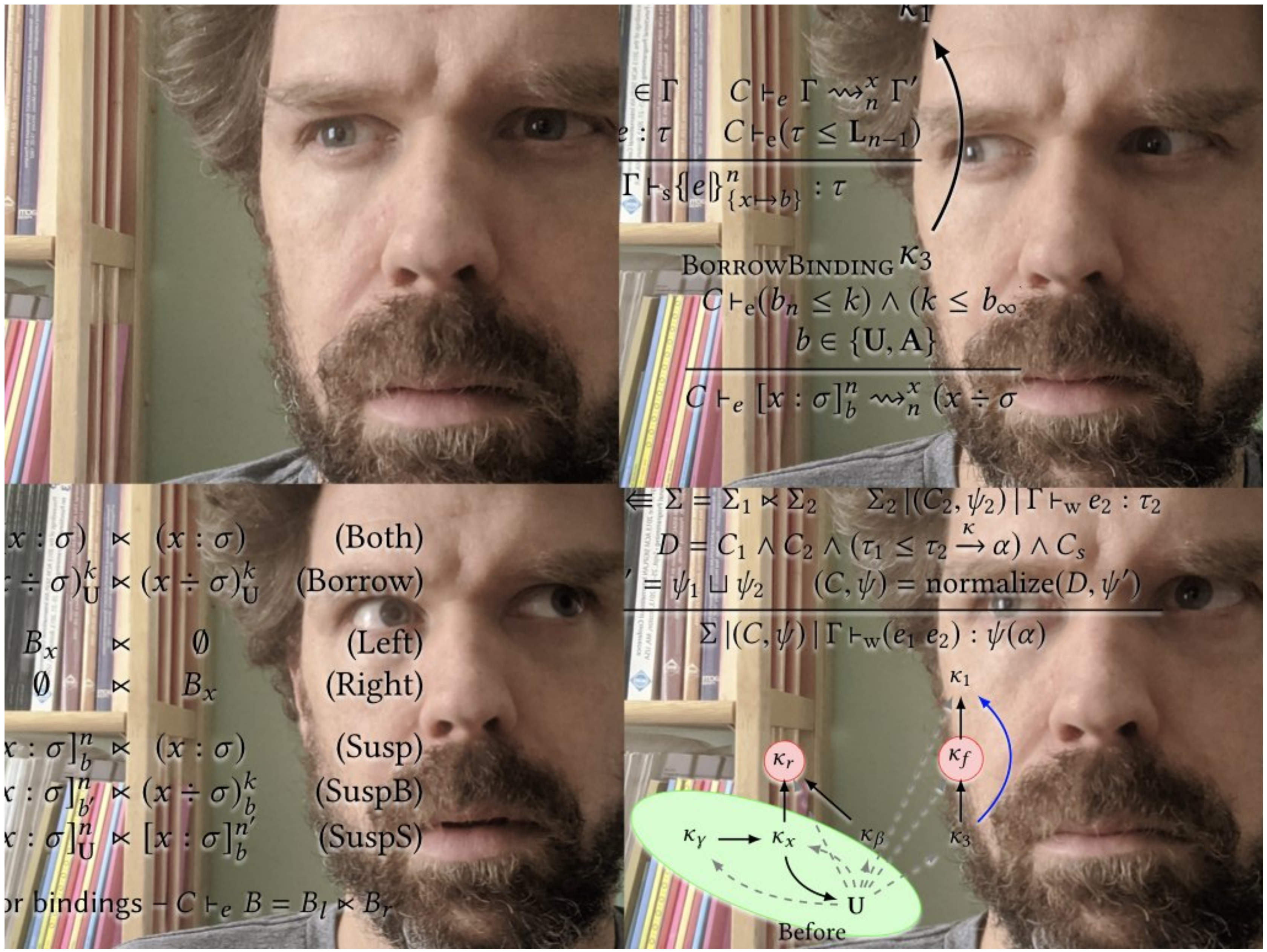
Basic Mechanics of Operational Semantics



David Van Horn

PLUM

What is an operational semantics?

A method of defining the meaning of programs by describing the actions carried out during execution.

