

Abstracting Abstract Machines

Storing & Stacking Continuations

David Van Horn
Northeastern

Matthew Might
Utah

Christopher Earl
Utah

What is an Abstract Machine?

An idealized, low-level model of an interpreter.
Typically: a first-order state transition system

$$s \mapsto s'$$

where each transition is a unit-cost operation.

Examples: Landin's SECD, Felleisen & Friedman's CEK,
Krivine's machine, ...

Non-examples: compositional evaluation function.

What is an Abstract Interpreter?

A computable approximation to an interpreter.

Typically: a sound approximation of intensional properties of program execution.

Examples: Shivers' kCFA, ...

Sort of non-examples: constraint-based analyses, type inference, ...

Storing:

store-allocate continuations & bindings
in a bounded store for finite state-space
abstractions.

Van Horn, Might
Abstracting Abstract Machines
ICFP, 2010

Stacking:

store-allocate bindings, but keep continuations
on the stack for infinite state-space
abstractions.

Earl, Might, Van Horn
Push-down control-flow of higher-order programs
SFP, 2010

PART I:

Storing

Expressions

Expr $e ::= x \mid \lambda x. e \mid e e$

Values

Den $d ::= \langle \lambda x. e, \rho \rangle$

Continuations

Cont $k ::= mt \mid ar(e, \rho, k) \mid fn(d, k)$

Continuations represent evaluation contexts (inside out)

$E ::= [] \mid (E e) \mid (d E)$

$[] \approx mt$

$E[[] e] \approx ar(e, \rho, k)$

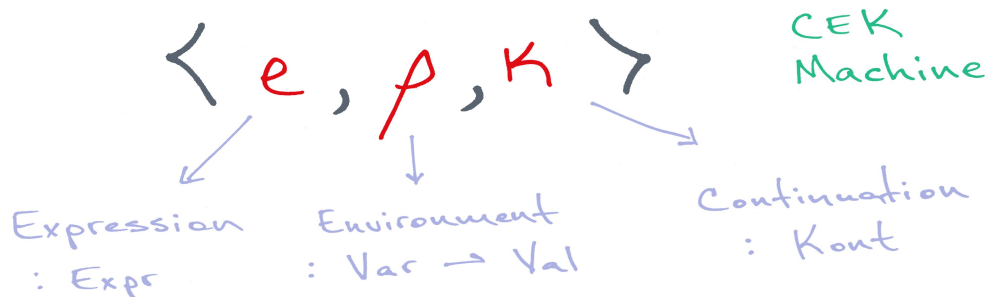
where $k \approx E$ and ρ closes e .

$E[(d [])] \approx fn(d, k)$

where $k \approx E$.

States

$s ::= \langle e, \rho, \kappa \rangle \mid \langle d, \kappa \rangle$



$\langle x, \rho, \kappa \rangle \mapsto \langle d, \kappa \rangle$ where $d = \rho(x)$

$\langle \lambda x. e, \rho, \kappa \rangle \mapsto \langle \langle \lambda x. e, \rho \rangle, \kappa \rangle$

$\langle e_0 e_1, \rho, \kappa \rangle \mapsto \langle e_0, \rho, \text{arg}(e_1, \rho, \kappa) \rangle$

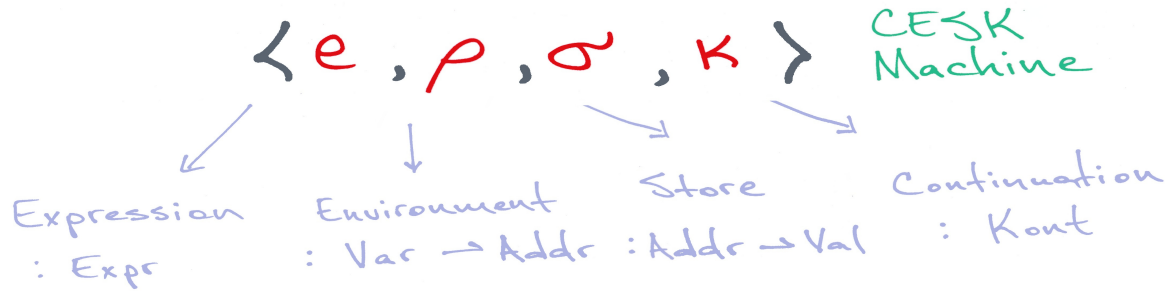
$\langle d, \text{arg}(e, \rho, \kappa) \rangle \mapsto \langle e, \text{fn}(d, \kappa) \rangle$

