\square} \in \widehat{Time}$ is a **finite** set of time-stamps

$$\hat{\square} \in \widehat{KontPtr} \subseteq \widehat{Addr}$$

$$\hat{\varsigma} \in \hat{\Sigma} = \widehat{\mathsf{VExp}} \times \widehat{BEnv} \times \widehat{Store} \times \widehat{KontPtr} \times \widehat{Time}$$

$$\hat{\beta} \in \widehat{BEnv} = \mathsf{Var} \rightharpoonup \widehat{Addr}$$

$$\hat{\sigma} \in \widehat{Store} = \widehat{Addr} \rightarrow \hat{D}$$

$$\hat{d} \in \hat{D} = \mathcal{P}\left(\widehat{Val}\right)$$

$$\widehat{val} \in \widehat{Val} = \widehat{Obj} + \widehat{Kont}$$

$$\hat{o} \in \widehat{Obj} = \mathsf{Class} \times \widehat{BEnv}$$

$$\hat{\kappa} \in \widehat{Kont} = \widehat{\mathsf{Ctxt}} \times \widehat{BEnv} \times \widehat{KontPtr}$$

$\hat{a} \in \widehat{Addr}$ is a **finite** set of addresses

$\hat{t} \in \widehat{Time}$ is a **finite** set of time-stamps

$$\hat{p} \in \widehat{KontPtr} \subseteq \widehat{Addr}$$

Abstract contexts

$$
\begin{aligned}
\hat{E} \in \widehat{\mathsf{VExp}} ::= \; & \hat{d} \mid v \\
\mid \; & \hat{E}\,\texttt{.}\,f \\
\mid \; & \hat{E}\,\texttt{.}\,m\texttt{(}\hat{E}_1 \ldots \hat{E}_n\texttt{)} \\
\mid \; & \texttt{new}\; C\texttt{(}\hat{E}_1, \ldots, \hat{E}_n\texttt{)} \\
\mid \; & \texttt{(}C\texttt{)}\; \hat{E}
\end{aligned}
$$

Abstract contexts

$$
\begin{aligned}
\hat{E} \in \widehat{\mathsf{VExp}} ::= \; & \hat{d} \mid v \\
\mid \; & \hat{E}\,.\,f \\
\mid \; & \hat{E}\,.\,m(\hat{E}_1 \ldots \hat{E}_n) \\
\mid \; & \texttt{new}\; C(\hat{E}_1, \ldots, \hat{E}_n) \\
\mid \; & (C)\; \hat{E}
\end{aligned}
$$

$$
\begin{aligned}
\hat{C} \in \widehat{\mathsf{Ctxt}} ::= \; & \square \\
\mid \; & \hat{C}\,.\,f \\
\mid \; & e\,.\,m(\hat{d}_1, \ldots, \hat{d}_n, \hat{C}, e_1, \ldots, e_m) \\
\mid \; & \hat{C}\,.\,m(\hat{d}_1, \ldots, \hat{d}_n) \\
\mid \; & \texttt{new}\; C(\hat{d}_1, \ldots, \hat{d}_n, \hat{C}, e_1, \ldots, e_m) \\
\mid \; & (C)\; \hat{C}
\end{aligned}
$$

Abstract semantics

$(\hat{C}[e], \hat{\beta}, \hat{\sigma}, \overset{\hat{\kappa}}{p}, \hat{t}) \rightsquigarrow (e, \hat{\beta}, \hat{\sigma}', \overset{\hat{\kappa}}{p}{}', \hat{t}')$, where

$$\hat{\kappa} = (\hat{C}, \hat{\beta}, \overset{\hat{\kappa}}{p})$$

$$\overset{\hat{\kappa}}{p}{}' = \widehat{alloc}_{\hat{\kappa}}(\ldots)$$

$$\hat{\sigma}' = \hat{\sigma} \sqcup [\overset{\hat{\kappa}}{p}{}' \mapsto \{\hat{\kappa}\}]$$

$$\hat{t}' = \widehat{tick}(\ldots)$$

Abstract semantics

$$
\begin{aligned}
(\llbracket \hat{d}_f\,.\,m(\hat{d}_1, \ldots, \hat{d}_n) \rrbracket, \hat{\beta}, \hat{\sigma}, \hat{\overset{\kappa}{p}}, \hat{t}) \rightsquigarrow (e, \hat{\beta}', \hat{\sigma}', \hat{\overset{\kappa}{p}}, \hat{t}'), &\text{ where} \\
\hat{\mathcal{M}} &: \hat{D} \times \mathsf{MethodCall} \to \mathcal{P}\,(\mathsf{Method}) \\
M &\in \hat{\mathcal{M}}(\hat{d}_f, m) \\
M &= \llbracket C'\ m\ (C_1\ v_1, \ldots, C_n\ v_n)\ e \rrbracket \\
\hat{a}_i &= \widehat{alloc}(v_i, \ldots) \\
\hat{\beta}' &= [\texttt{this} \mapsto \hat{d}_f, v_i \mapsto \hat{a}_i] \\
\hat{\sigma}' &= \hat{\sigma} \sqcup [\hat{a}_i \mapsto \hat{d}_i] \\
\hat{t}' &= \widehat{tick}(\ldots)
\end{aligned}
$$

Abstract semantics

$$(\llbracket \texttt{new}\ C\texttt{(}\hat{d}_1, \ldots, \hat{d}_n\texttt{)} \rrbracket, \hat{\beta}, \hat{\sigma}, \overset{\hat{\kappa}}{p}, \hat{t}) \rightsquigarrow (\hat{d}, \hat{\beta}, \hat{\sigma}', \overset{\hat{\kappa}}{p}, \hat{t}'), \text{ where}$$

$$(\hat{\mathcal{K}}, n, m) = \hat{\mathcal{C}}(C)$$

$$\hat{\mathcal{K}} : \hat{D}^n \times \widehat{Addr}^m \to (\widehat{Addr} \to \hat{D}) \times (\mathsf{Field} \to \widehat{Addr})$$

$$\hat{a}_i = \widehat{alloc}_i(\ldots)$$

$$(\Delta\hat{\sigma}, \hat{\beta}') = \hat{\mathcal{K}}(\hat{d}_1, \ldots, \hat{d}_n, \hat{a}_1, \ldots, \hat{a}_m)$$

$$\hat{\sigma}' = \hat{\sigma} \sqcup \Delta\hat{\sigma}$$

$$\hat{d} = \left\{ (C, \hat{\beta}') \right\}$$

$$\hat{t}' = \widehat{tick}(\ldots)$$

Abstract semantics

$$(\hat{d}, \hat{\beta}, \hat{\sigma}, \overset{\hat{\kappa}}{p}, \hat{t}) \rightsquigarrow (\hat{C}[\hat{d}], \hat{\beta}', \hat{\sigma}, \overset{\hat{\kappa}}{p}{}', \hat{t}'), \text{ where}$$

$$(\hat{C}, \hat{\beta}', \overset{\hat{\kappa}}{p}{}') \in \hat{\sigma}(\overset{\hat{\kappa}}{p})$$

$$\hat{t}' = \widehat{tick}(\ldots)$$

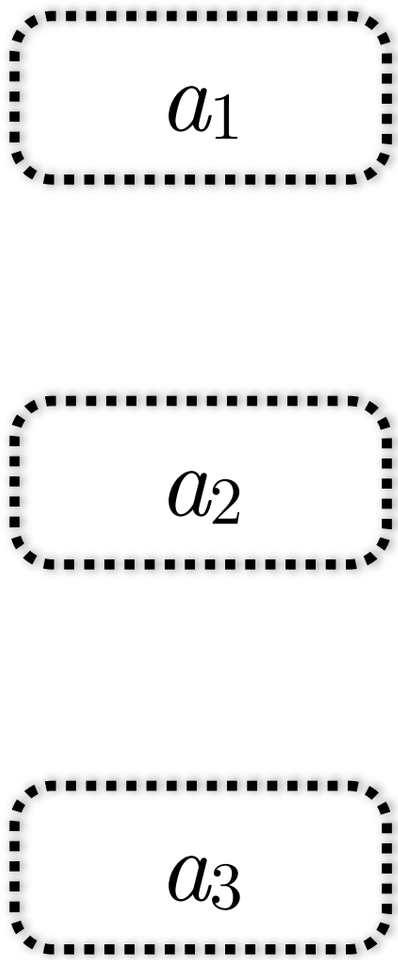
Abstract garbage collection

Main idea

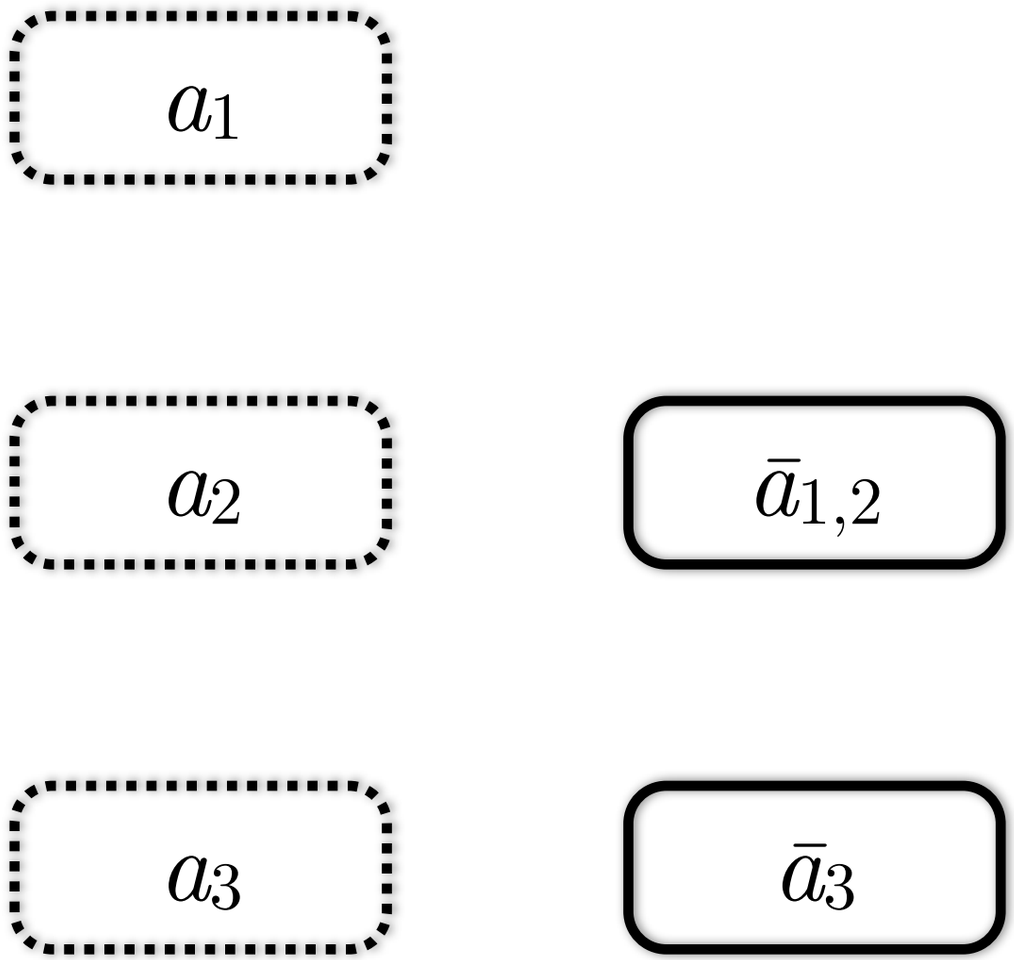
- Garbage collect a concrete semantics
- Perform an abstract interpretation
- Watch precision and speed go up

