Abstracting Abstract Machines

Storing & Stacking Continuations

David Van Horn
Northeastern

Matthew Might
Utah

Christopher Earl
Utah

The Harvard Programming Languages Seminar

July 7, 2010

What is an Abstract Machine?

An idealized, low-level model of an interpreter.

Typically: a first-order state transition system

$$S \longmapsto S'$$

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	$e ::= x \mid \lambda x.e \mid ee$
Values	Den	$d ::= \langle \lambda x.e, \rho \rangle$
Continuations	Cont	$\kappa ::= \mathrm{mt} \mid \mathrm{ar}(e, \rho, \kappa) \mid \mathrm{fn}(d, \kappa)$

Continuations represent evaluation contexts (inside out)

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

$[\,] \approxeq \mathrm{mt}$

$E[([\,]\ e)] \approxeq \mathrm{ar}(e, \rho, \kappa)$

where $\kappa \approxeq E$ and ρ closes e.

$E[(d\ [\,])] \approxeq \mathrm{fn}(d, \kappa)$

where $\kappa \approxeq E$.

States

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

CEK Machine

$$\langle e, \rho, \kappa \rangle$$

Expression : Expr

Environment : $Var \rightharpoonup Val$

Continuation : Kont

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

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

$$\langle e_0\, e_1, \rho, \kappa \rangle \mapsto \langle e_0, \rho, \mathbf{ar}(e_1, \rho, \kappa) \rangle$$

$$\langle d, \mathbf{ar}(e, \rho, \kappa) \rangle \mapsto \langle e, \mathbf{fn}(d, \kappa) \rangle$$

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

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

States

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

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

CESK Machine

Expression : Expr

Environment : $Var \to Addr$

Store : $Addr \to Val$

Continuation : Kont

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

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

$$\langle e_0 e_1, \rho, \sigma, \kappa \rangle \mapsto \langle e_0, \rho, \sigma, ar(e_1, \rho, \kappa) \rangle$$

$$\langle d, \sigma, ar(e, \rho, \kappa) \rangle \mapsto \langle e, \sigma, fn(d, \kappa) \rangle$$

$$\langle d', \sigma, fn(d, \kappa) \rangle \mapsto \langle e, \rho[x \mapsto a], \sigma[a \mapsto d'], \kappa \rangle$$

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

$$a \notin dom(\sigma)$$

States

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

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

CESK* Machine

Expression : Expr

Environment : Var $\rightarrow$ Addr

Store : Addr $\rightarrow$ Val

Continuation pointer : Addr

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

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

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

$\sigma' = \sigma[a' \mapsto ar(e_1, \rho, a)]$ $\quad a' \notin dom(\sigma)$

$\langle d, \sigma, a \rangle$ where $\sigma(a) = ar(e, \rho, a')$ $\mapsto \langle e, \rho, \sigma', a'' \rangle$

$\sigma' = \sigma[a'' \mapsto fn(d, a)]$ $\quad a'' \notin dom(\sigma)$

$\langle d', \sigma, a \rangle$ where $\sigma(a) = fn(d, a')$ $\mapsto \langle e, \rho[x \mapsto a], \sigma[a \mapsto d'], a \rangle$

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

$a \notin dom(\sigma)$

States

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

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

CESK* Machine

- Expression $: \mathit{Expr}$
- Environment $: \mathit{Var} \rightarrow \mathit{Addr}$
- Store $: \mathit{Addr} \rightarrow \mathit{Val}$
- Continuation pointer $: \mathit{Addr}$

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

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

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

$$\sigma' = \sigma[a' \mapsto ar(e_1, \rho, a)] \quad a' = alloc(s)$$

$$\langle d, \sigma, a \rangle \text{ where } \sigma(a) = ar(e, \rho, a') \mapsto \langle e, \rho, \sigma', a'' \rangle$$

$$\sigma' = \sigma[a'' \mapsto fn(d, a)] \quad a'' = alloc(s)$$

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

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

$$a = alloc(s)$$

States

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

$$\langle e, \rho, \sigma, a \rangle$$ $\widehat{\text{CESK}^*}$ Machine

- Expression : Expr
- Environment : Var $\to$ Addr
- Store : Addr $\to$ Val
- Continuation pointer : Addr