$\langle d', \text{fn}(d, \kappa) \rangle \mapsto \langle e, \rho[x \mapsto d'] \rangle$

where $d = \langle \lambda x. e, \rho \rangle$

States

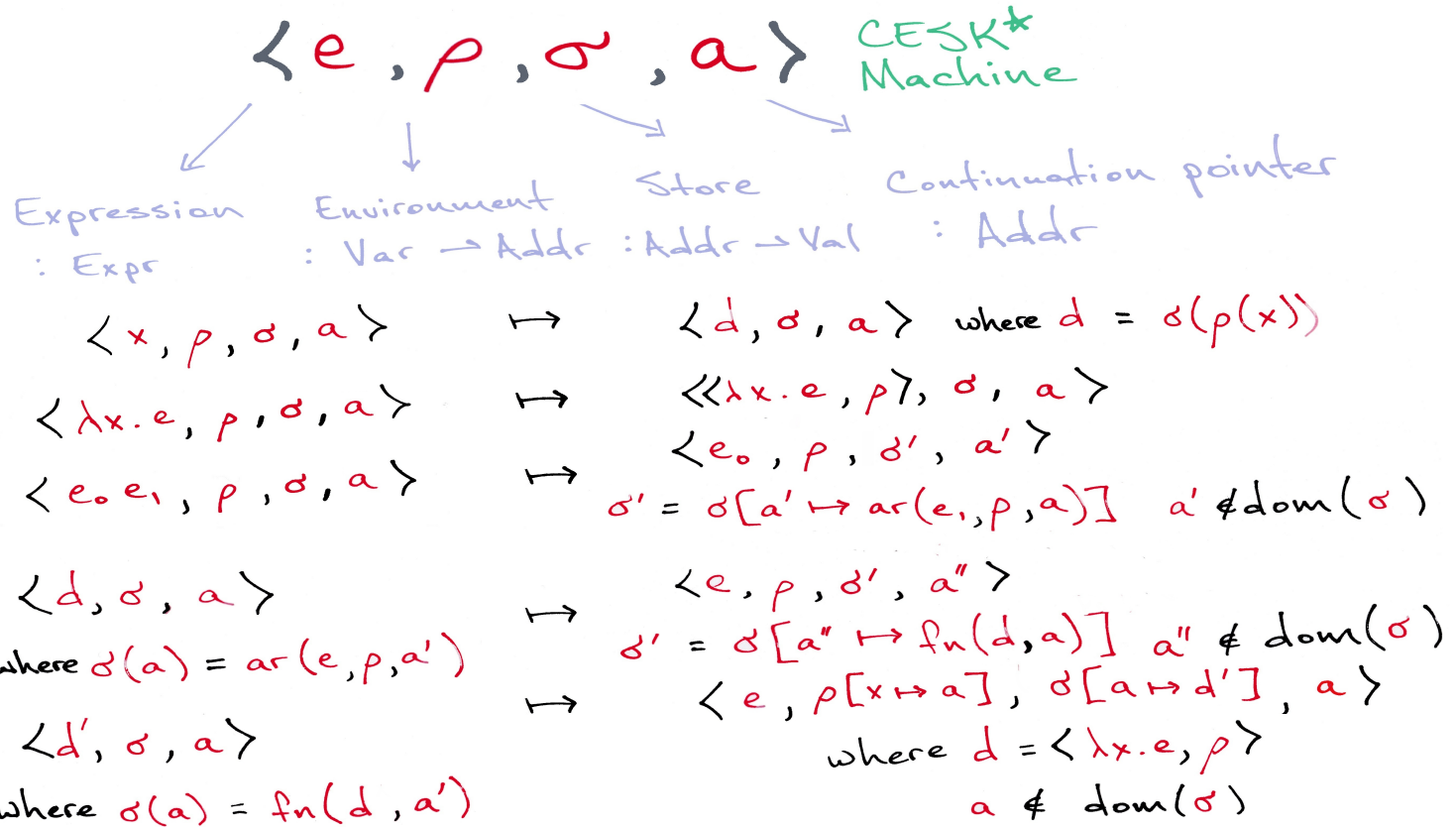
$s ::= \langle e, \rho, \sigma, \kappa \rangle \mid \langle d, \sigma, \kappa \rangle$