Abstract GC Illustrated

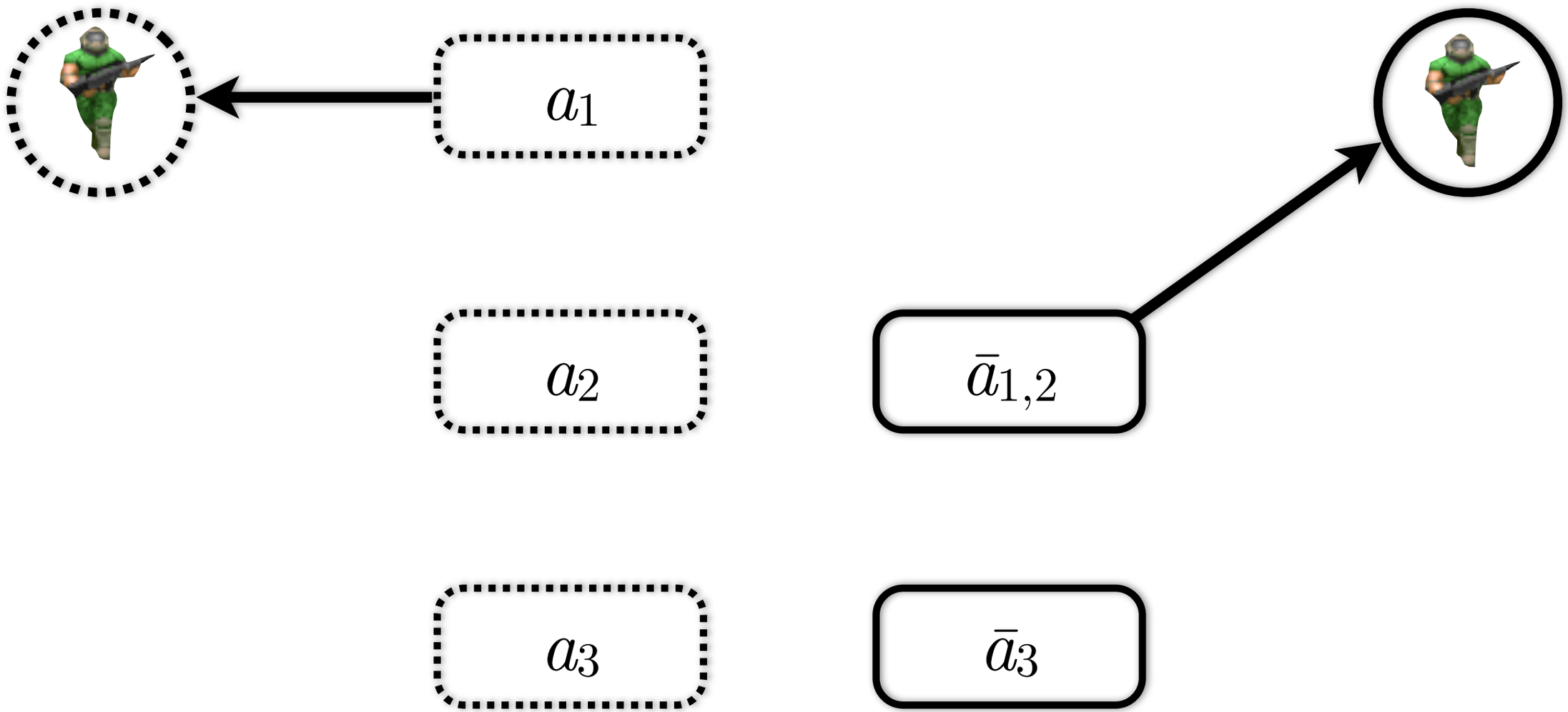
Abstract GC Illustrated



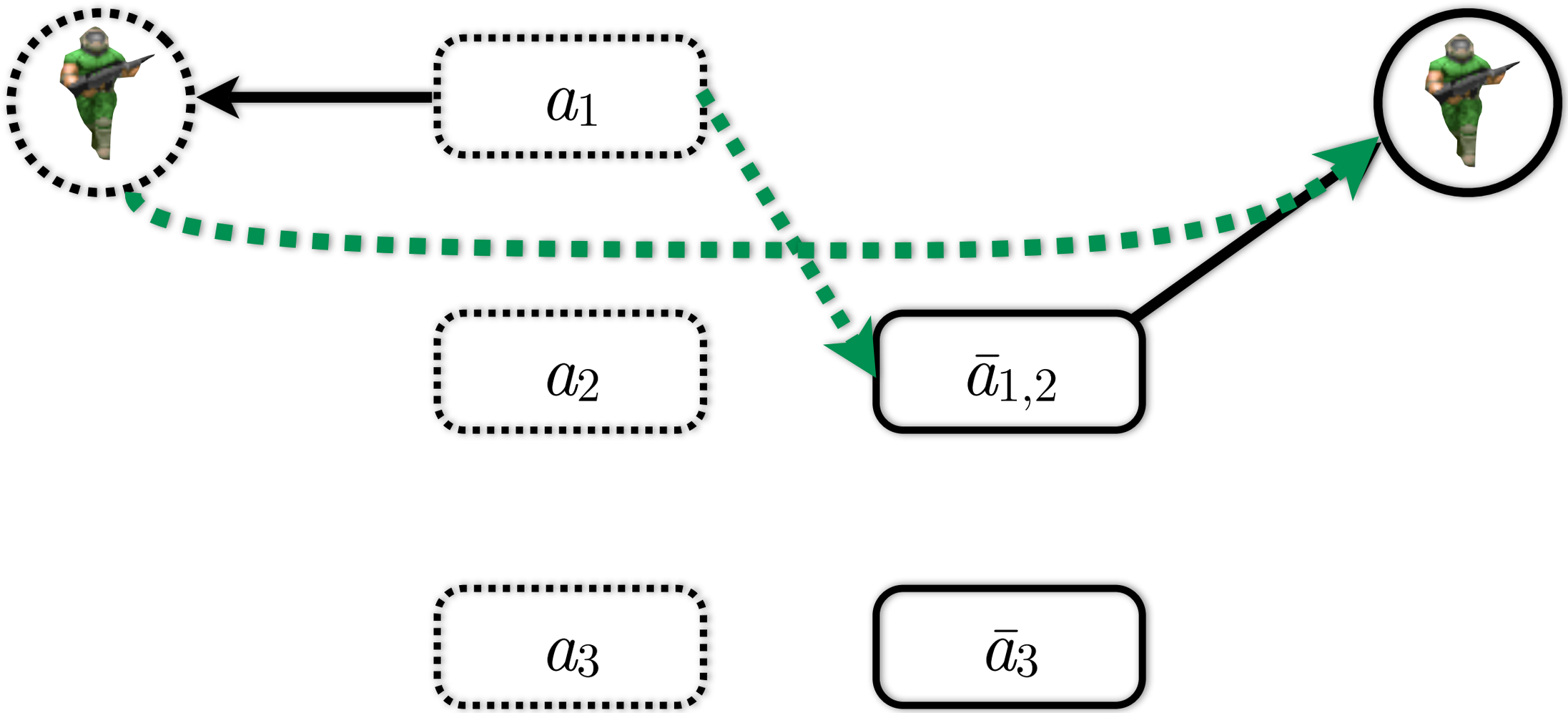
Abstract GC Illustrated



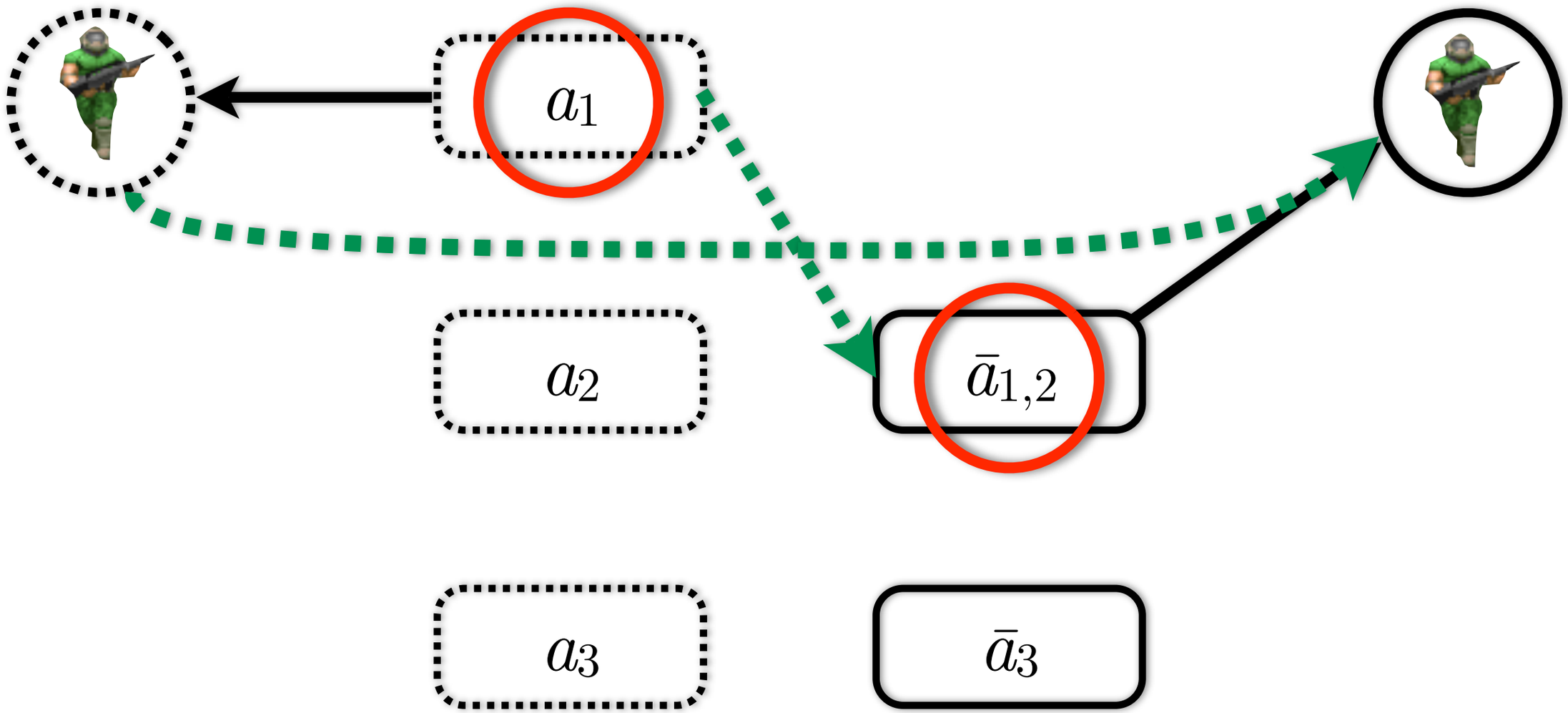
Abstract GC Illustrated



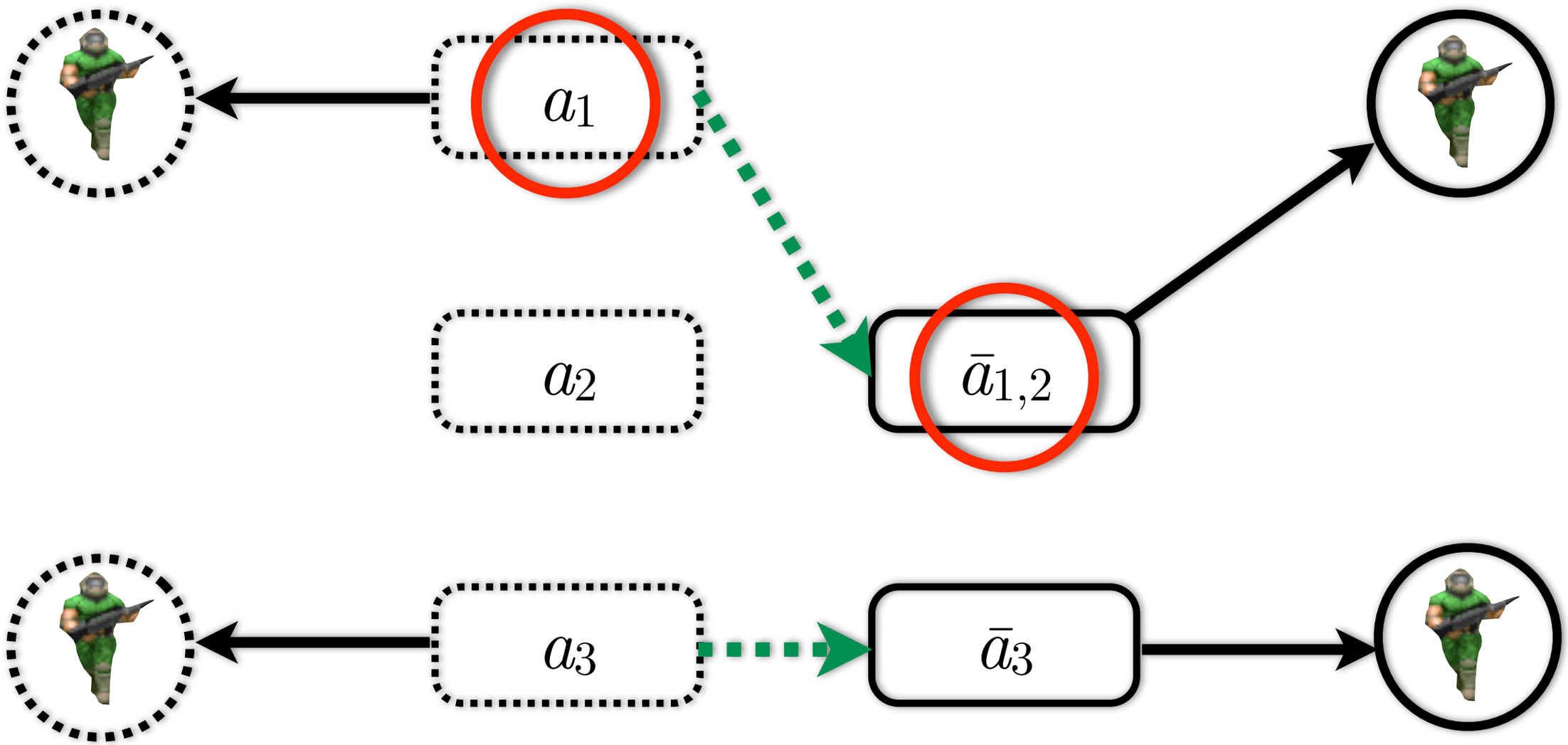
Abstract GC Illustrated



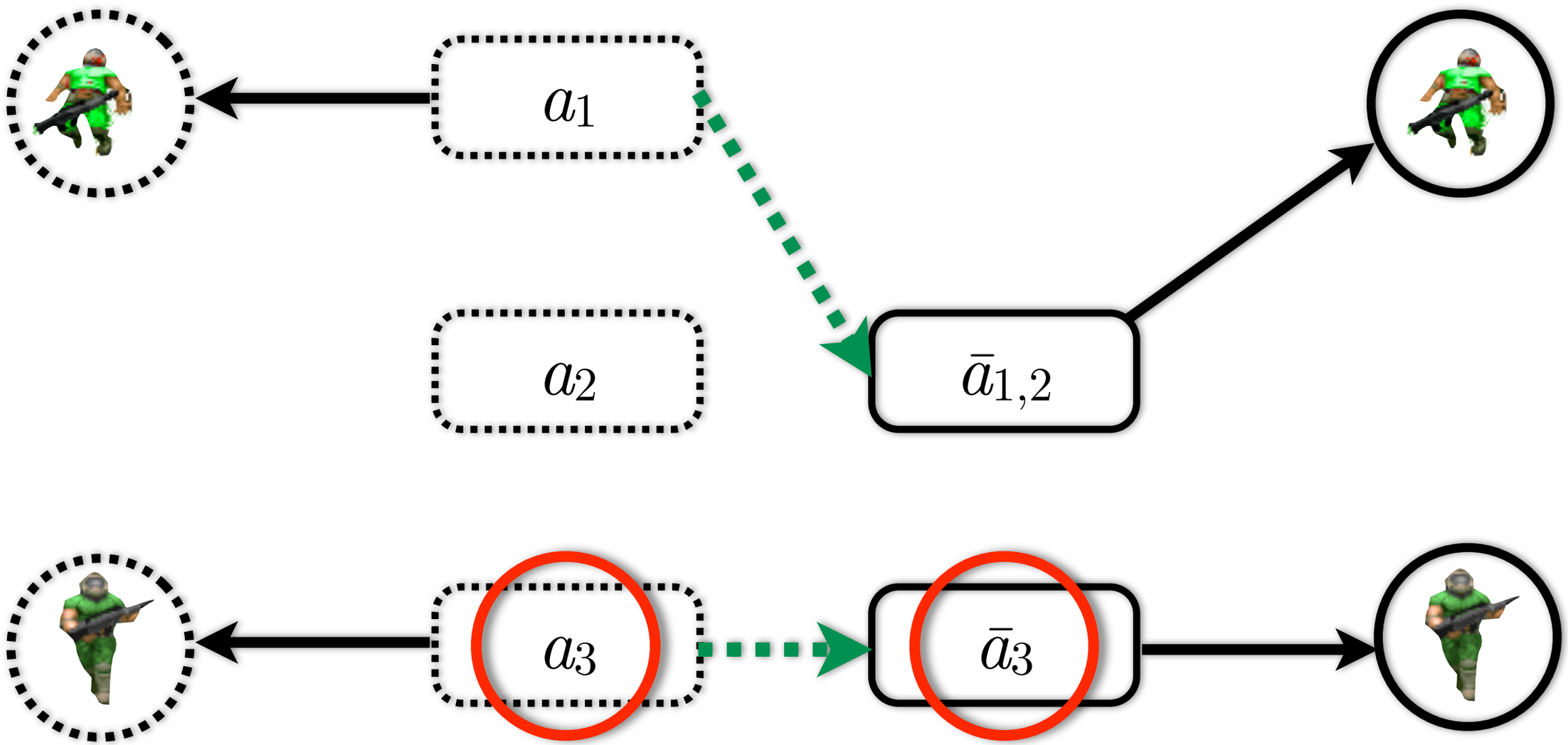
Abstract GC Illustrated



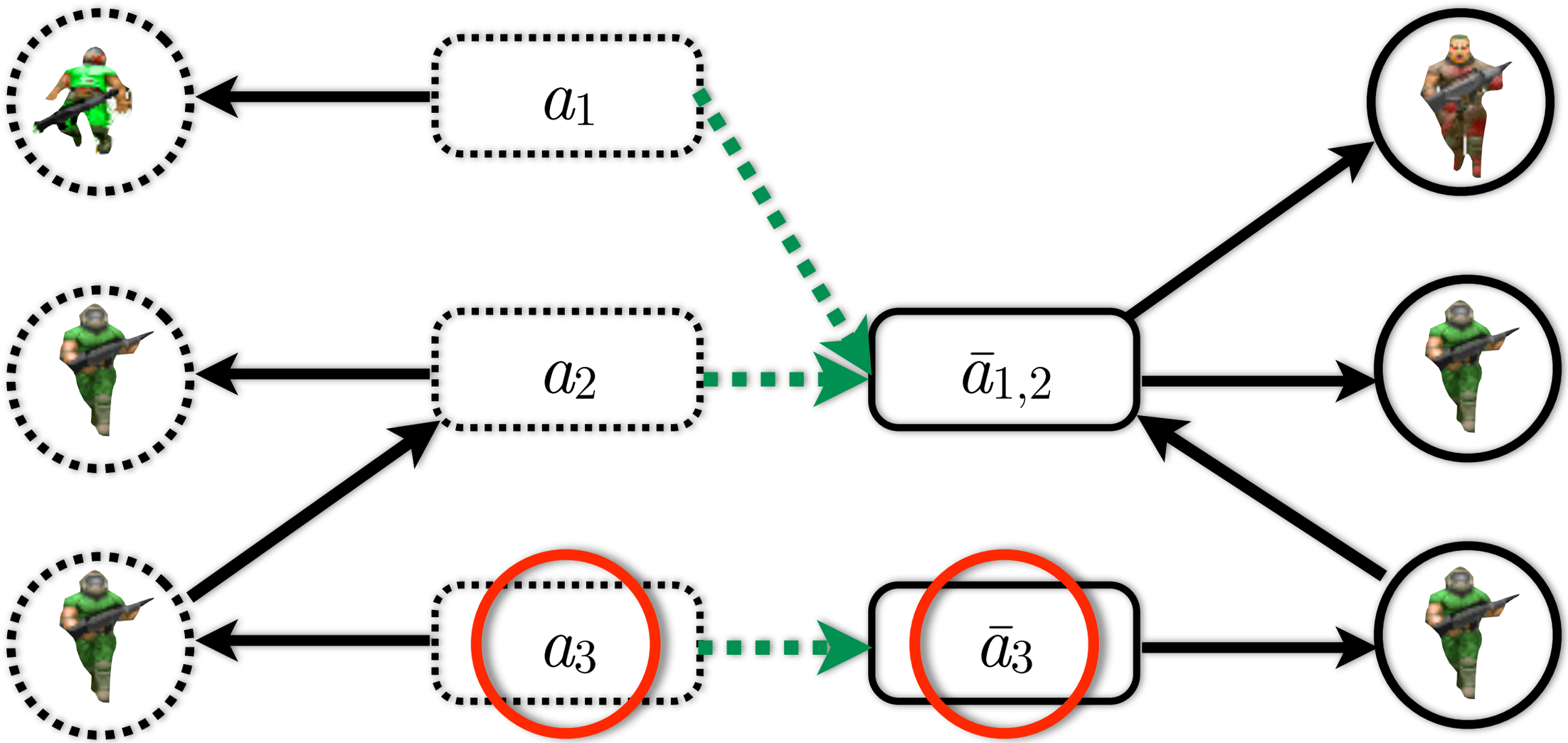
Abstract GC Illustrated



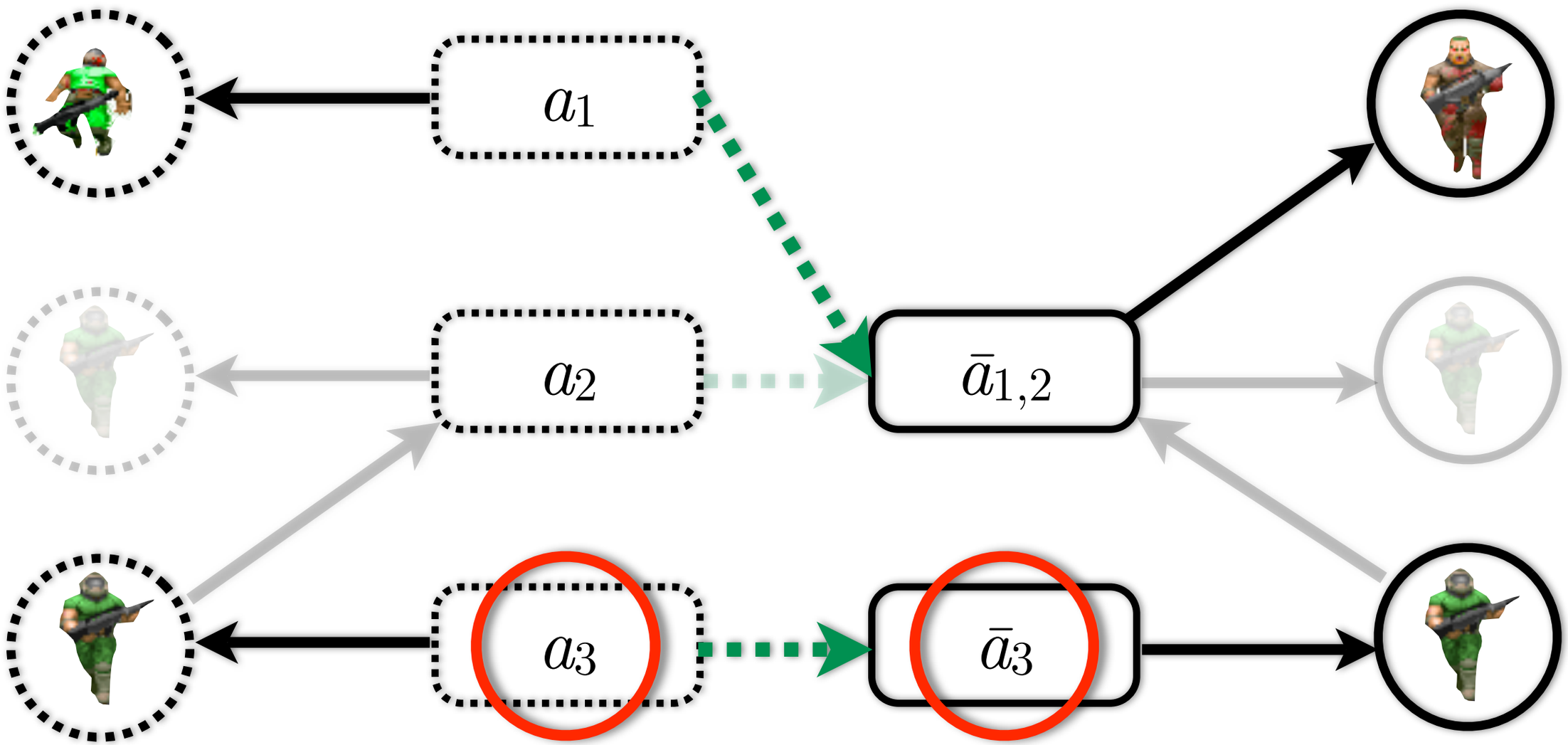
Abstract GC Illustrated



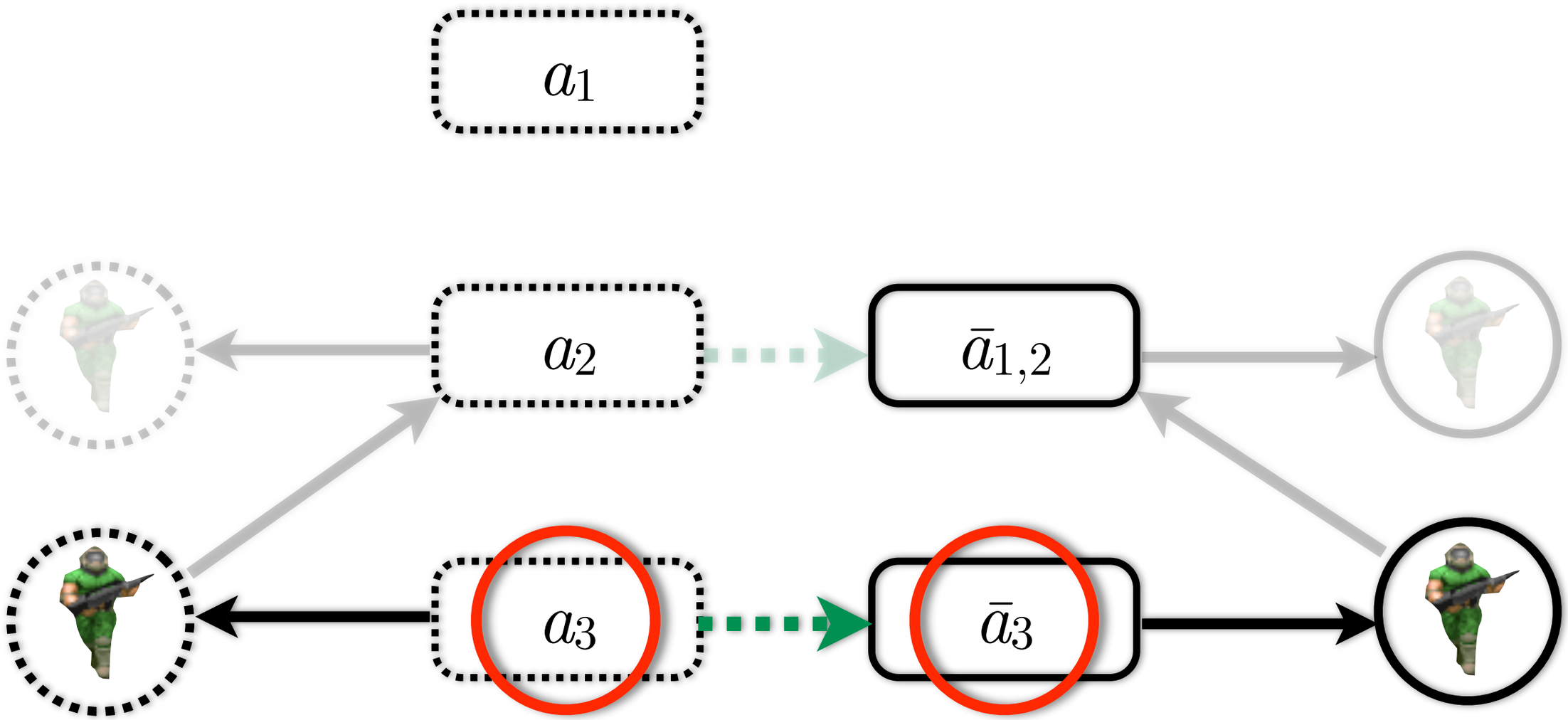
Abstract GC Illustrated



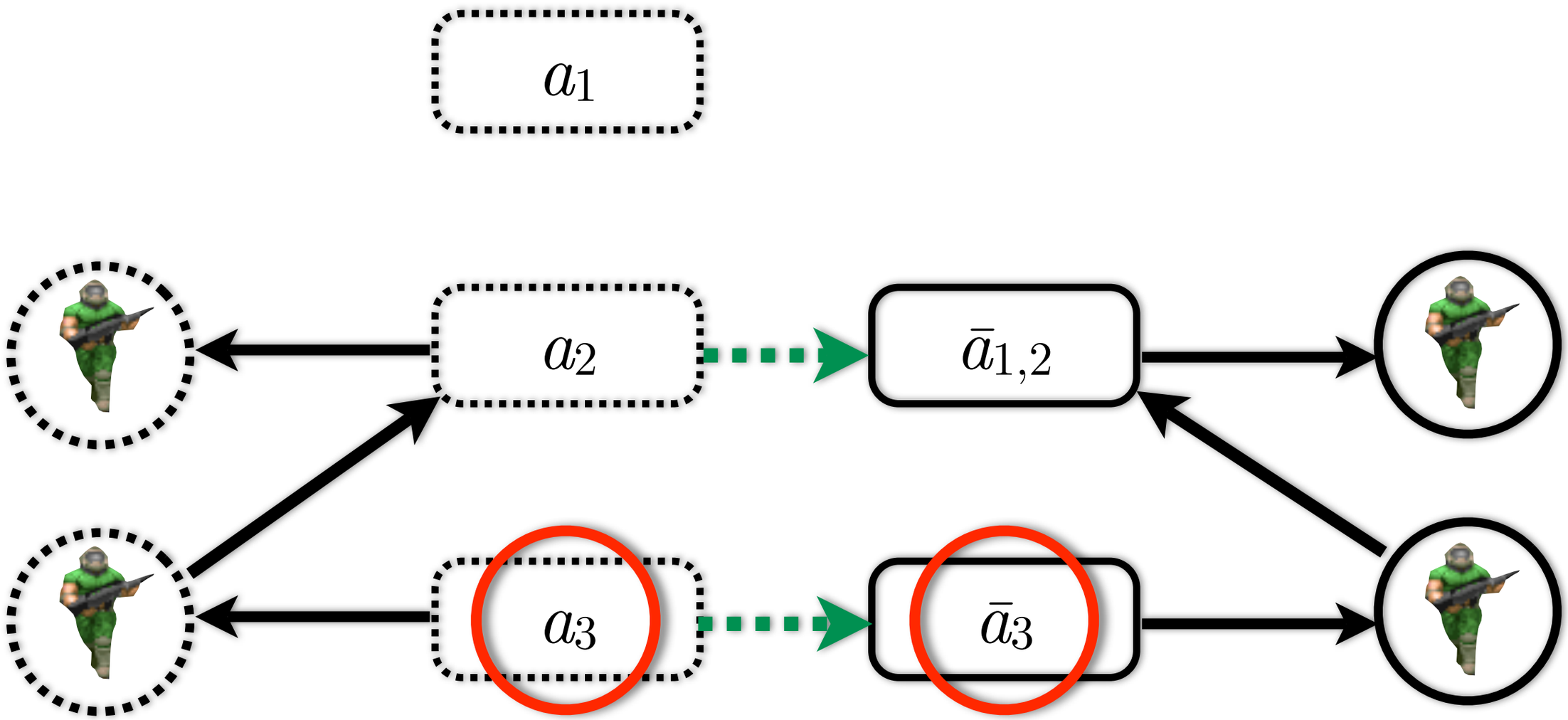
Abstract GC Illustrated



Abstract GC Illustrated

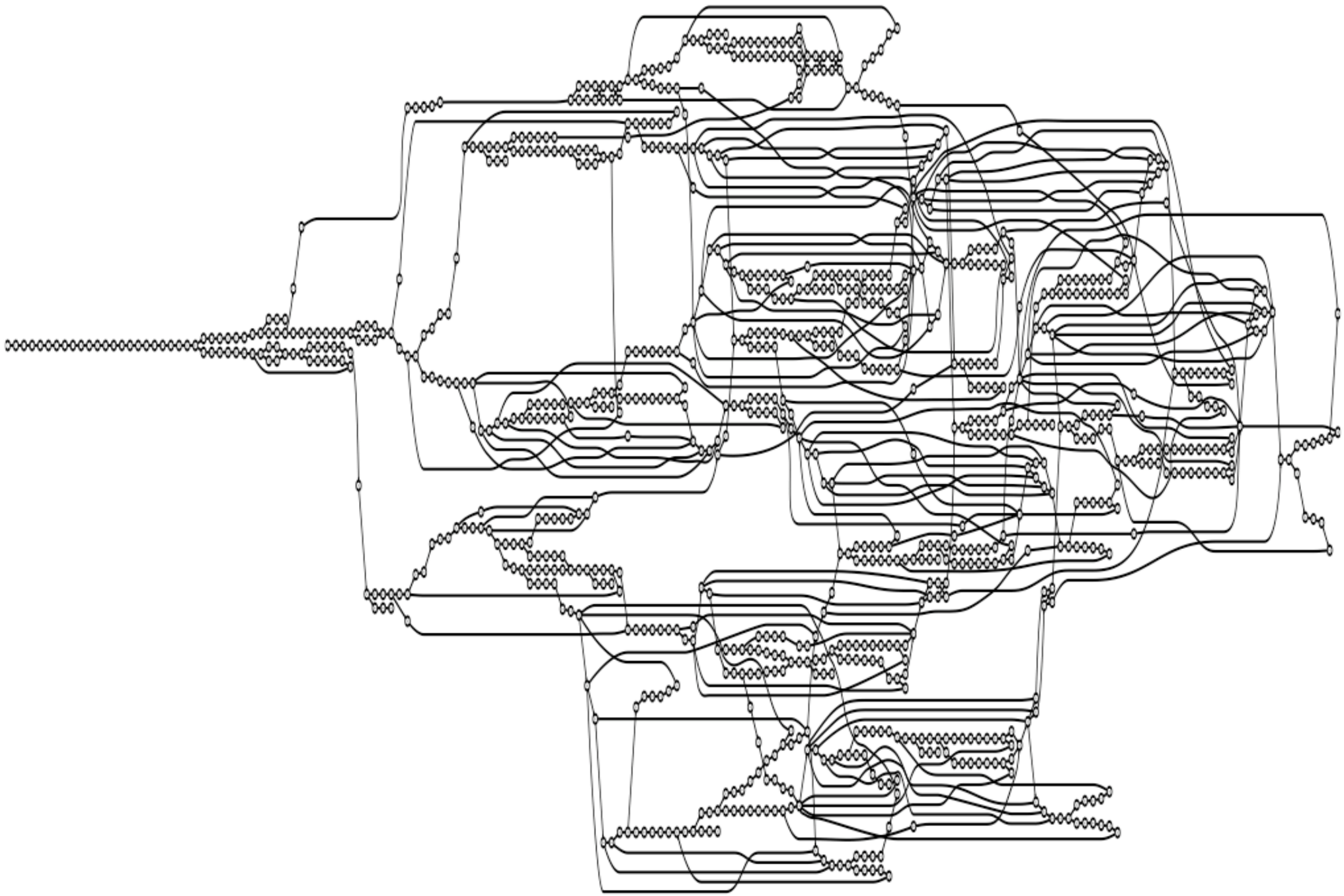


Abstract GC Illustrated

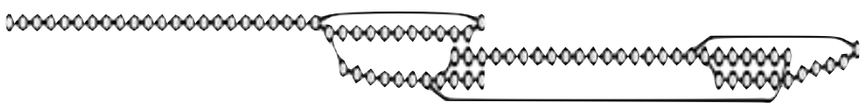


Effects of abstract GC

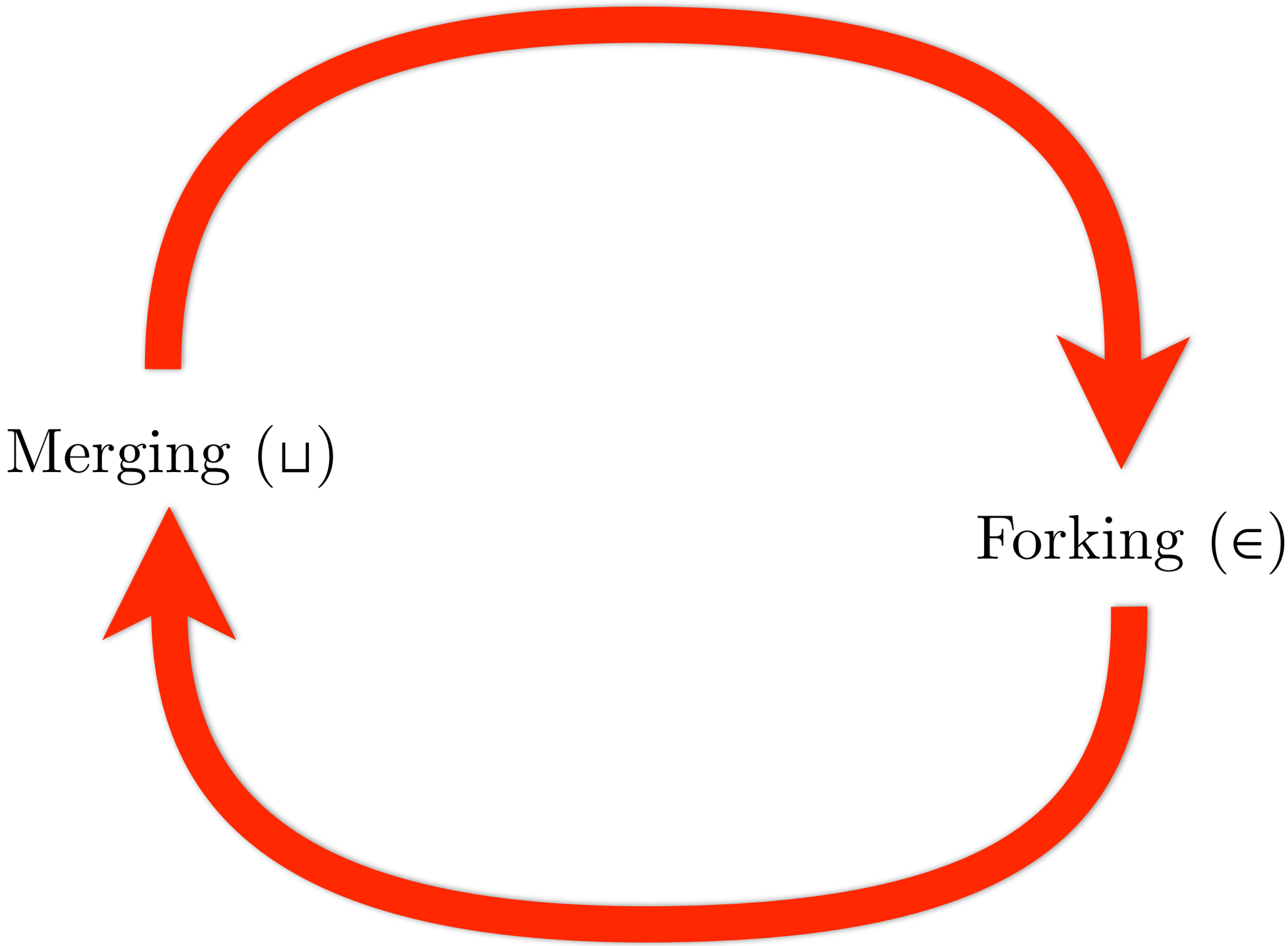
Effects of abstract GC



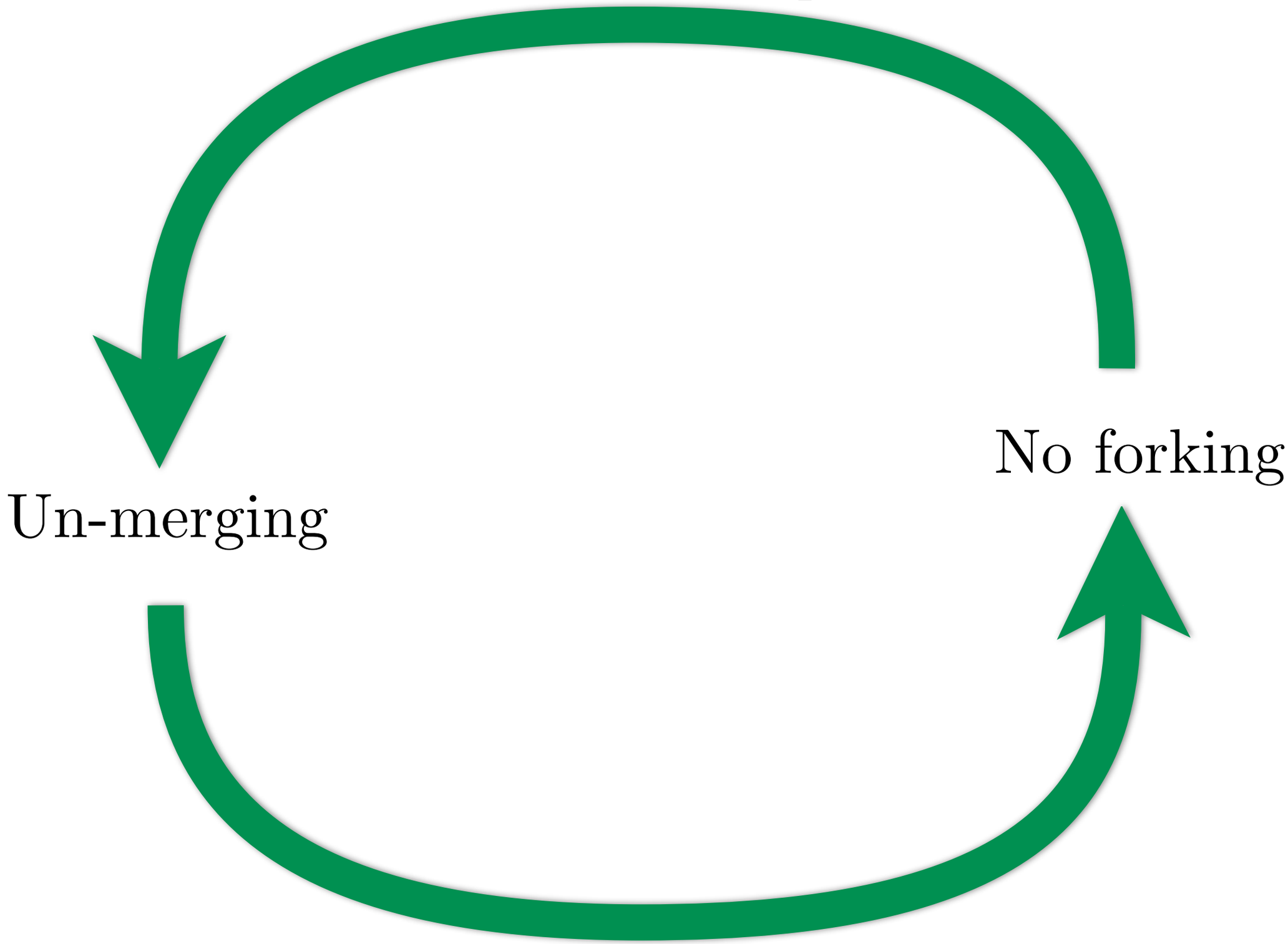
Effects of abstract GC



Vicious cyle