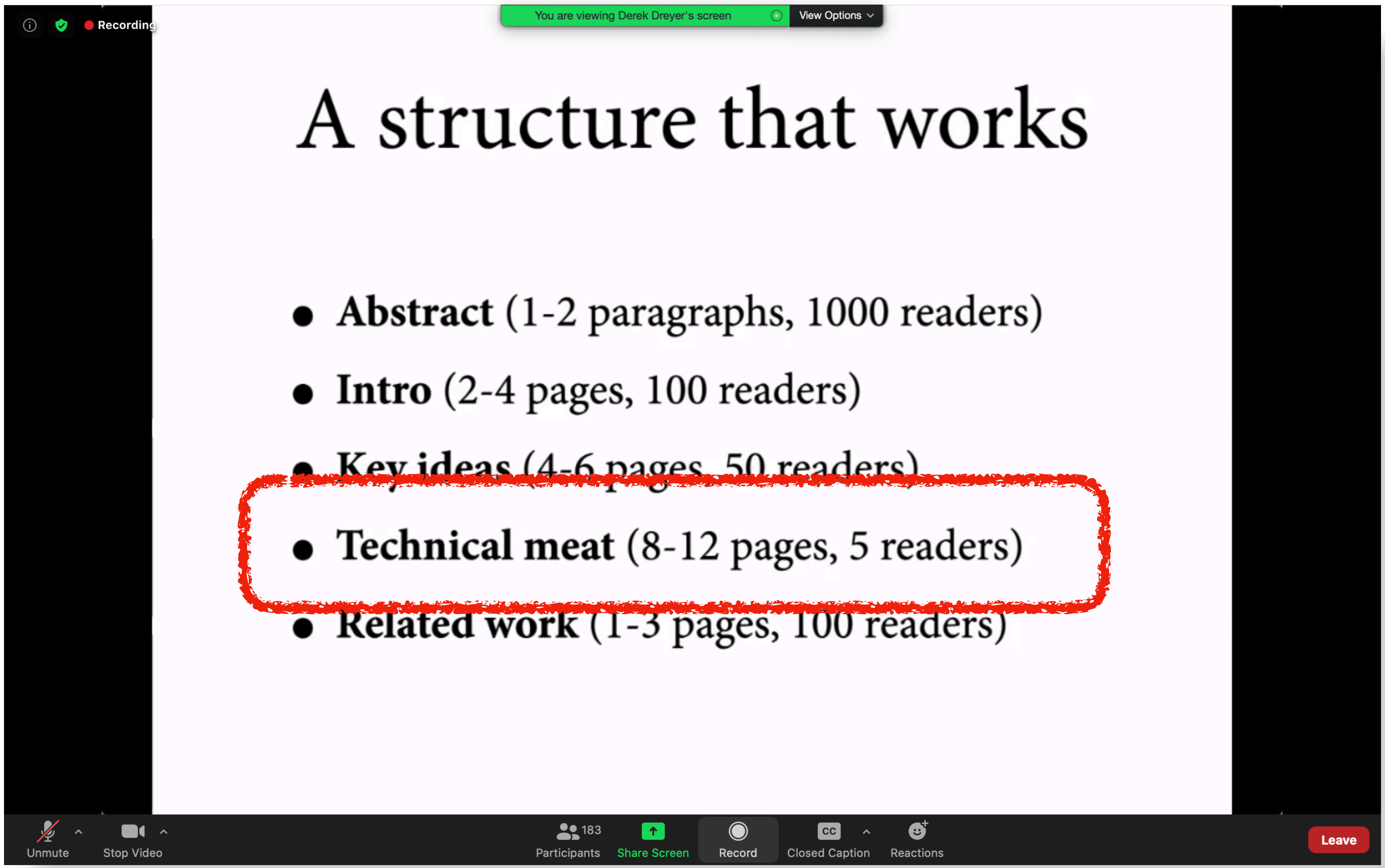
There are many different flavors:

- Evaluator
- Natural (aka big-step)
- Reduction (aka SOS, small-step)
- Abstract machine

What is an operational semantics *used for*?

- Specifying a programming language
- Communicating language design ideas
- Validating claims about languages
- Validating claims about type systems, etc
- Proving correctness of a compiler
- ...

From Derek's talk



Effects for Efficiency

Asymptotic Speedup with First-Class Control

DANIEL HILLERSTRÖM, The University of Edinburgh, UK
SAM LINDLEY, The University of Edinburgh and Imperial College London and Heriot-Watt University, UK
JOHN LONGLEY, The University of Edinburgh, UK

We study the fundamental efficiency of delimited control. Specifically, we show that effect handlers enable an asymptotic improvement in runtime complexity for a certain class of functions. We consider the *generic count* problem using a pure PCF-like base language λ_b and its extension with effect handlers λ_h. We show that λ_h admits an asymptotically more efficient implementation of generic count than any λ_b implementation. We also show that this efficiency gap remains when λ_b is extended with mutable state.

To our knowledge this result is the first of its kind for control operators.

CCS Concepts: • **Theory of computation → Lambda calculus**; **Abstract machines**; **Control primitives**.