- $\langle x, \rho, \sigma, \kappa \rangle \mapsto \langle d, \sigma, \kappa \rangle$ where $d = \sigma(\rho(x))$
- $\langle \lambda x. e, \rho, \sigma, \kappa \rangle \mapsto \langle \langle \lambda x. e, \rho \rangle, \sigma, \kappa \rangle$
- $\langle e_0 e_1, \rho, \sigma, \kappa \rangle \mapsto \langle e_0, \rho, \sigma, \text{arg}(e_1, \rho, \kappa) \rangle$
- $\langle d, \sigma, \text{arg}(e, \rho, \kappa) \rangle \mapsto \langle e, \sigma, \text{fn}(d, \kappa) \rangle$
- $\langle d', \sigma, \text{fn}(d, \kappa) \rangle \mapsto \langle e, \rho[x \mapsto a], \sigma[a \mapsto d'], \kappa \rangle$
where $d = \langle \lambda x. e, \rho \rangle$
 $a \notin \text{dom}(\sigma)$

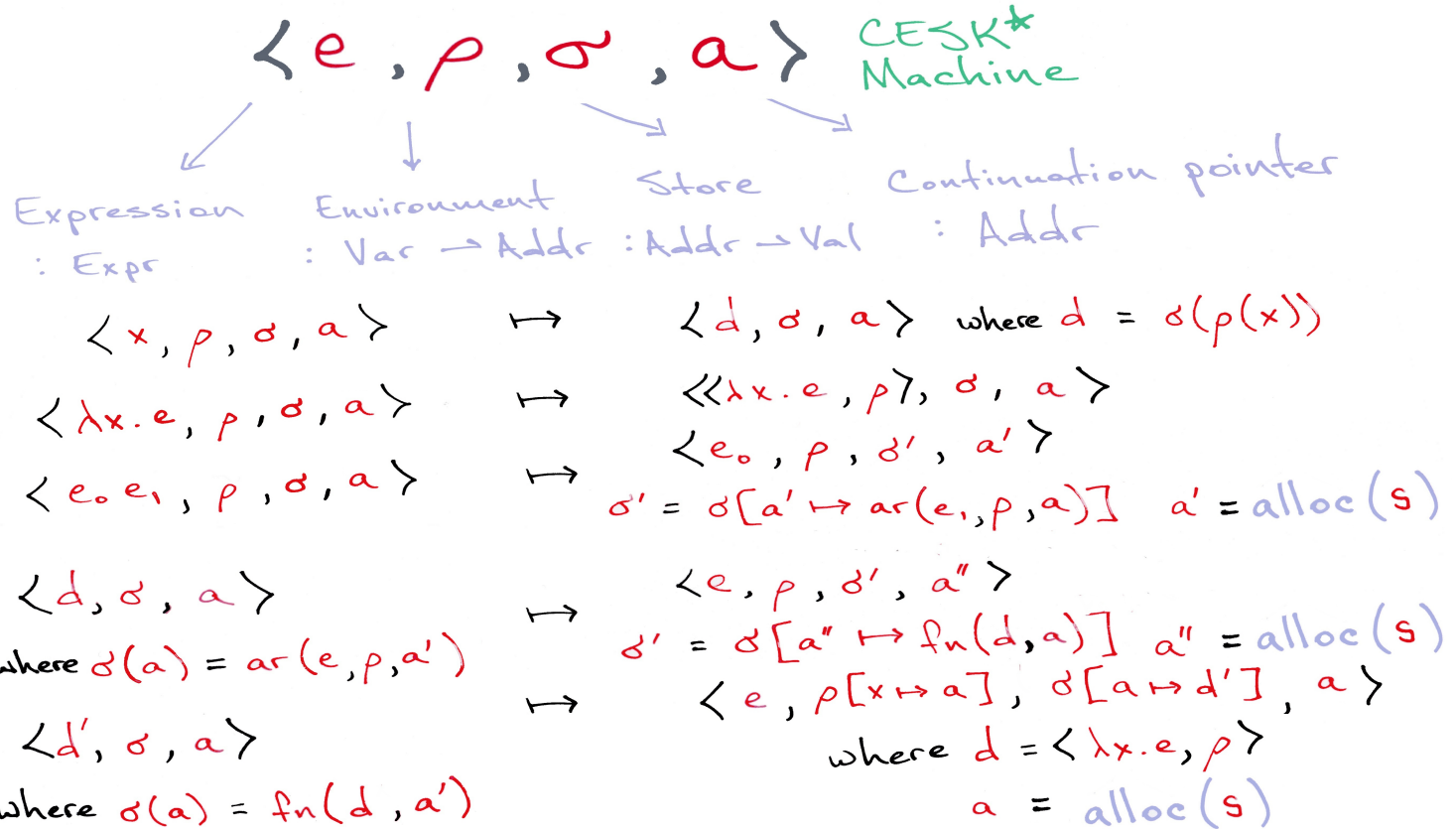