Virtuous cycle



Impact

- Orders-of-magnitude improvements

Implementing abstract garbage collection

The GC function

$$\hat{\Gamma} : \hat{\Sigma} \rightarrow \hat{\Sigma}$$

$$\hat{\Gamma}(\hat{\varsigma}) = (call, \hat{\beta}, \hat{\sigma} | \hat{\mathcal{R}}(\hat{\varsigma}), \hat{t})$$

where $\hat{\varsigma} = (call, \hat{\beta}, \hat{\sigma}, \hat{t})$

Reachable addresses

$$\hat{\mathcal{R}} : \hat{\Sigma} \to \mathcal{P}\left(\widehat{Addr}\right)$$

$$\hat{\mathcal{R}}(\hat{\varsigma}) = \left\{ \hat{a} : \hat{a}_0 \in Root(\hat{\varsigma}) \text{ and } \hat{a}_0 \overset{\hat{\sigma}_{\hat{\varsigma}}}{\rightsquigarrow} \hat{a} \right\}$$

Finding the root set

$$Root : \hat{\Sigma} \rightarrow \mathcal{P}\left(\widehat{Addr}\right)$$

$$Root(call, \hat{\beta}, \hat{\sigma}, \hat{t}) = range(\hat{\beta})$$

Reachability relation

Reflexive, transitive relation such that:

$$\frac{(lam, \hat{\beta}) \in \hat{\sigma}(\hat{a}) \qquad \hat{a}' \in range(\hat{\beta})}{\hat{a} \overset{\hat{\sigma}}{\rightsquigarrow} \hat{a}'}$$

Example: Without GC

```
let id = λx.x
    a  = id 3
    b  = id 4
in b
```

Example: Without GC

```
let id = λx.x
         ---------
         id 3
    a  = id 3
    b  = id 4
in b
```

$\hat{\sigma}(\mathtt{x}) = \{\}$

Example: Without GC

```
let id = λx.x
    a  = id 3
    b  = id 4
in b
```

$\hat{\sigma}(\mathtt{x}) = \{\}$

$\hat{\sigma}(\mathtt{x}) = \{3\}$

Example: Without GC

```
let id = λx.x
    a  = id 3
    b  = id 4
in b
```

$\hat{\sigma}(\mathtt{x}) = \{\}$

$\hat{\sigma}(\mathtt{x}) = \{3\}$

$\hat{\sigma}(\mathtt{x}) = \{3, 4\}$

Example: With GC

```
let id = λx.x
    a  = id 3
    b  = id 4
in b
```

Example: With GC

```
let id = λx.x
         ---------
         id 3
    a  = id 3
    b  = id 4
in b
```

$\hat{\sigma}(\mathtt{x}) = \{\}$

Example: With GC

```
let id = λx.x
    a  = id 3
    b  = id 4
in b
```

