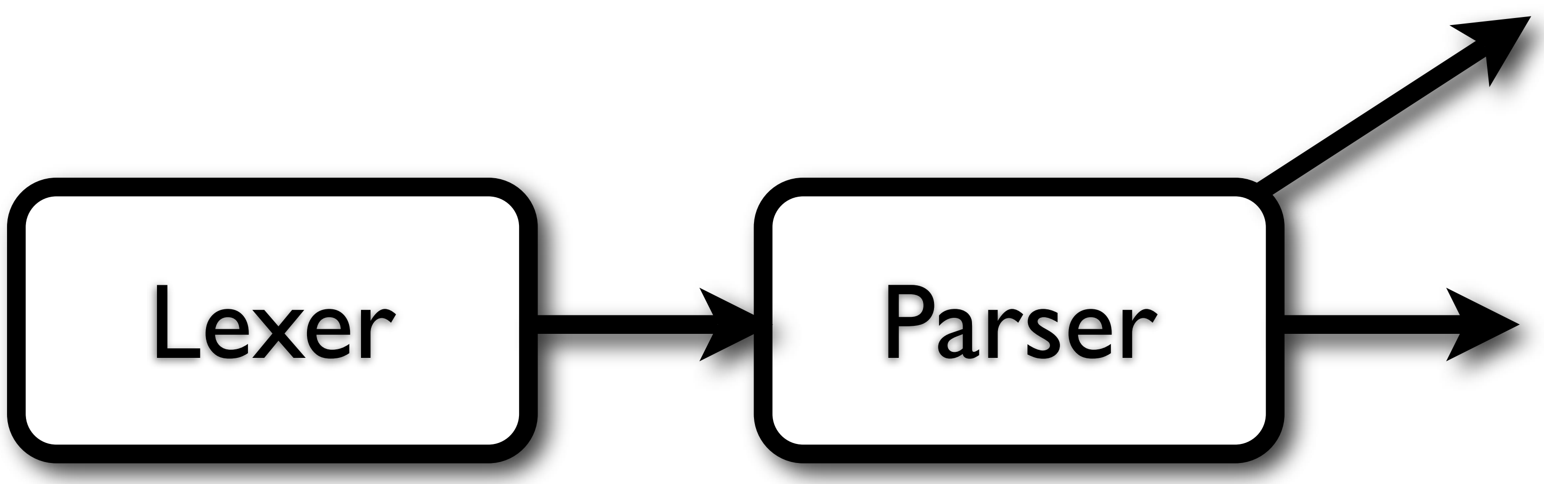
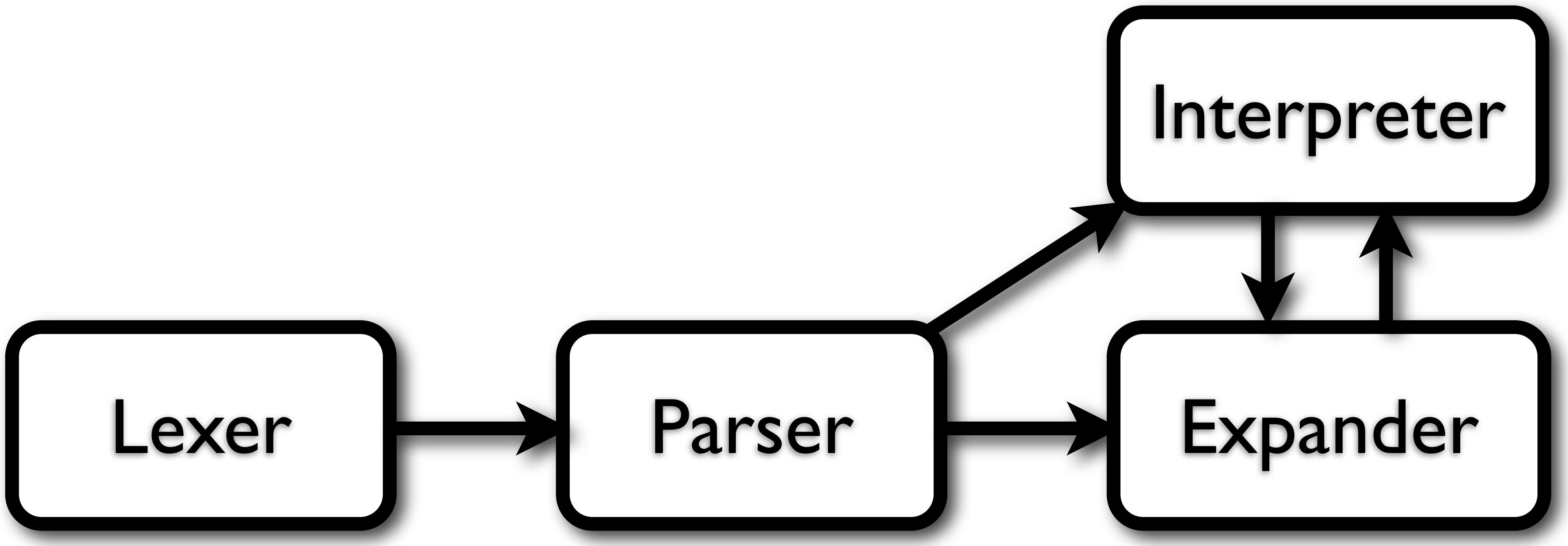
Semantics and interpreters