States

$s ::= \langle e, \rho, \sigma, a \rangle \mid \langle d, \sigma, a \rangle$



States

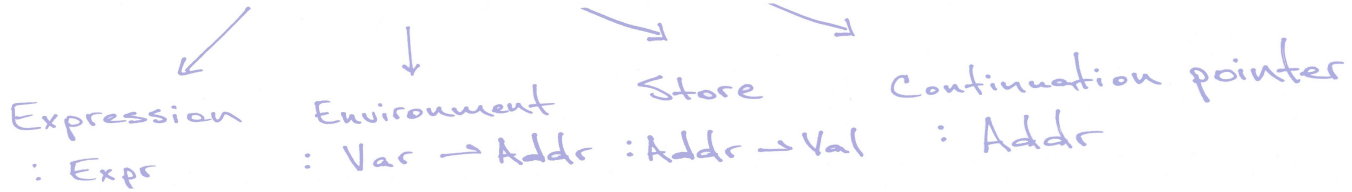
$s ::= \langle e, \rho, \sigma, a \rangle \mid \langle d, \sigma, a \rangle$



States

$$\hat{s} ::= \langle e, \rho, \sigma, a \rangle \mid \langle d, \sigma, a \rangle$$

$$\langle e, \rho, \sigma, a \rangle \quad \text{CEK Machine}$$



$$\langle x, \rho, \sigma, a \rangle \mapsto \langle d, \sigma, a \rangle \quad \text{where } d \ni \sigma(\rho(x))$$

$$\langle \lambda x. e, \rho, \sigma, a \rangle \mapsto \langle \langle \lambda x. e, \rho \rangle, \sigma, a \rangle$$

$$\langle e_0 e_1, \rho, \sigma, a \rangle \mapsto \langle e_0, \rho, \sigma', a' \rangle$$