$\hat{\sigma}(\mathtt{x}) = \{\}$

$\hat{\sigma}(\mathtt{x}) = \{\}$

$\hat{\sigma}(\mathtt{a}) = \{3\}$

Example: With GC

```
let id = λx.x
    a  = id 3
    b  = id 4
in b
```

$\hat{\sigma}(\mathtt{x}) = \{\}$

$\hat{\sigma}(\mathtt{x}) = \{\}$

$\hat{\sigma}(\mathtt{a}) = \{3\}$

$\hat{\sigma}(\mathtt{x}) = \{\}$

$\hat{\sigma}(\mathtt{b}) = \{4\}$

Applications

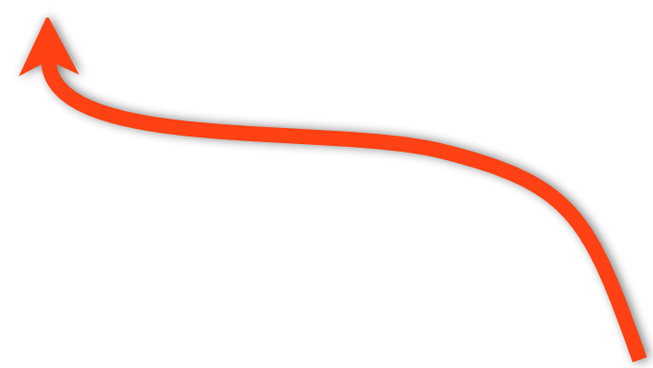
Inlining

Flow-based inlining

```
(f x)
```

Flow-based inlining

```
(f x)
```



```
(λ (z) z)
```

Flow-based inlining

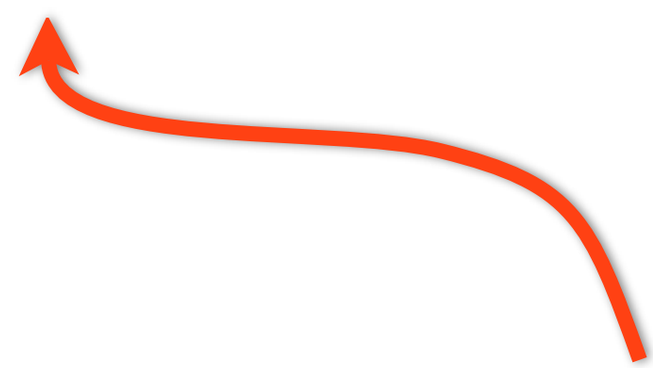
```
((λ (z) z) x)
```

Flow-based inlining

```
(f x)
```

Flow-based inlining

```
(f x)
```



```
(λ (z) y)
```

Flow-based inlining

```
((λ (z) y) x)
```

Flow-based inlining

Environment scenarios

Environment scenarios

- What if the free variables aren't in scope?

Environment scenarios

- What if the free variables aren't in scope?
- What if the free variables **are** in scope?