Matthew Might
University of Utah
matt.might.net
ucombinator.org

The road thus far...



The road thus far...



Agenda

- Types of interpreters
- Examples: Math and code

Semantics

A **semantics** is equivalent to a
function that maps a program to its meaning.

Interpreters

An **interpreter** is a program
that computes a program's meaning.

The difference?

Side effects!

Interpreter v. compiler

Are they equivalent?

Is there a language which **must** be compiled?

Is there a language which **must** be interpreted?

Equivalence

$$\text{Interpreter} = \text{Program} \times \text{Input} \rightarrow \text{Output}$$

$$\text{Compiler} = \text{Program} \rightarrow (\text{Input} \rightarrow \text{Output})$$

Interpreter styles

- Substitution v. environment
- Denotational v. operational
- Big-step v. small-step rules
- Continuation-passing v. direct

Substitution-based

- Machine configuration is program text
- Execution is program transformation

Example: Substitution

```
(let ((f (lambda (x) x))) (f 3))
```

Environment-based

- Configuration is program text, env.
- Environment maps variable to value
- Program text is never transformed

Example

```
((f x), [f → ((lambda (x) z),[z → 3]),
         x → 10])
```

Denotational semantics

- Computes "denotation" of program terms
- Mutually recursive functions over domains

Typical functions

$$\mathcal{K} : \textsf{Num} \rightarrow \mathbb{Z}$$

$$\mathcal{E} : \textsf{Exp} \times Env \rightarrow Value$$

$$\mathcal{D} : \textsf{Def} \rightarrow \textsf{Var} \times Value$$

$$\mathcal{S} : \textsf{Stmt} \rightarrow State \rightarrow State$$

$$\mathcal{A} : \textsf{Lambda} \times Env \rightarrow (Value \rightarrow Value)$$

Example: Arithmetic

$$
\begin{aligned}
e \in \mathsf{Exp} &::= t \texttt{+} e \mid t \\
t \in \mathsf{Term} &::= f \texttt{*} t \mid f \\
f \in \mathsf{Factor} &::= \texttt{(}\ e\ \texttt{)} \\
&\mid\ \ c \\
c \in \mathsf{Const} &\ \ \text{is}\ \ \text{a set of constants,}
\end{aligned}
$$