$\sigma' = \sigma \cup [a' \mapsto \text{ar}(e_1, \rho, a)] \quad a' = \widehat{\text{alloc}}(s)$

$$\langle d, \sigma, a \rangle \mapsto \langle e, \rho, \sigma', a'' \rangle$$

$\sigma' = \sigma \cup [a'' \mapsto \text{fn}(d, a)] \quad a'' = \widehat{\text{alloc}}(s)$

where $\sigma(a) \ni \text{ar}(e, \rho, a')$

$$\langle d', \sigma, a \rangle$$

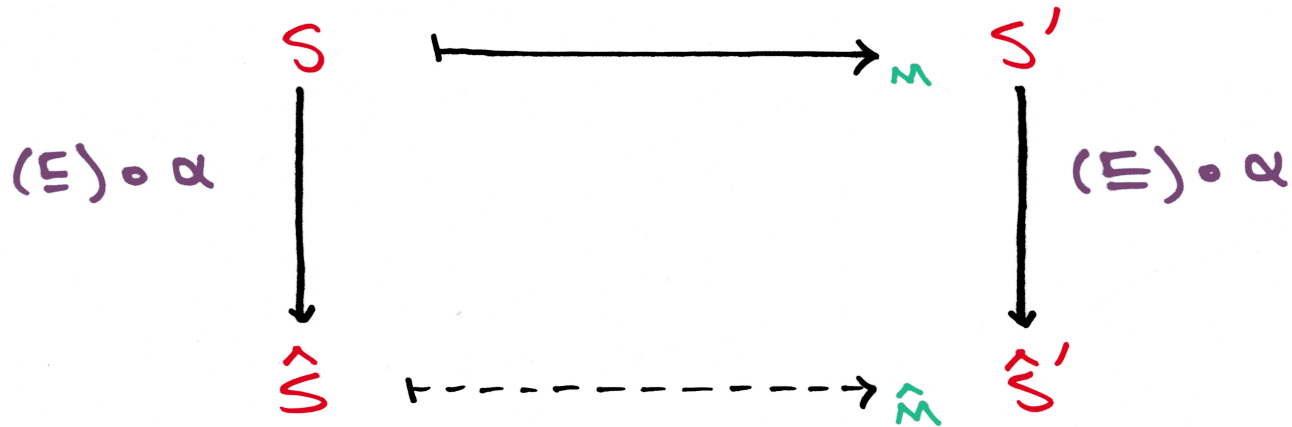
where $\sigma(a) \ni \text{fn}(d, a')$

$$\mapsto \langle e, \rho[x \mapsto a], \sigma \cup [a \mapsto d'], a \rangle$$

where $d = \langle \lambda x. e, \rho \rangle$

$a = \widehat{\text{alloc}}(s)$

Soundness



Extends to:

- State

Felleisen, Friedman

A calculus for assignments
in higher-order languages

POPL, 1987

- Exceptions

- First-class control

- Garbage collection

Felleisen, Findler, Flatt

Semantics Engineering with PLT Redex

MIT Press, 2009

- Laziness

Ager, Danvy, Midtgaard

A functional correspondence between
call-by-need evaluators & lazy abstract machines

Information Processing Letters, 2004

- Stack-inspection

Clements, Felleisen

A tail-recursive machine with stack-inspection

ACM Trans. Program. Lang. Syst., 2004

Future work:

- Blame analysis

Meunier, Findler, Felleisen

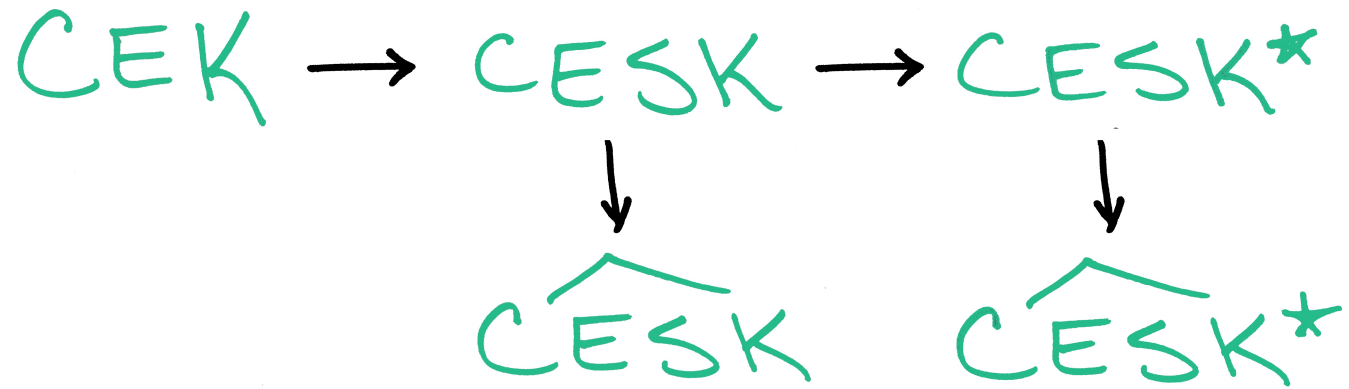
Modular set-based analysis from contracts