Pathological example

```
(let ((f (λ (x h) (if (zero? x)
                      (h)
                      (λ () x)))))
  (f 0 (f 3 #f)))
```

Pathological example

```
(let ((f (λ (x h) (if (zero? x)
                      (h)
                      (λ () x)))))
  (f 0 (f 3 #f)))
```

Pathological example

```
(let ((f (λ (x h) (if (zero? x)
                      (h)
                      (λ () x)))))
  (f 0 (f 3 #f)))
```

Pathological example

```
(let ((f (λ (x h) (if (zero? x)
                      (h)
                      (λ () x)))))
  (f 0 (f 3 #f)))
```

Pathological example

```
(let ((f (λ (x h) (if (zero? x)
                      (h)
                      (λ () x)))))
  (f 0 (f 3 #f)))
```

Pathological example

```
(let ((f (λ (x h) (if (zero? x)
                      (h)
                      (λ () x)))))
  (f 0 (f 3 #f)))
```

```
(λ () 3)
```

Pathological example

```
(let ((f (λ (x h) (if (zero? x)
                      (h)
                      (λ () x)))))
  (f 0 (f 3 #f)))
      (λ () 3)
```

Pathological example

```
(let ((f (λ (x h) (if (zero? x)
                      (h)
                      (λ () x)))))
  (f 0 (f 3 #f)))
      (λ () 3)
```

Pathological example

```
(let ((f (λ (x h) (if (zero? x)
                      (h)
                      (λ () x)))))
  (f 0 (f 3 #f)))
```

```
(λ () 3)
```

Pathological example

```
(let ((f (λ (x h) (if (zero? x)
                      ((λ () x))
                      (λ () x)))))
  (f 0 (f 3 #f)))
  (λ () 3)
```

Pathological example

```
(let ((f (λ (x h) (if (zero? x)
                      ((λ () 0))
                      (λ () x)))))
  (f 0 (f 3 #f)))
  (λ () 3)
```

Environment analysis

A little lesson

- My first solution: ΔCFA (Might & Shivers, 2006)
- Hallmarks of an elegant result, good dissertation
- Poor results in practice: explosions inevitable
- 2.5 years invested with ΔCFA -- a wrong turn

It’s never too late to stop making a bad decision.

</rant>

Environment problem

Environment problem

- Are two environments equivalent?

Environment problem

- Are two environments equivalent?
- What is **equivalent**?

Environment problem

- Are two environments equivalent?
- What is **equivalent**?
- **Which** two environments?

Equivalent environments

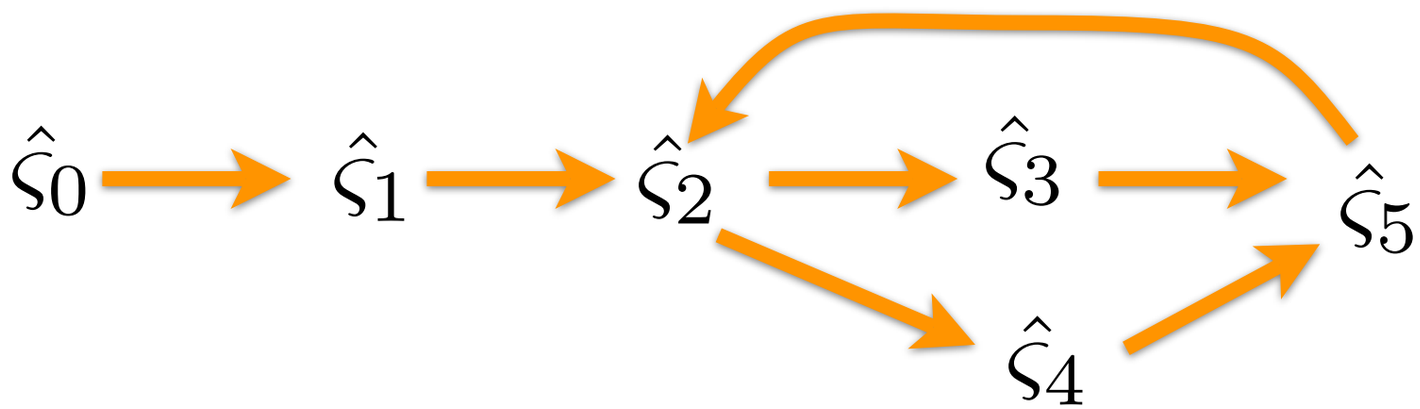
Environments ρ and ρ' are **equivalent**
over variables $v_1, \ldots, v_n$
if $\rho(v_i) = \rho'(v_i)$.

Equivalent environments

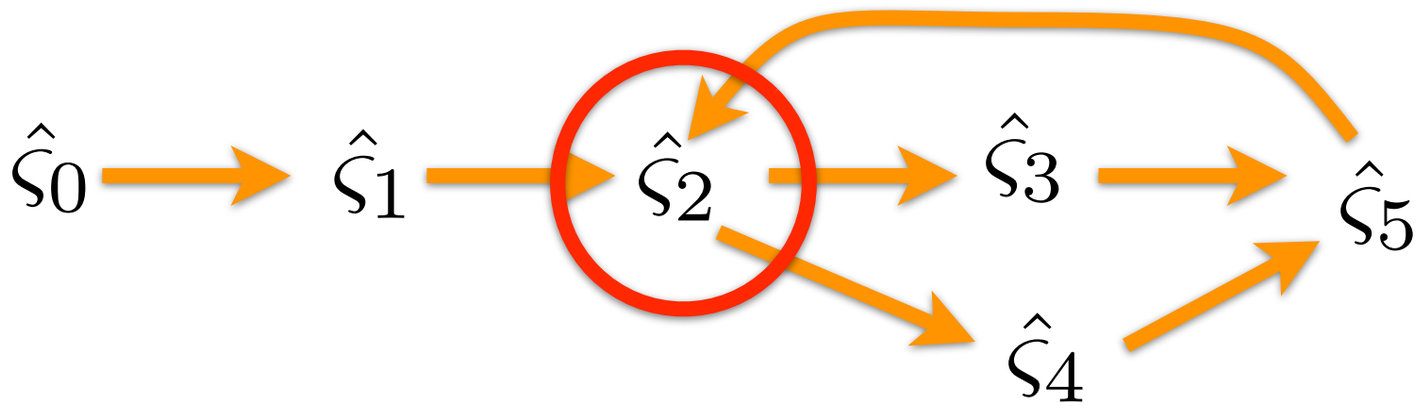
Environments β and β' are **equivalent** over variables $v_1, \ldots, v_n$ if $\beta(v_i) = \beta'(v_i)$.

Which environments

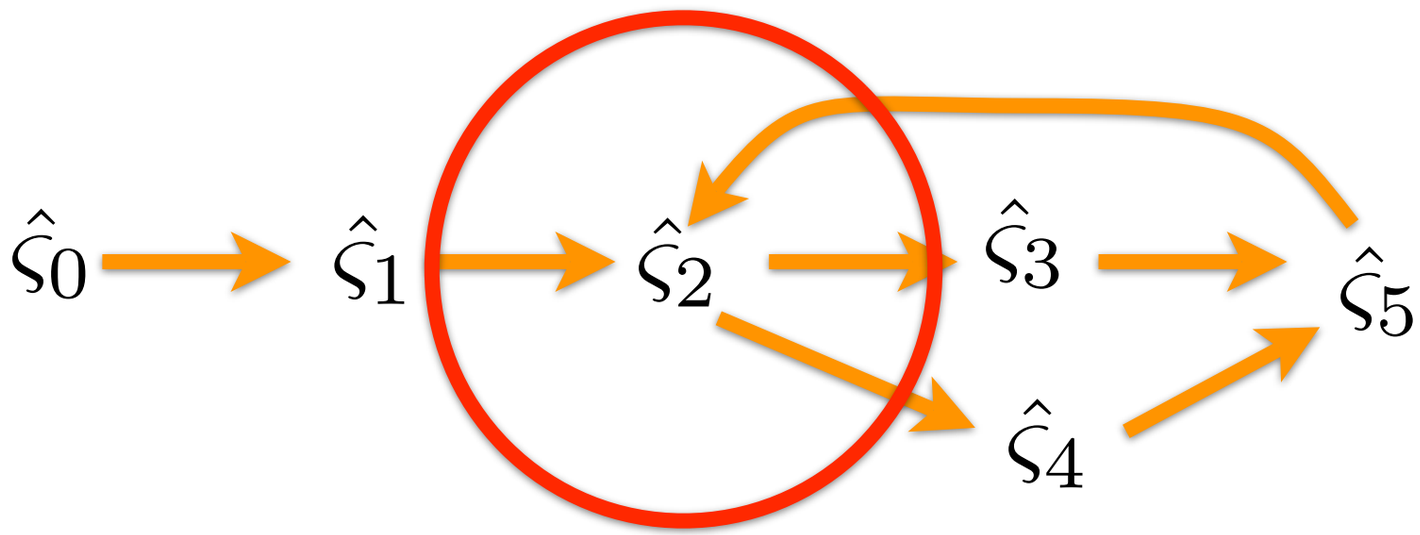
Which environments

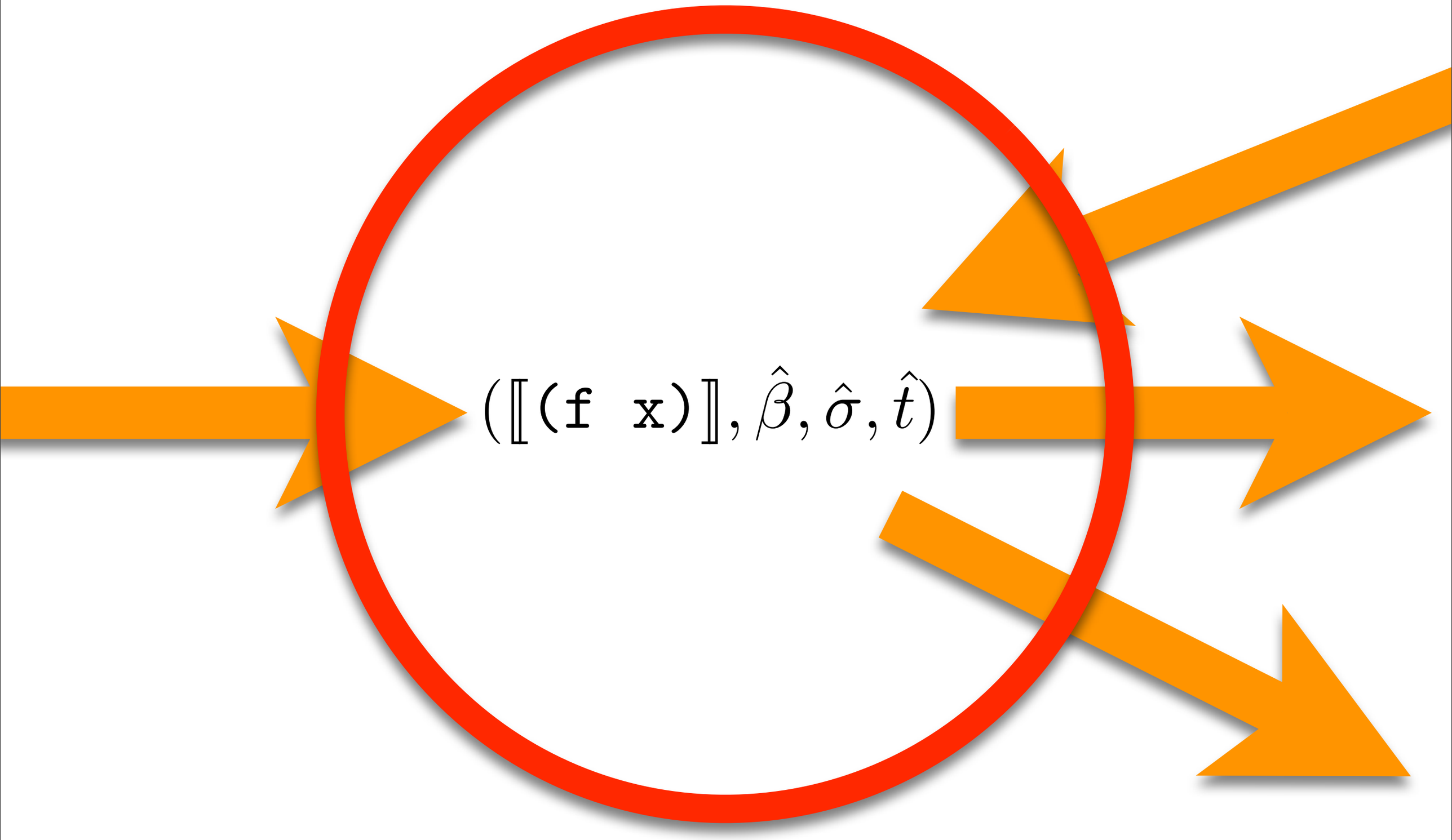


Which environments



Which environments




$([\![\texttt{(f x)}]\!], \hat{\beta}, \hat{\sigma}, \hat{t})$

$$(\llbracket \texttt{(}\lambda\ \texttt{(x) z)} \rrbracket, \hat{\beta}')$$

$$\xrightarrow{\texttt{f}}$$

$$(\llbracket \texttt{(f x)} \rrbracket, \hat{\beta}, \hat{\sigma}, \hat{t})$$

Are the counterparts to $\hat{\beta}$ and $\hat{\beta}'$ equivalent?

$$(\llbracket \texttt{(}\lambda\ \texttt{(x) z)} \rrbracket, \hat{\beta}')$$

$$\uparrow \texttt{f}$$

$$(\llbracket \texttt{(f x)} \rrbracket, \hat{\beta}, \hat{\sigma}, \hat{t})$$

Are the counterparts to $\hat{\beta}$ and $\hat{\beta}'$ equivalent?

$$(\llbracket \texttt{(}\lambda\ \texttt{(x) z)} \rrbracket, \hat{\beta}')$$

$$\uparrow \texttt{f}$$

$$(\llbracket \texttt{(f x)} \rrbracket, \hat{\beta}, \hat{\sigma}, \hat{t})$$

$$\hat{\beta}\llbracket \texttt{z} \rrbracket = \hat{a}$$

$$\hat{\beta}'\llbracket \texttt{z} \rrbracket = \hat{a}$$

Are the counterparts to $\hat{\beta}$ and $\hat{\beta}'$ equivalent?