Example: Lambda

$$d \in D = D \to D$$

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

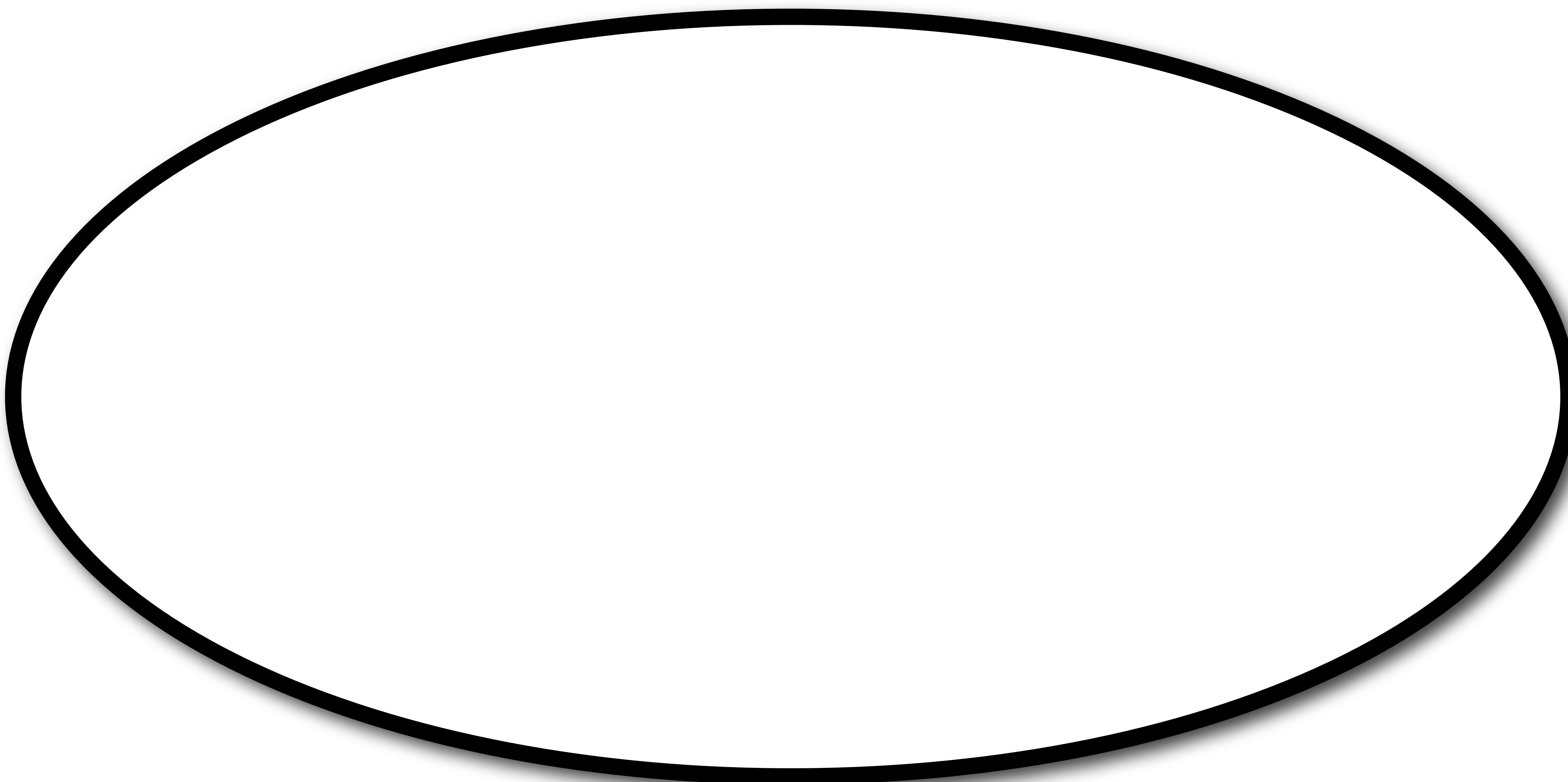
Example: Lambda

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

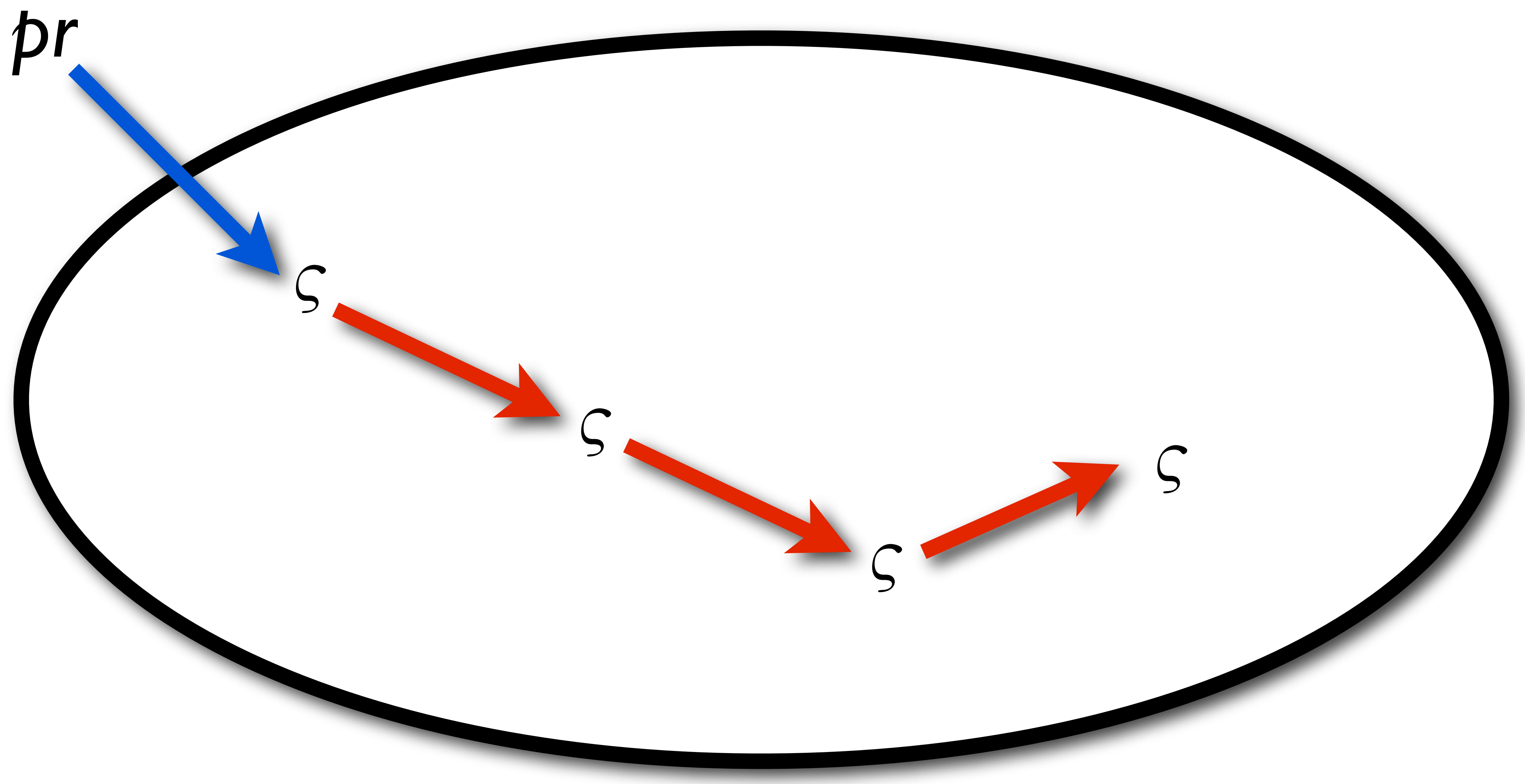
Example: Lambda

$$
\begin{aligned}
\mathcal{E}(\llbracket v \rrbracket, \rho) &= \rho(v) \\
\mathcal{E}(\llbracket \texttt{(}f\ e\texttt{)} \rrbracket, \rho) &= (\mathcal{E}(f, \rho))(\mathcal{E}(e, \rho)) \\
\mathcal{E}(\llbracket \texttt{(}\lambda\ \texttt{(}v\texttt{)}\ e\texttt{)} \rrbracket, \rho) &= \lambda d.\mathcal{E}(e, \rho[v \mapsto d])
\end{aligned}
$$