Additional Key Words and Phrases: effect handlers, asymptotic complexity analysis, generic search

ACM Reference Format:
Daniel Hillerström, Sam Lindley, and John Longley. 2020. Effects for Efficiency: Asymptotic Speedup with First-Class Control. *Proc. ACM Program. Lang.* 4, ICFP, Article 100 (August 2020), 29 pages. https://doi.org/10.1145/3408982

1 INTRODUCTION

In today's programming languages we find a wealth of powerful constructs and features — exceptions, higher-order store, dynamic method dispatch, coroutines, explicit continuations, concurrency features, Lisp-style 'quote' and so on — which may be present or absent in various combinations in any given language. There are of course many important pragmatic and stylistic differences between languages, but here we are concerned with whether languages may differ more essentially in their expressive power, according to the selection of features they contain.

One can interpret this question in various ways. For instance, Felleisen [1991] considers the question of whether a language $\mathcal{L}$ admits a translation into a sublanguage $\mathcal{L}'$ in a way which respects not only the behaviour of programs but also aspects of their (global or local) syntactic structure. If the translation of some $\mathcal{L}$-program into $\mathcal{L}'$ requires a complete global restructuring, we may say that $\mathcal{L}'$ is in some way less expressive than $\mathcal{L}$. In the present paper, however, we have in mind even more fundamental expressivity differences that would not be bridged even if whole-program translations were admitted. These fall under two headings.

(1) *Computability*: Are there operations of a given type that are programmable in $\mathcal{L}$ but not expressible at all in $\mathcal{L}'$?

Authors' addresses: Daniel Hillerström, The University of Edinburgh, UK, daniel.hillerstrom@ed.ac.uk; Sam Lindley, The University of Edinburgh and Imperial College London and Heriot-Watt University, UK, sam.lindley@ed.ac.uk; John Longley, The University of Edinburgh, UK, jrl@staffmail.ed.ac.uk.

2475-1421/2020/8-ART100
https://doi.org/10.1145/3408982

Effects for Efficiency

Asymptotic Speedup with First-Class Control

We study the fundamental efficiency of delimited control. Specifically, we show that effect handlers enable an asymptotic improvement in runtime complexity for a certain class of functions. We consider the *generic count* problem using a pure PCF-like base language λ_b and its extension with effect handlers λ_h. We show that λ_h admits an asymptotically more efficient implementation of generic count than any λ_b implementation. We also show that this efficiency gap remains when λ_b is extended with mutable state.

Daniel Hillerström, Sam Lindley, and John Longley. 2020. Effects for Efficiency: Asymptotic Speedup with First-Class Control. *Proc. ACM Program. Lang.* 4, ICFP, Article 100 (August 2020), 29 pages. https://doi.org/10.1145/3408982

1 INTRODUCTION

In today's programming languages we find a wealth of powerful constructs and features — exceptions, higher-order store, dynamic method dispatch, coroutines, explicit continuations, concurrency features, Lisp-style 'quote' and so on — which may be present or absent in various combinations in any given language. There are of course many important pragmatic and stylistic differences between languages, but here we are concerned with whether languages may differ more essentially in their expressive power, according to the selection of features they contain.

One can interpret this question in various ways. For instance, Felleisen [1991] considers the question of whether a language $\mathcal{L}$ admits a translation into a sublanguage $\mathcal{L}'$ in a way which respects not only the behaviour of programs but also aspects of their (global or local) syntactic structure. If the translation of some $\mathcal{L}$-program into $\mathcal{L}'$ requires a complete global restructuring, we may say that $\mathcal{L}'$ is in some way less expressive than $\mathcal{L}$. In the present paper, however, we have in mind even more fundamental expressivity differences that would not be bridged even if whole-program translations were admitted. These fall under two headings.

(1) *Computability*: Are there operations of a given type that are programmable in $\mathcal{L}$ but not expressible at all in $\mathcal{L}'$?

Authors' addresses: Daniel Hillerström, The University of Edinburgh, UK, daniel.hillerstrom@ed.ac.uk; Sam Lindley, The University of Edinburgh and Imperial College London and Heriot-Watt University, UK, sam.lindley@ed.ac.uk; John Longley, The University of Edinburgh, UK, jrl@staffmail.ed.ac.uk.

2475-1421/2020/8-ART100
https://doi.org/10.1145/3408982

Effects for Efficiency

Asymptotic Speedup with First-Class Control

We study the fundamental efficiency of delimited control. Specifically, we show that effect handlers enable an asymptotic improvement in runtime complexity for a certain class of functions. We consider the *generic count* problem using a pure PCF-like base language λ_{b} and its extension with effect handlers λ_{h}. We show that λ_{h} admits an asymptotically more efficient implementation of generic count than any λ_{b} implementation. We also show that this efficiency gap remains when λ_{b} is extended with mutable state.