$\hat{a} \equiv \{a_1\}$?

$$(\llbracket \texttt{(}\lambda\ \texttt{(x) z)} \rrbracket, \hat{\beta}')$$

$$\uparrow \texttt{f}$$

$$(\llbracket \texttt{(f x)} \rrbracket, \hat{\beta}, \hat{\sigma}, \hat{t})$$

$$\hat{\beta}\llbracket \texttt{z} \rrbracket = \hat{a}$$

$$\hat{\beta}'\llbracket \texttt{z} \rrbracket = \hat{a}$$

Are the counterparts to $\hat{\beta}$ and $\hat{\beta}'$ equivalent?

$\hat{a} \equiv \{a_1\}$?

$\hat{a} \equiv \{a_1, a_2\}$?

$$(\llbracket \texttt{(}\lambda\ \texttt{(x) z)} \rrbracket, \hat{\beta}')$$

$$\uparrow \texttt{f}$$

$$(\llbracket \texttt{(f x)} \rrbracket, \hat{\beta}, \hat{\sigma}, \hat{t})$$

$$\hat{\beta}\llbracket \texttt{z} \rrbracket = \hat{a}$$

$$\hat{\beta}'\llbracket \texttt{z} \rrbracket = \hat{a}$$

The counting principle

If $\{x\} = \{y\}$, then $x = y$.

Abstract counting

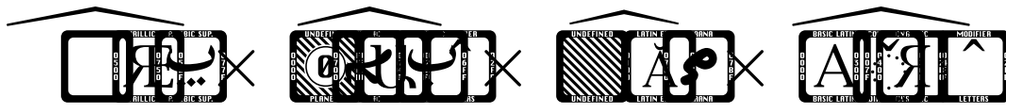
- Count abstract addresses as allocated
- When count is one, only one concrete

Counting machinery

$$\hat{c} \in \widehat{Count} = \{0, 1, \infty\}$$

$$\hat{\mu} \in \widehat{Max} = \widehat{Addr} \rightarrow \widehat{Count}$$

Abstract state-space

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

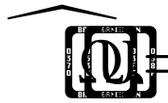
$\hat{\beta} \in$ $=$ $\mathsf{Var} \rightharpoonup$

$\hat{\sigma} \in$ $=$ $\rightarrow \hat{D}$

$\hat{d} \in \hat{D} = \mathcal{P}\left($ $\right)$

$=$ $=$

$=$ $=$

$=$  $=$ $\mathsf{Lam} \times$

$\hat{a} \in$ is a **finite** set of addresses

$\hat{t} \in$ is a **finite** set of time-stamps

Abstract state-space

$\hat{\varsigma} \in \hat{\Sigma} =$ Call $\times$ [illegible] $\times$ [illegible] $\times$ [illegible] $\times$ [illegible]

$\hat{\beta} \in$ [illegible] $=$ Var $\rightharpoonup$ [illegible]

$\hat{\sigma} \in$ [illegible] $=$ [illegible] $\rightarrow \hat{D}$

$\hat{d} \in \hat{D} = \mathcal{P}\left(\text{[illegible]}\right)$

[illegible] $=$ [illegible] $=$ [illegible]

[illegible] $=$ [illegible] $=$ [illegible]

[illegible] $=$ [illegible] $=$ Lam $\times$ [illegible]

$\hat{a} \in$ [illegible] is a **finite** set of addresses

$\hat{t} \in$ [illegible] is a **finite** set of time-stamps

Abstract state-space

When $call = [\![(f\ e_1 \ldots e_n)]\!]$:

$$(call, \hat{\beta}, \hat{\sigma}, \hat{\mu}, \hat{t}) \rightsquigarrow (call', \hat{\beta}'', \hat{\sigma}', \hat{\mu}', \hat{t}'), \text{ where}$$

$$(lam, \hat{\beta}') \in \hat{\mathcal{E}}(f, \hat{\beta}, \hat{\sigma})$$

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

$$lam = [\![(\lambda\ (v_1 \ldots v_n)\ call')]\!]$$

$$\hat{t}' = \widehat{tick}(call, \hat{t})$$

$$\hat{a}_i = \widehat{alloc}(v_i, \hat{t}')$$

$$\hat{\beta}'' = \hat{\beta}'[v_i \mapsto \hat{a}_i]$$

$$\hat{\sigma}' = \hat{\sigma} \sqcup [\hat{a}_i \mapsto \hat{C}_i]$$

$$\hat{\mu}' = \hat{\mu}' \oplus [\hat{a}_i \mapsto 1]$$

Abstract state-space

When $call = [\![(f\ e_1 \ldots e_n)]\!]$:

$$(call, \hat{\beta}, \hat{\sigma}, \hat{\mu}, \hat{t}) \rightsquigarrow (call', \hat{\beta}'', \hat{\sigma}', \hat{\mu}', \hat{t}'), \text{ where}$$

$$(lam, \hat{\beta}') \in \hat{\mathcal{E}}(f, \hat{\beta}, \hat{\sigma})$$

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

$$lam = [\![(\lambda\ (v_1 \ldots v_n)\ call')]\!]$$

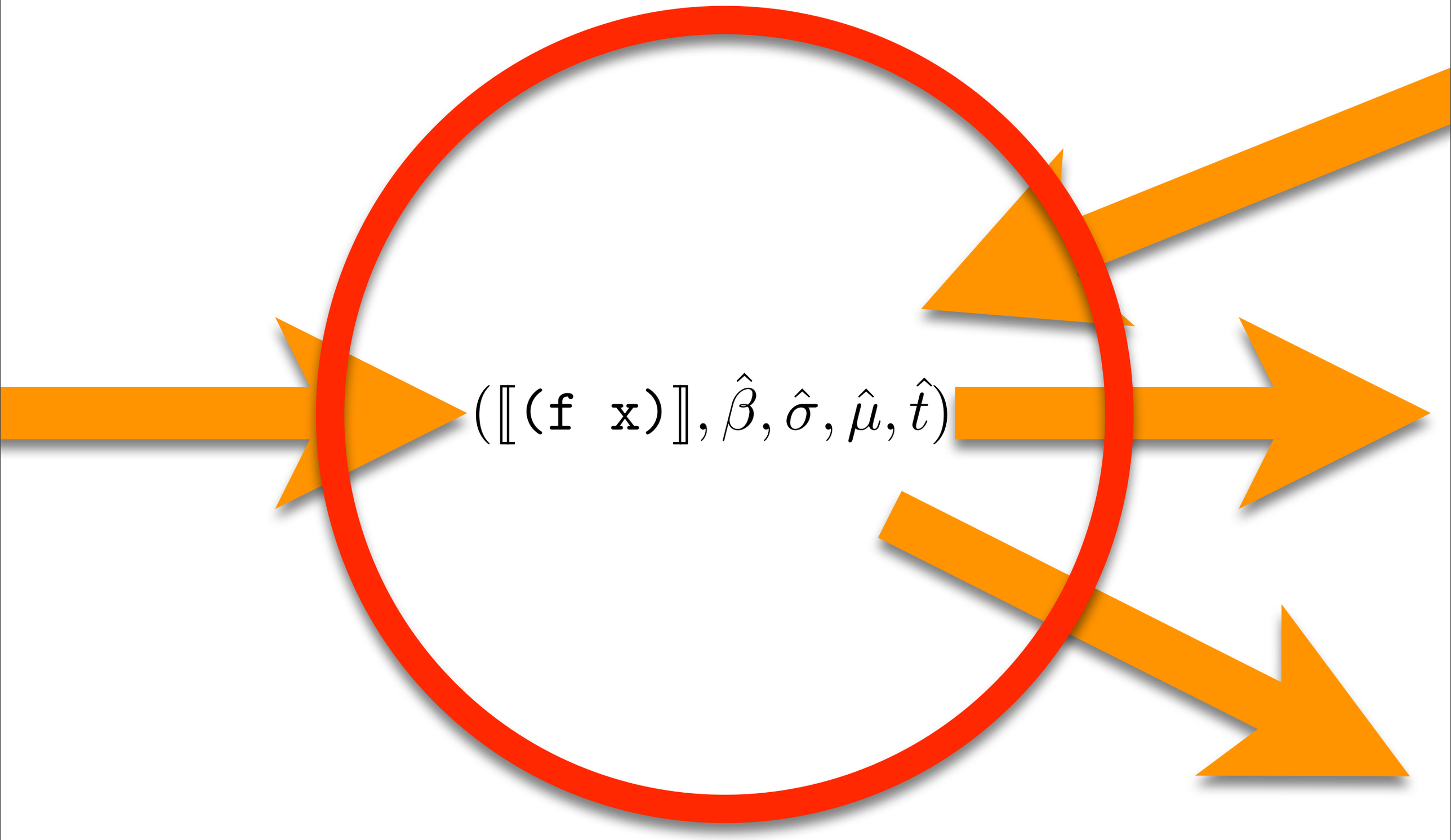
$$\hat{t}' = \widehat{tick}(call, \hat{t})$$

$$\hat{a}_i = \widehat{alloc}(v_i, \hat{t}')$$

$$\hat{\beta}'' = \hat{\beta}'[v_i \mapsto \hat{a}_i]$$

$$\hat{\sigma}' = \hat{\sigma} \sqcup [\hat{a}_i \mapsto \hat{C}_i]$$

$$\hat{\mu}' = \hat{\mu}' \oplus [\hat{a}_i \mapsto 1]$$


$([\![\texttt{(f x)}]\!], \hat{\beta}, \hat{\sigma}, \hat{\mu}, \hat{t})$

Are the counterparts to $\hat{\beta}$ and $\hat{\beta}'$ equivalent?

$$(\llbracket \texttt{(}\lambda\ \texttt{(x) z)} \rrbracket, \hat{\beta}')$$

$$\uparrow \texttt{f}$$

$$(\llbracket \texttt{(f x)} \rrbracket, \hat{\beta}, \hat{\sigma}, \hat{\mu}, \hat{t})$$

Are the counterparts to $\hat{\beta}$ and $\hat{\beta}'$ equivalent?

$$(\llbracket \texttt{(}\lambda\ \texttt{(x) z)} \rrbracket, \hat{\beta}')$$

$$\uparrow \texttt{f}$$

$$(\llbracket \texttt{(f x)} \rrbracket, \hat{\beta}, \hat{\sigma}, \hat{\mu}, \hat{t})$$

$$\hat{\beta}\llbracket \texttt{z} \rrbracket = \hat{a}$$

$$\hat{\beta}'\llbracket \texttt{z} \rrbracket = \hat{a}$$

Are the counterparts to $\hat{\beta}$ and $\hat{\beta}'$ equivalent?

$\hat{a} \equiv \{a_1\}$?

$$(\llbracket \texttt{(}\lambda\ \texttt{(x) z)} \rrbracket, \hat{\beta}')$$

$$\uparrow \texttt{f}$$

$$(\llbracket \texttt{(f x)} \rrbracket, \hat{\beta}, \hat{\sigma}, \hat{\mu}, \hat{t})$$

$$\hat{\beta}\llbracket \texttt{z} \rrbracket = \hat{a}$$

$$\hat{\beta}'\llbracket \texttt{z} \rrbracket = \hat{a}$$

Are the counterparts to $\hat{\beta}$ and $\hat{\beta}'$ equivalent?

$\hat{a} \equiv \{a_1\}$?