Principles of Programming Languages, 2006

PART II:

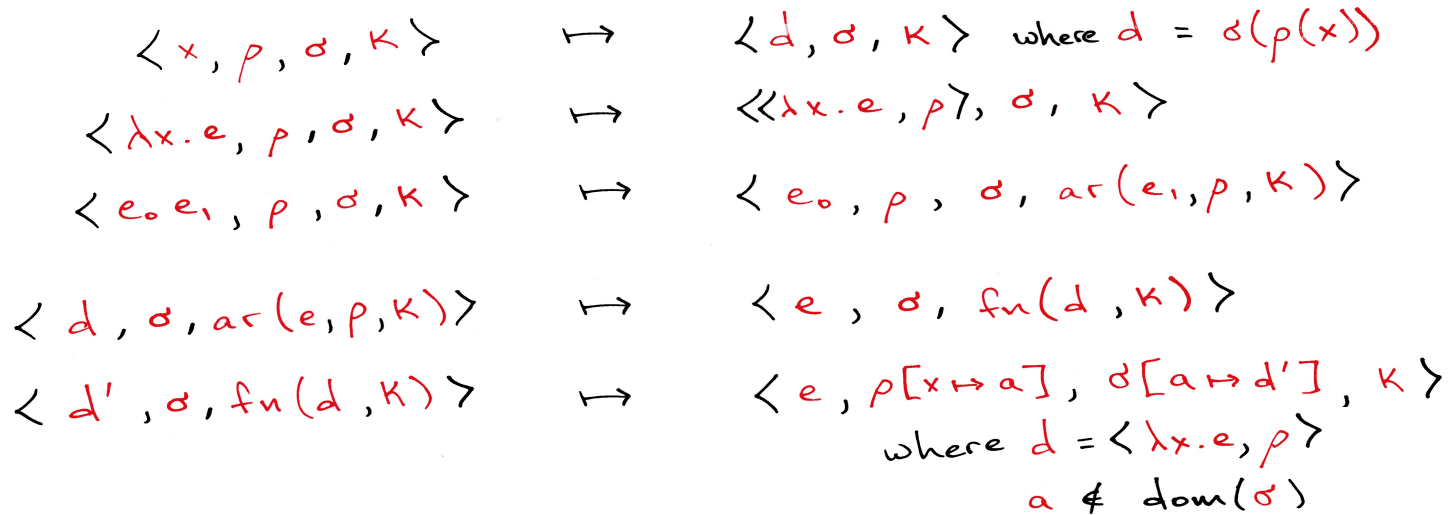
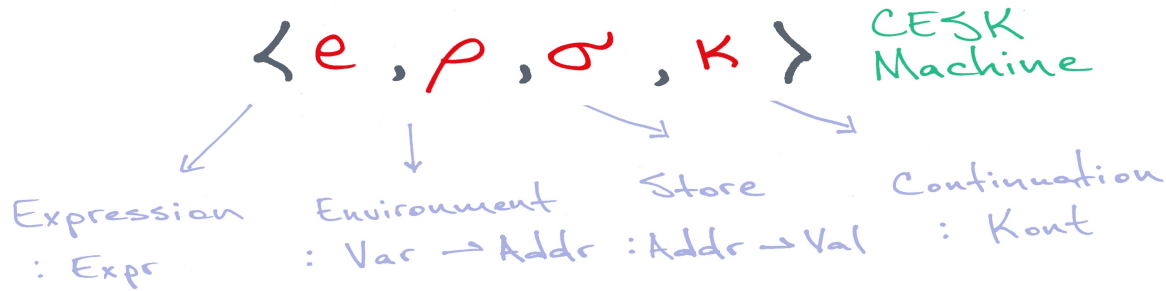
Stacking