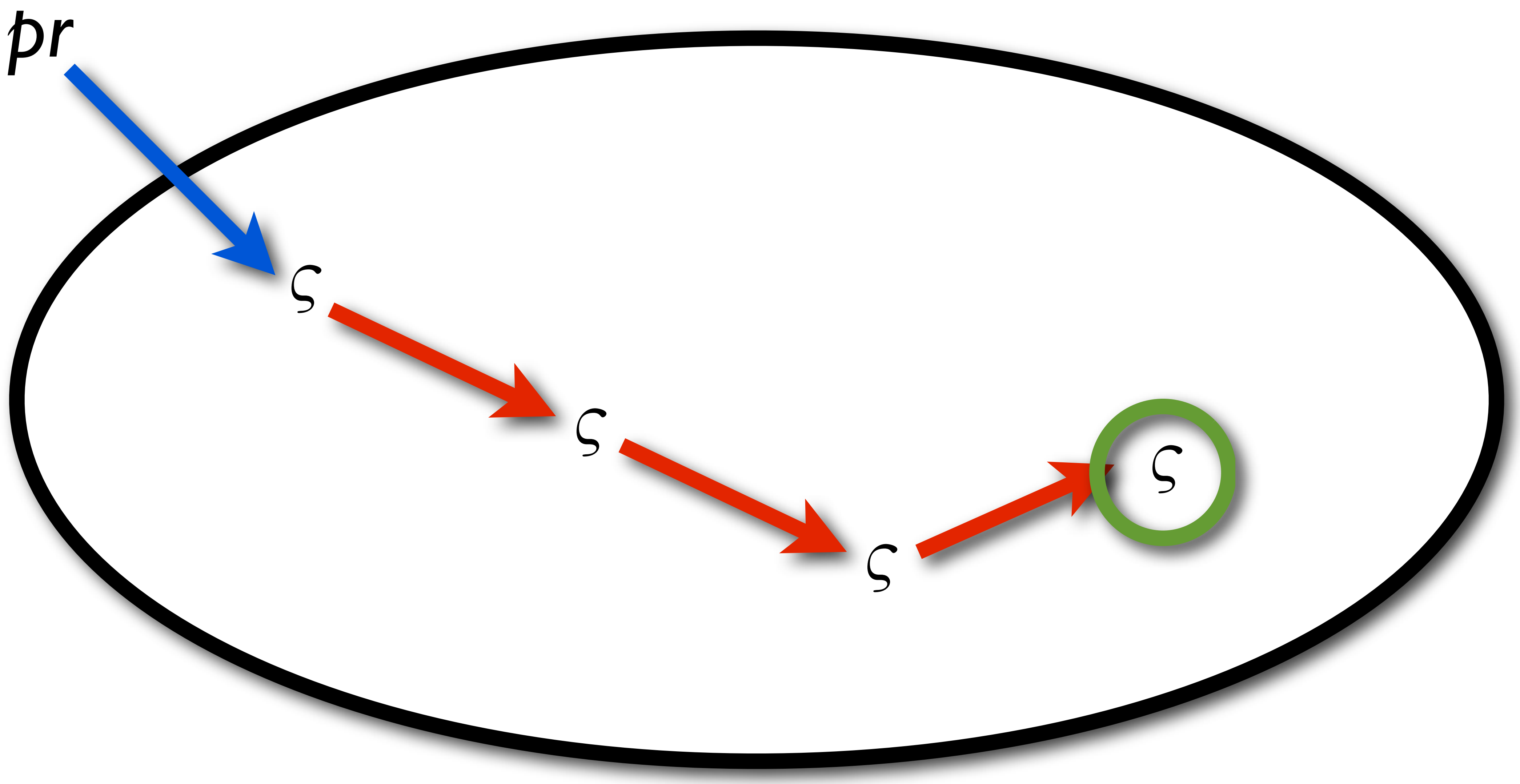
Operational semantics



Operational semantics



Operational semantics



Operational semantics

- Configuration (state) space
- Transition relation/function

State-space

$$\Sigma$$

Transition relation

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

$$(\Rightarrow) : \Sigma \to \mathcal{P}(\Sigma)$$

Small-step

Given one state, subsequent states are computable.

Big-step

Given one state, subsequent states may be incomputable.

Big-step relation

$$(\Downarrow) \subseteq \Sigma \times \Sigma$$

Example: Small-step

$$(\llbracket v \;\; \texttt{:=} \;\; e \rrbracket : \vec{s}, \rho) \Rightarrow (\vec{s}, \rho[v \mapsto \mathcal{A}(e, \rho)])$$

Example: Big-step

$$\frac{(f,\rho) \Downarrow (\llbracket\texttt{(}\lambda\ \ \texttt{(}v\texttt{)}\ \ e'\texttt{)}\rrbracket,\rho') \quad (e,\rho) \Downarrow cl \quad (e',\rho'[v \mapsto cl]) \Downarrow cl'}{(\llbracket\texttt{(}f\ \ e\texttt{)}\rrbracket,\rho) \Downarrow cl'}$$