$\hat{a} \equiv \{a_1, a_2\}$?

$$(\llbracket \texttt{(}\lambda\ \texttt{(x) z)} \rrbracket, \hat{\beta}')$$

$$\xrightarrow{\texttt{f}}$$

$$(\llbracket \texttt{(f x)} \rrbracket, \hat{\beta}, \hat{\sigma}, \hat{\mu}, \hat{t})$$

$$\hat{\beta}\llbracket \texttt{z} \rrbracket = \hat{a}$$

$$\hat{\beta}'\llbracket \texttt{z} \rrbracket = \hat{a}$$

Are the counterparts to $\hat{\beta}$ and $\hat{\beta}'$ equivalent?

$\hat{a} \equiv \{a_1\}?$ $\quad \hat{\mu}(\hat{a}) = 1$ $\quad ([\![(\lambda\ \ (\mathtt{x})\ \ \mathtt{z})]\!], \hat{\beta}')$

$\hat{a} \equiv \{a_1, a_2\}?$

$\mathtt{f}$

$$([\![(\mathtt{f}\ \ \mathtt{x})]\!], \hat{\beta}, \hat{\sigma}, \hat{\mu}, \hat{t})$$

$$\hat{\beta}[\![\mathtt{z}]\!] = \hat{a}$$

$$\hat{\beta}'[\![\mathtt{z}]\!] = \hat{a}$$

Are the counterparts to $\hat{\beta}$ and $\hat{\beta}'$ equivalent?

$\hat{a} \equiv \{a_1\}?$ $\quad \hat{\mu}(\hat{a}) = 1$ $\quad (\llbracket \texttt{(}\lambda\ \texttt{(x) z)} \rrbracket, \hat{\beta}')$

$\hat{a} \equiv \{a_1, a_2\}?$ $\quad \hat{\mu}(\hat{a}) = \infty$

$\texttt{f}$

$$(\llbracket \texttt{(f x)} \rrbracket, \hat{\beta}, \hat{\sigma}, \hat{\mu}, \hat{t})$$

$$\hat{\beta}\llbracket \texttt{z} \rrbracket = \hat{a}$$

$$\hat{\beta}'\llbracket \texttt{z} \rrbracket = \hat{a}$$

More applications of counting

CPS with `set!-then`

$$v \in \mathsf{Var}$$

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

$$lam \in \mathsf{Lam} ::= (\lambda\ (v_1 \ldots v_n)\ call)$$

$$call \in \mathsf{Call} ::= (f\ e_1 \ldots e_n)$$

CPS with `set!-then`

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

Mutation w/o counting

When $call = [\![\texttt{(set!-then}\ v\ e\ call'\texttt{)}]\!]$:

$$(call, \beta, \sigma, t) \Rightarrow (call', \beta, \sigma', t'), \text{ where}$$
$$t' = tick(call, t)$$
$$d = \mathcal{E}(e, \beta, \sigma)$$
$$\sigma' = \sigma[\beta(v) \mapsto d]$$

When $call = [\![\texttt{(set!-then}\ v\ e\ call'\texttt{)}]\!]$:

$$(call, \hat{\beta}, \hat{\sigma}, \hat{t}) \rightsquigarrow (call', \hat{\beta}, \hat{\sigma}', \hat{t}'), \text{ where}$$
$$\hat{t}' = \widehat{tick}(call, \hat{t})$$
$$\hat{d} = \hat{\mathcal{E}}(e, \hat{\beta}, \hat{\sigma})$$
$$\hat{\sigma}' = \hat{\sigma} \sqcup [\hat{\beta}(v) \mapsto \hat{d}]$$

Strong update rule

When $call = [\![\texttt{(set!-then}\ v\ e\ call\texttt{)}]\!]$:

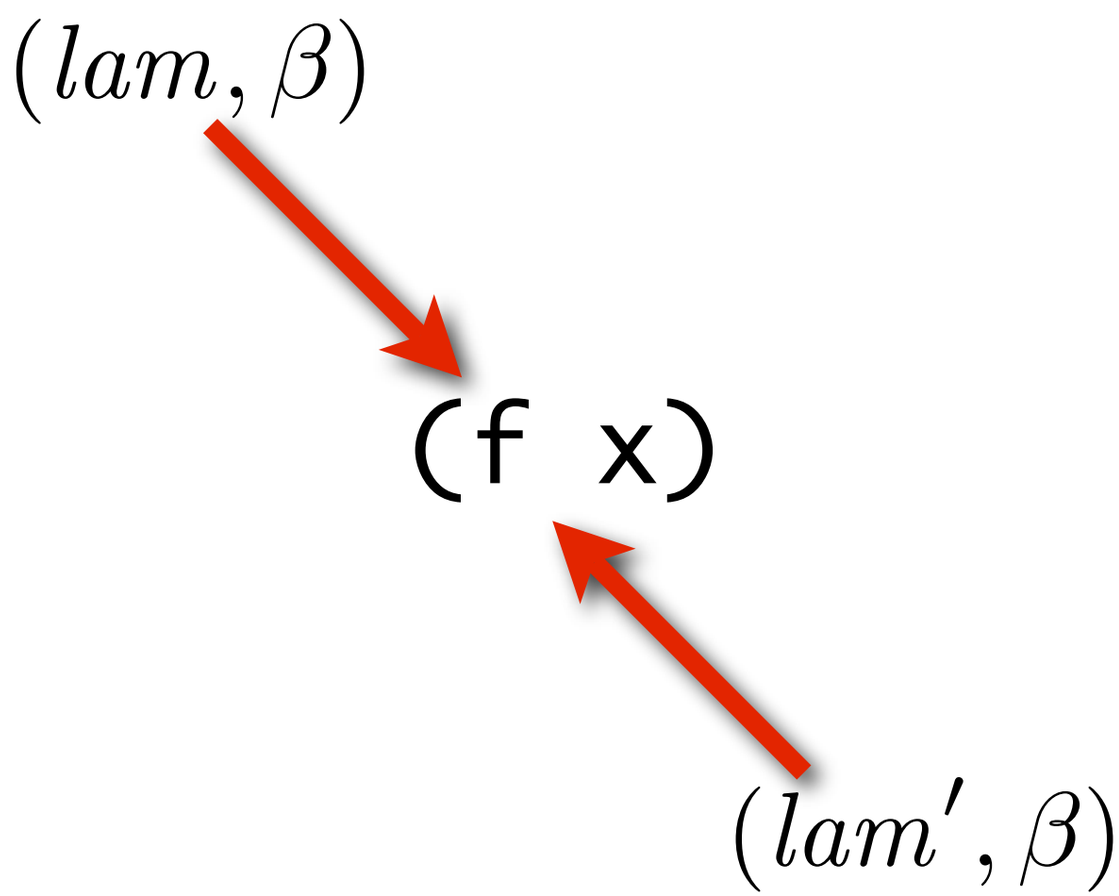
$([\![\texttt{(set!-then}\ v\ e\ call\texttt{)}]\!], \hat{\beta}, \hat{\sigma}, \hat{\mu}, \hat{t}) \rightsquigarrow (call, \hat{\beta}, \hat{\sigma}', \hat{\mu}, \hat{t}')$, where

$$\hat{t}' = \widehat{tick}(call, \hat{t})$$

$$\hat{d} = \hat{\mathcal{E}}(e, \hat{\beta}, \hat{\sigma})$$

$$\hat{\sigma}' = \begin{cases} \hat{\sigma}[\hat{\beta}(v) \mapsto \hat{d}] & \hat{\mu}(\hat{\beta}(v)) = 1 \\ \hat{\sigma} \sqcup [\hat{\beta}(v) \mapsto \hat{d}] & \text{otherwise.} \end{cases}$$

Lightweight closures



Globalization

- If a variable has max count of one,
- then it may be turned into a global.

Sparse interprocedural constant propagation

CPS with numerals and conditionals

$$v \in \mathsf{Var}$$

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

$$lam \in \mathsf{Lam} ::= (\lambda\ (v_1 \ldots v_n)\ call)$$

$$call \in \mathsf{Call} ::= (f\ e_1 \ldots e_n)$$

CPS with numerals and conditionals

$$v \in \mathsf{Var}$$

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

$$lam \in \mathsf{Lam} ::= (\lambda\ (v_1 \ldots v_n)\ call)$$

$$call \in \mathsf{Call} ::= (f\ e_1 \ldots e_n)$$

CPS with numerals and conditionals

$$
\begin{aligned}
v &\in \mathsf{Var} \\
f, e &\in \mathsf{Exp} = \mathsf{Var} + \mathsf{Lam} + \mathsf{Num} \\
lam &\in \mathsf{Lam} ::= \texttt{(}\lambda\ \texttt{(}v_1 \ldots v_n\texttt{)}\ call\texttt{)} \\
call &\in \mathsf{Call} ::= \texttt{(}f\ e_1 \ldots e_n\texttt{)} \\
&\qquad\quad \mid\ \texttt{(}\texttt{if}\ e\ call_1\ call_2\texttt{)}
\end{aligned}
$$

k-CFA with constants

$$\hat{\varsigma} \in \hat{\Sigma} = \mathsf{Call} \times \widehat{BEnv} \times \widehat{Store} \times \widehat{Time}$$

$$\hat{\beta} \in \widehat{BEnv} = \mathsf{Var} \rightharpoonup \widehat{Addr}$$

$$\hat{\sigma} \in \widehat{Store} = \widehat{Addr} \to \hat{D}$$

$$\hat{d} \in \hat{D} = \mathcal{P}\left(\widehat{Val}\right)$$

$$\widehat{val} \in \widehat{Val} = \widehat{Proc} + \widehat{Bas}$$

$$\widehat{proc} \in \widehat{Proc} = \widehat{Clo}$$

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

$$\widehat{bas} \in \widehat{Bas} = \mathsf{Num}_{\top}^{\top}$$

$\hat{a} \in \widehat{Addr}$ is a **finite** set of addresses

$\hat{t} \in \widehat{Time}$ is a **finite** set of time-stamps

k-CFA with constants

$$\hat{\varsigma} \in \hat{\Sigma} = \mathsf{Call} \times \widehat{BEnv} \times \widehat{Store} \times \widehat{Time}$$

$$\hat{\beta} \in \widehat{BEnv} = \mathsf{Var} \rightharpoonup \widehat{Addr}$$

$$\hat{\sigma} \in \widehat{Store} = \widehat{Addr} \to \hat{D}$$

$$\hat{d} \in \hat{D} = \mathcal{P}\left(\widehat{Val}\right)$$

$$\widehat{val} \in \widehat{Val} = \widehat{Proc} + \boxed{\widehat{Bas}}$$

$$\widehat{proc} \in \widehat{Proc} = \widehat{Clo}$$

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

$$\boxed{\widehat{bas} \in \widehat{Bas} = \mathsf{Num}^{\top}_{\bot}}$$

$\hat{a} \in \widehat{Addr}$ is a **finite** set of addresses

$\hat{t} \in \widehat{Time}$ is a **finite** set of time-stamps

k-CFA with constants

When $call = [\![\texttt{(if}\ e\ call_t\ call_f\texttt{)}]\!]$, if $\{\texttt{0}\} \not\sqsubseteq \hat{\mathcal{E}}(e, \hat{\beta}, \hat{\sigma})$:

$$(call, \hat{\beta}, \hat{\sigma}, \hat{t}) \rightsquigarrow (call_t, \hat{\beta}, \hat{\sigma}, \hat{t}'), \text{ where}$$

$$\hat{t}' = \widehat{tick}(call, \hat{t})$$

k-CFA with constants

When $call = [\![\texttt{(if}\ e\ call_t\ call_f\texttt{)}]\!]$, if $\{\texttt{0}\} = \hat{\mathcal{E}}(e, \hat{\beta}, \hat{\sigma})$:

$$(call, \hat{\beta}, \hat{\sigma}, \hat{t}) \rightsquigarrow (call_f, \hat{\beta}, \hat{\sigma}, \hat{t}'), \text{ where}$$

$$\hat{t}' = \widehat{tick}(call, \hat{t})$$

k-CFA with constants

When $call = [\![\texttt{(if}\ e\ call_t\ call_f\texttt{)}]\!]$, if $\{\texttt{0}\} \sqsubset \hat{\mathcal{E}}(e, \hat{\beta}, \hat{\sigma})$:

$$(call, \hat{\beta}, \hat{\sigma}, \hat{t}) \rightsquigarrow (call', \hat{\beta}, \hat{\sigma}, \hat{t}'), \text{ where}$$

$$call' \in \{call_t, call_f\}$$

$$\hat{t}' = \widehat{tick}(call, \hat{t})$$

Wrap-up

- Techniques for 0CFA
- Frameworks for CFA
- Methods to improve
- Applications to drive

Questions?