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

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

$$\langle e_0 e_1, \rho, \sigma, a \rangle \mapsto \langle e_0, \rho, \sigma', a' \rangle$$
$$\sigma' = \sigma \sqcup [a' \mapsto ar(e_1, \rho, a)] \quad a' = \widehat{alloc}(s)$$

$$\langle d, \sigma, a \rangle \text{ where } \sigma(a) \ni ar(e, \rho, a') \mapsto \langle e, \rho, \sigma', a'' \rangle$$
$$\sigma' = \sigma \sqcup [a'' \mapsto fn(d, a)] \quad a'' = \widehat{alloc}(s)$$

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

Soundness

$$
\begin{tikzcd}
S \arrow[r, mapsto, "m"'] \arrow[d, "(E)\circ\alpha"'] & S' \arrow[d, "(E)\circ\alpha"] \\
\hat{S} \arrow[r, mapsto, dashed, "\hat{m}"'] & \hat{S}'
\end{tikzcd}
$$

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

$$\begin{tikzcd} \mathrm{CEK} \arrow[r] & \mathrm{CESK} \arrow[r] & \mathrm{CESK}^* \arrow[d] \\ & & \widehat{\mathrm{CESK}}^* \end{tikzcd}$$

$$
\begin{tikzcd}
\mathrm{CEK} \arrow[r] & \mathrm{CESK} \arrow[r] \arrow[d] & \mathrm{CESK}^{*} \arrow[d] \\
 & \widehat{\mathrm{CESK}} & \widehat{\mathrm{CESK}^{*}}
\end{tikzcd}
$$

States

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

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

CESK Machine

Expression : Expr

Environment : Var → Addr

Store : Addr → Val

Continuation : Kont

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

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

$$\langle e_0 e_1, \rho, \sigma, \kappa \rangle \mapsto \langle e_0, \rho, \sigma, ar(e_1, \rho, \kappa) \rangle$$

$$\langle d, \sigma, ar(e, \rho, \kappa) \rangle \mapsto \langle e, \sigma, fn(d, \kappa) \rangle$$

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

States

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

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

CESK Machine

- e — Expression : Expr
- ρ — Environment : Var $\to$ Addr
- σ — Store : Addr $\to$ Val
- κ — Continuation : Kont

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

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

$$\langle e_0 e_1, \rho, \sigma, \kappa \rangle \mapsto \langle e_0, \rho, \sigma, ar(e_1, \rho, \kappa) \rangle$$

$$\langle d, \sigma, ar(e, \rho, \kappa) \rangle \mapsto \langle e, \sigma, fn(d, \kappa) \rangle$$

$$\langle d', \sigma, fn(d, \kappa) \rangle \mapsto \langle e, \rho[x \mapsto a], \sigma[a \mapsto d'], \kappa \rangle$$

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

$$a = alloc(s)$$

States

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

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

CE$\hat{S}$K Machine

Expression : Expr

Environment : Var $\to$ Addr

Store : Addr $\to$ Val

Continuation : Kont

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

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

$$\langle e_0\, e_1, \rho, \sigma, \kappa \rangle \mapsto \langle e_0, \rho, \sigma, ar(e_1, \rho, \kappa) \rangle$$

$$\langle d, \sigma, ar(e, \rho, \kappa) \rangle \mapsto \langle e, \sigma, fn(d, \kappa) \rangle$$

$$\langle d', \sigma, fn(d, \kappa) \rangle \mapsto \langle e, \rho[x \mapsto a], \sigma \sqcup [a \mapsto d'], \kappa \rangle$$

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

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

States

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

Stack alphabet

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

$$\langle x, \rho, \sigma \rangle \xmapsto{\epsilon/\epsilon} \langle d, \sigma \rangle \quad \text{where } d \ni \sigma(\rho(x))$$

$$\langle \lambda x.e, \rho, \sigma \rangle \xmapsto{\epsilon/\epsilon} \langle \langle \lambda x.e, \rho \rangle, \sigma \rangle$$

$$\langle e_0 e_1, \rho, \sigma \rangle \xmapsto{\epsilon/ar(e_1, \rho)} \langle e_0, \rho, \sigma \rangle$$

$$\langle d, \sigma \rangle \xmapsto{ar(e, \rho)/fn(d)} \langle e, \sigma \rangle$$

$$\langle d', \sigma \rangle \xmapsto{fn(d)/\epsilon} \langle e, \rho[x \mapsto a], \sigma \sqcup [a \mapsto d'] \rangle$$

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

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

THE END

Thank You