Extended example:
Small-step miniScheme

Grammar

$$
\begin{array}{rcl}
æ \in \textsf{Atom} & = & \textsf{Lam} + \textsf{Const} + \textsf{Prim} + \textsf{Var} \\
\\
lam \in \textsf{Lam} & ::= & \texttt{(}\lambda\ \texttt{(}v_1 \ldots v_n\texttt{)}\ e\texttt{)} \\
v \in \textsf{Var} & = & \{[\![\texttt{x}]\!], [\![\texttt{y}]\!], [\![\texttt{foo}]\!], \ldots\} \\
c \in \textsf{Const} & = & \textsf{Int} + \textsf{Bool} \\
op \in \textsf{Prim} & = & \{[\![\texttt{+}]\!], [\![\texttt{*}]\!], [\![\texttt{halt}]\!], \ldots\} \\
\\
i \in \textsf{Int} & = & \{\ldots, [\![\texttt{-1}]\!], [\![\texttt{0}]\!], [\![\texttt{1}]\!], \ldots\} \\
b \in \textsf{Bool} & ::= & \texttt{\#t} \mid \texttt{\#f} \\
\\
f, e \in \textsf{Exp} & ::= & æ \\
& \mid & \texttt{(}f\ e_1 \cdots e_n\texttt{)} \\
& \mid & \texttt{(let ((}v\ e_{\text{arg}}\texttt{))}\ e_{\text{body}}\texttt{)} \\
& \mid & \texttt{(letrec ((}v_1\ lam_1\texttt{)} \cdots \texttt{(}v_n\ lam_n\texttt{))}\ e\texttt{)} \\
& \mid & \texttt{(if}\ e_{\text{cond}}\ e_{\text{true}}\ e_{\text{false}}\texttt{)} \\
& \mid & \texttt{(set!}\ v\ e\texttt{)} \\
& \mid & \texttt{(begin}\ e_1 \cdots e_n\texttt{)}.
\end{array}
$$

State-space

$$
\begin{array}{rcl}
\varsigma \in \Sigma & = & \mathsf{ValExp} \times \mathit{Env} \times \mathit{Store} \times \mathit{KontPtr} \times \mathit{Time} \\
\\
\rho \in \mathit{Env} & = & \mathsf{Var} \rightharpoonup \mathit{Addr} \\
\sigma \in \mathit{Store} & = & \mathit{Addr} \rightharpoonup D \\
a \in \mathit{Addr} & \text{is} & \text{an infinite set of addresses} \\
\overset{\kappa}{p} \in \mathit{KontPtr} & \subseteq & \mathit{Addr} \\
\\
d \in D & = & \mathit{Val} \\
\mathit{val} \in \mathit{Val} & = & \mathit{Proc} + \mathit{Bas} + \mathit{Kont} \\
\mathit{proc} \in \mathit{Proc} & = & \mathit{Clo} + \mathsf{Prim} \\
\mathit{bas} \in \mathit{Bas} & = & \mathbb{Z} + \mathsf{Bool} \\
\kappa \in \mathit{Kont} & = & \mathsf{Val\dot{E}xp} \times \mathit{Env} \times \mathit{KontPtr} \\
 & + & \{\mathbf{halt}\}
\end{array}
$$

Value expressions

$$
\begin{aligned}
V \in \mathsf{ValExp} ::= {} & d \\
\mid {} & \text{æ} \\
\mid {} & (d_1 \ldots d_n\ e_1 \cdots e_m) \\
\mid {} & (\texttt{let}\ ((v\ V))\ e_{\text{body}}) \\
\mid {} & (\texttt{letrec}\ ((v_1\ lam_1) \cdots (v_n\ lam_n))\ e) \\
\mid {} & (\texttt{if}\ V\ e_{\text{true}}\ e_{\text{false}}) \\
\mid {} & (\texttt{set!}\ v\ V) \\
\mid {} & (\texttt{begin}\ d_1 \cdots d_n\ e_1 \cdots e_m).
\end{aligned}
$$

Contexts

$$
\begin{aligned}
\dot{V} \in \mathsf{Val\dot{E}xp} ::= {} & \Box \\
\mid {} & (d_1 \ldots d_n\ \dot{V}\ e_1 \cdots e_m) \\
\mid {} & (\texttt{let}\ ((v\ \dot{V}))\ e_{\text{body}}) \\
\mid {} & (\texttt{letrec}\ ((v_1\ lam_1) \cdots (v_n\ lam_n))\ e) \\
\mid {} & (\texttt{if}\ \dot{V}\ e_{\text{true}}\ e_{\text{false}}) \\
\mid {} & (\texttt{set!}\ v\ \dot{V}) \\
\mid {} & (\texttt{begin}\ d_1 \cdots d_n\ \dot{V}\ e_1 \cdots e_m).
\end{aligned}
$$

