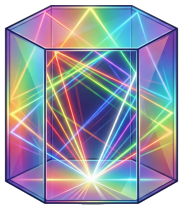

# Prism Semantics

*Unverified sketch*

## 1. Conventions

Sequences are written with an overbar, as in $\overline{v}$, and may be empty. Comma-separated extension is left-biased: the newest binding shadows an older binding with the same name. Square brackets denote syntactic substitution; angle brackets denote machine configurations. A superscript star on a relation denotes reflexive-transitive closure.

| Family | Notation | Meaning |
|---|---|---|
| $x, y, z$ | $\in$ Var | term variables |
| $f$ | $\in$ FnName | top-level Core function names |
| $K$ | $\in$ CtorName | data-constructor names |
| $\ell$ | $\in$ OpName | effect-operation names |
| $n, i$ | $\in \mathbb{Z}, \in \mathbb{N}$ | integer literals and natural tags |
| $d$ | $\in$ Float64 | IEEE-754 binary64 literals |
| $b, s$ | $\in \mathbb{B}, \in$ String | booleans and strings |
| $\overline{a}$ | $\in$ List$(a)$ | a finite sequence of objects of family $a$ |

## 2. Static Core terms

The calculus is in computation-passing form: syntactic values $v$ are inert, while computations $c$ describe evaluation. Patterns $p$ occur only in case arms. Handler clauses $h$ bind operation parameters and a resumption variable.

### 2.1. Patterns

$$p ::= x \mid n \mid d \mid b \mid K(\overline{p}) \mid (\overline{p}) \mid K\{\overline{x : p}\}_{\omega}$$

The subscript $\omega \in \{\text{open}, \text{closed}\}$ records whether a record pattern admits fields not explicitly listed. Pattern matching produces a finite binding environment when it succeeds.

### 2.2. Values

$$v ::= x \mid n \mid d \mid b \mid () \mid s \mid \text{thunk}$$
$$c \mid K_{i(\overline{v})} \mid (\overline{v})$$

$K_i$ pairs a constructor name with its natural-number runtime tag. A thunk is syntax here; it becomes a closure over an environment when converted to a runtime value.

### 2.3. Handler clauses and computations

$$h ::= \ell(\overline{x}, k) \Rightarrow c$$

Here $k$ is the variable bound to the captured, multishot resumption.

$$\begin{aligned} c ::= \ & \text{return } v \mid c \text{ to } x.c \\ & \mid \text{force } v \mid \lambda\overline{x}.c \\ & \mid c(\overline{v}) \\ & \mid \text{if } v \text{ then } c \text{ else } c \\ & \mid \delta_{\mathtt{op}}(v_1, v_2) \mid \mathrm{neg}_{\lambda(v)} \\ & \mid f(\overline{v}) \mid \text{case } v \text{ of } \overline{p \Rightarrow c} \\ & \mid \text{do } \ell(\overline{v}) \\ & \mid \text{handle } c \text{ with } \{c_r; \overline{h}\} \\ & \mid \text{mask } \overline{\ell} \text{ in } c \mid \mathrm{builtin}_{a(\overline{v})} \\ & \mid \text{dup } v \mid \text{drop } v \\ & \mid \text{with-reuse } t = v \text{ in } c \mid \text{reuse } t \text{ as } v \\ & \mid \text{error } v \end{aligned}$$

$\mathtt{op}$ ranges over the integer, boolean, and floating primitive operations; $\lambda \in \{\mathrm{Int}, \mathrm{I64}, \mathrm{Float64}\}$ is a negation lane. The metavariable $a$ names an opaque runtime builtin. The optional return clause $c_r$ binds the handled computation's ordinary result. Print, string, float, and input nodes are included in the $\mathrm{builtin}_a$ family in this presentation; the Lean syntax keeps their constructors separate.

### 2.4. Programs

$$F ::= f(\overline{x}) = c \quad \Gamma ::= \{\overline{F}\}$$

$\Gamma$ is a closed Core program: a finite table of named functions. Function lookup is written $\Gamma(f)$ and is partial.