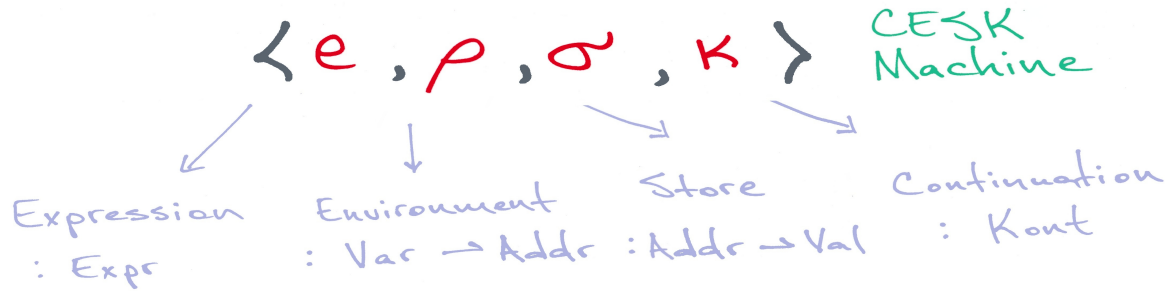
States

$s ::= \langle e, \rho, \sigma, \kappa \rangle \mid \langle d, \sigma, \kappa \rangle$



States

$s ::= \langle e, \rho, \sigma, \kappa \rangle \mid \langle d, \sigma, \kappa \rangle$



$\langle x, \rho, \sigma, \kappa \rangle \mapsto \langle d, \sigma, \kappa \rangle$ where $d = \sigma(\rho(x))$

$\langle \lambda x. e, \rho, \sigma, \kappa \rangle \mapsto \langle \langle \lambda x. e, \rho \rangle, \sigma, \kappa \rangle$

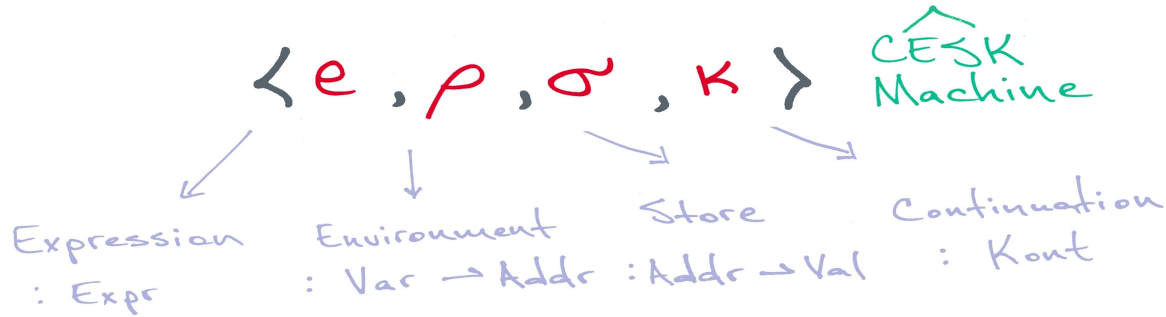
$\langle e_0 e_1, \rho, \sigma, \kappa \rangle \mapsto \langle e_0, \rho, \sigma, \text{arg}(e_1, \rho, \kappa) \rangle$

$\langle d, \sigma, \text{arg}(e, \rho, \kappa) \rangle \mapsto \langle e, \sigma, \text{fn}(d, \kappa) \rangle$

$\langle d', \sigma, \text{fn}(d, \kappa) \rangle \mapsto \langle e, \rho[x \mapsto a], \sigma[a \mapsto d'], \kappa \rangle$
where $d = \langle \lambda x. e, \rho \rangle$
 $a = \text{alloc}(s)$