Helpers

$$\mathcal{A} : \text{Atom} \times Env \times Store \rightharpoonup D$$

$$\mathcal{A}(lam, \rho, \sigma) = (lam, \rho)$$

$$\mathcal{A}(v, \rho, \sigma) = \sigma(\rho(v))$$

$$\mathcal{A}(c, \rho, \sigma) = \mathcal{K}(c)$$

$$\mathcal{A}(op, \rho, \sigma) = op.$$

Return

$$(d, \rho, \sigma, \overset{\kappa}{p}, t) \Rightarrow (\dot{V}[d], \rho', \sigma, \overset{\kappa}{p}{}', t'), \text{ where:}$$

$$(\dot{V}, \rho', \overset{\kappa}{p}{}') = \sigma(\overset{\kappa}{p})$$

$$t' = tick(t).$$

Atomic expressions

$$(\dot{V}[\text{æ}], \rho, \sigma, \overset{\kappa}{p}, t) \Rightarrow (\mathcal{A}(\text{æ}, \rho, \sigma), \rho, \sigma, \overset{\kappa}{p}, t'), \text{ where:}$$

$$t' = tick(t).$$

Non-atomics

$$\overbrace{(\dot{V}[e], \rho, \sigma, \overset{\kappa}{p}, t)}^{\varsigma} \Rightarrow (e, \rho, \sigma', \overset{\kappa}{p}{}', t'), \text{ where:}$$

$$\kappa' = (\dot{V}, \rho, \overset{\kappa}{p})$$

$$\overset{\kappa}{p}{}' = alloc_{\kappa}(\varsigma)$$

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

$$t' = tick(t).$$

Procedure call

If $(lam, \rho') = d_0$:

$$([\![(d_0\ d_1 \cdots d_n)]\!], \quad , \quad , \overset{\kappa}{p}, t) \Rightarrow (e', \quad '', \quad ', \overset{\kappa}{p}, t'), \text{ where:}$$

$$t' = tick(t)$$

$$lam = [\![(\lambda\ (v_1 \cdots v_n)\ e')]\!]$$

$$a_i = alloc(v_i, t')$$

$$'' = \quad '[v_i \mapsto a_i]$$

$$' = \quad [a_i \mapsto d_i].$$

Primitives

$$
\begin{aligned}
(\llbracket (d_0\ d_1 \cdots d_n) \rrbracket, \rho, \sigma, \overset{\kappa}{p}, t) \Rightarrow (d', \rho, \sigma, \overset{\kappa}{p}, t'),\ \text{where:} \\
t' &= tick(t) \\
d' &= \mathcal{O}(op)\langle d_1, \ldots, d_n \rangle.
\end{aligned}
$$

Letrec

$$([\![\texttt{(letrec (}\cdots\texttt{(}v_i\ lam_i\texttt{)}\cdots\texttt{)}\ e\texttt{)}]\!], \rho, \sigma, \overset{\kappa}{p}, t) \Rightarrow (e, \rho', \sigma', \overset{\kappa}{p}, t'), \text{ where:}$$

$$\begin{aligned} t' &= tick(t) \\ a_i &= alloc(v_i, \varsigma) \\ \rho' &= \rho[v_i \mapsto a_i] \\ d_i &= \mathcal{A}(lam_i, \rho', \sigma) \\ \sigma' &= \sigma[a_i \mapsto d_i]. \end{aligned}$$

Conditionals

$$([\![\texttt{(if \#t } e_{\text{true}}\ e_{\text{false}}\texttt{)}]\!], \rho, \sigma, \overset{\kappa}{p}, t) \Rightarrow (e_{\text{true}}, \rho, \sigma, \overset{\kappa}{p}, tick(t))$$

$$([\![\texttt{(if \#f } e_{\text{true}}\ e_{\text{false}}\texttt{)}]\!], \rho, \sigma, \overset{\kappa}{p}, t) \Rightarrow (e_{\text{false}}, \rho, \sigma, \overset{\kappa}{p}, tick(t))$$

Set!

$$
\begin{aligned}
([\![\texttt{(set! } v\ d\texttt{)}]\!], \rho, \sigma, \overset{\kappa}{p}, t) \Rightarrow (d_{\text{void}}, \rho, \sigma', \overset{\kappa}{p}, t), \text{ where:} \\
t' &= tick(t) \\
d &= \mathcal{A}(æ, \rho, \sigma) \\
\sigma' &= \sigma[\rho(v) \mapsto d].
\end{aligned}
$$