## 3. Runtime terms

Runtime values are distinct from syntactic values because closures, thunks, and resumptions carry dynamic context.

$$r ::= n \mid d \mid b \mid () \mid s \mid \mathrm{closure}(\overline{x}, c, \rho) \mid \mathrm{thunk}(c, \rho) \mid K_{i(\overline{r})} \mid (\overline{r}) \mid \mathrm{resume}(\kappa)$$

$$\rho ::= \emptyset \mid \rho[x \mapsto r]$$

$\rho$ is a left-biased runtime environment. Lookup is $\rho(x)$, and simultaneous extension is $\rho[\overline{x \mapsto r}]$.

### 3.1. Continuation machine

$$\varphi ::= \mathrm{bind}(x, c, \rho) \mid \mathrm{args}(\overline{v}, \rho) \mid \mathrm{handler}\big(\overline{h}, c_r, \rho\big) \mid \mathrm{mask}\big(\overline{\ell}\big)$$

$$\kappa ::= \varepsilon \mid \varphi :: \kappa \quad \mu ::= \mathrm{eval}(c, \rho) \mid \mathrm{ret}(r) \quad q ::= \langle \mu, \kappa \rangle$$

$\varphi$ is one continuation frame, $\kappa$ a stack of frames, $\mu$ the CEK control state, and $q$ a complete machine configuration. The initial configuration for a closed computation is

$$\mathrm{load}(c) = \langle \mathrm{eval}(c, \emptyset), \varepsilon \rangle.$$

## 4. Auxiliary notation

| Family | Notation | Meaning |
|---|---|---|
| $c[v/x]$ | capture-avoiding substitution | replace free $x$ in $c$ by syntactic value $v$ |
| $c\big[\overline{v/x}\big]$ | simultaneous substitution | apply a finite parameter-to-argument binding |
| $\delta(\mathtt{op}, r_1, r_2) = r$ | primitive interpretation | evaluate a binary primitive when its lane and operands are valid |

| Family | Notation | Meaning |
|---|---|---|
| $\mathrm{neg}(\lambda, r) = r'$ | unary primitive interpretation | evaluate lane-specific numeric negation |
| $p$ matches $r = \theta$ | pattern matching | match $p$ against $r$ and return bindings $\theta$ |
| $\rho \vdash v \Downarrow r$ | atomic value evaluation | resolve a syntactic value in $\rho$, closing thunks over $\rho$ |
| $\Gamma(f) = (\overline{x}, c)$ | function lookup | find $f$'s parameters and body in the Core table |
| $H(\ell) = (\overline{x}, k, c)$ | handler-clause lookup | find the clause for operation $\ell$ in handler $H$ |
| $\mathrm{find}(\ell, \overline{r}, j, \kappa)$ | handler search | find the nearest matching handler after skipping $j$ masked matches |

All of these operations are partial. An undefined result is written $\bot$. This notation does not identify machine failure with semantic divergence: halting, an explicit error, and a stuck configuration will be distinguished when the dynamic rules are written.

# 5. Reserved judgement forms

The following declarations fix how later rules will be read. They do not yet define when any judgement is derivable.

| Family | Notation | Meaning |
|---|---|---|
| $\Gamma \vdash c \to c'$ | Core one-step reduction | $c$ reduces to $c'$ under program $\Gamma$ |
| $\Gamma \vdash c \xrightarrow{*} c'$ | Core many-step reduction | reflexive-transitive closure of Core reduction |
| $\Gamma \vdash q \mapsto q'$ | CEK transition | configuration $q$ takes one machine step to $q'$ |
| $\Gamma \vdash q \overset{*}{\mapsto} q'$ | CEK run | zero or more CEK transitions |
| $\Gamma; \rho \vdash c \Downarrow r$ | natural evaluation | $c$ evaluates to runtime value $r$ in $\rho$ |
| $\mathrm{terminal}_{\Gamma(c)}$ | terminal computation | $c$ is a Core result or function |
| $\mathrm{stuck}_{\Gamma(c)}$ | explicitly blocked computation | $c$ is nonterminal and has no Core successor |
| $\mathrm{handles}(\ell, j, \kappa)$ | handler coverage | $\kappa$ handles $\ell$ after $j$ matching handlers are skipped |
| $\mathrm{tunnels}(\ell, \kappa)$ | transparent stack segment | $\kappa$ contains no frame that traps or masks $\ell$ |