States

$$\hat{s} ::= \langle e, \rho, \sigma, \kappa \rangle \mid \langle d, \sigma, \kappa \rangle$$



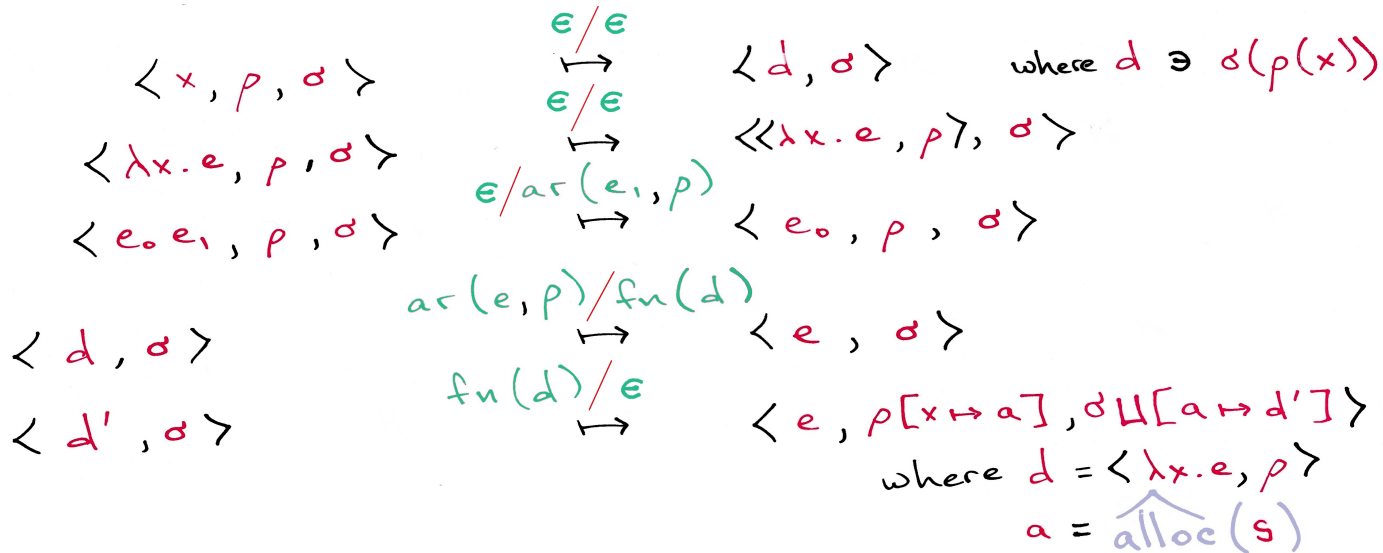
$$\begin{aligned} \langle x, \rho, \sigma, \kappa \rangle &\mapsto \langle d, \sigma, \kappa \rangle \text{ where } d \ni \sigma(\rho(x)) \\ \langle \lambda x. e, \rho, \sigma, \kappa \rangle &\mapsto \langle \langle \lambda x. e, \rho \rangle, \sigma, \kappa \rangle \\ \langle e_0 e_1, \rho, \sigma, \kappa \rangle &\mapsto \langle e_0, \rho, \sigma, \text{arg}(e_1, \rho, \kappa) \rangle \\ \langle d, \sigma, \text{arg}(e, \rho, \kappa) \rangle &\mapsto \langle e, \sigma, \text{fn}(d, \kappa) \rangle \\ \langle d', \sigma, \text{fn}(d, \kappa) \rangle &\mapsto \langle e, \rho[x \mapsto a], \sigma \sqcup [a \mapsto d'], \kappa \rangle \\ &\text{where } d = \langle \lambda x. e, \rho \rangle \\ &a = \widehat{\text{alloc}}(s) \end{aligned}$$

States

$\langle e, \rho, \sigma \rangle$

Stack alphabet

$ar(e, \rho) \mid fn(d)$



THE END

Thank You