Daniel Hillerström, Sam Lindley, and John Longley. 2020. Effects for Efficiency: Asymptotic Speedup with First-Class Control. *Proc. ACM Program. Lang.* 4, ICFP, Article 100 (August 2020), 29 pages. https://doi.org/10.1145/3408982

1 INTRODUCTION

In today's programming langu
tions, higher-order store, dyna
features, Lisp-style 'quote' an
in any given language. There
between languages, but here w
in their expressive power, acc

One can interpret this ques
question of whether a langua
respects not only the behavio
structure. If the translation of
we may say that $\mathcal{L}'$ is in som
have in mind even more fund
whole-program translations w

(1) *Computability*: Are the
expressible at all in $\mathcal{L}'$?

$$
\begin{array}{lrl}
\textsc{S-App} & (\lambda x^A.\, M)V \rightsquigarrow & M[V/x] \\
\textsc{S-App-Rec} & (\mathbf{rec}\ f^A\ x.\, M)V \rightsquigarrow & M[(\mathbf{rec}\ f^A\ x.\, M)/f, V/x] \\
\textsc{S-Const} & c\ V \rightsquigarrow & \mathbf{return}\ (\ulcorner c \urcorner (V)) \\
\textsc{S-Split} & \mathbf{let}\ \langle x, y\rangle = \langle V, W\rangle\ \mathbf{in}\ N \rightsquigarrow & N[V/x, W/y] \\
\textsc{S-Case-inl} & \mathbf{case}\ (\mathbf{inl}\ V)^B\ \{\mathbf{inl}\ x \mapsto M; \mathbf{inr}\ y \mapsto N\} \rightsquigarrow & M[V/x] \\
\textsc{S-Case-inr} & \mathbf{case}\ (\mathbf{inr}\ V)^A\ \{\mathbf{inl}\ x \mapsto M; \mathbf{inr}\ y \mapsto N\} \rightsquigarrow & N[V/y] \\
\textsc{S-Let} & \mathbf{let}\ x \leftarrow \mathbf{return}\ V\ \mathbf{in}\ N \rightsquigarrow & N[V/x] \\
\textsc{S-Lift} & \mathcal{E}[M] \rightsquigarrow & \mathcal{E}[N], \qquad \text{if } M \rightsquigarrow N
\end{array}
$$

$$
\begin{array}{ll}
\textsc{S-Ret} & \mathbf{handle}\ (\mathbf{return}\ V)\ \mathbf{with}\ H \rightsquigarrow N[V/x], \qquad \text{where } H^{\text{val}} = \{\mathbf{val}\ x \mapsto N\} \\
\textsc{S-Op} & \mathbf{handle}\ \mathcal{E}[\mathbf{do}\ \ell\ V]\ \mathbf{with}\ H \rightsquigarrow N[V/p, (\lambda y.\mathbf{handle}\ \mathcal{E}[\mathbf{return}\ y]\ \mathbf{with}\ H)/r], \\
& \qquad \text{where } H^{\ell} = \{\ell\ p\ r \mapsto N\}
\end{array}
$$

Authors' addresses: Daniel Hillerströ
University of Edinburgh and Imperial
The University of Edinburgh, UK, jrl@

2475-1421/2020/8-ART100
https://doi.org/10.1145/3408982

$\mathcal{A}$rithmetic

- Syntax
- Semantics
  - Natural, big-step
  - Evaluator
  - Structured, small-step
  - Reduction
  - Standard reduction
  - Abstract machine

Syntax of $\mathcal{A}$

3 ways: 1/3

$$i \in \mathbb{Z} \Rightarrow i \in \mathcal{A}$$

$$e \in \mathcal{A} \Rightarrow \mathit{Pred}(e) \in \mathcal{A}$$

$$e \in \mathcal{A} \Rightarrow \mathit{Succ}(e) \in \mathcal{A}$$

$$e_1 \in \mathcal{A} \wedge e_2 \in \mathcal{A} \Rightarrow \mathit{Plus}(e_1, e_2) \in \mathcal{A}$$

$$e_1 \in \mathcal{A} \wedge e_2 \in \mathcal{A} \Rightarrow \mathit{Mult}(e_1, e_2) \in \mathcal{A}$$

Syntax of $\mathcal{A}$

3 ways: 2/3

$$
\begin{array}{llll}
\mathbb{Z} & i & ::= & \ldots \mid -1 \mid 0 \mid 1 \mid \ldots \\
\mathcal{A} & e & ::= & i \\
 & & \mid & \mathit{Pred}(e) \\
 & & \mid & \mathit{Succ}(e) \\
 & & \mid & \mathit{Plus}(e, e) \\
 & & \mid & \mathit{Mult}(e, e)
\end{array}
$$

Inference rules

$$\frac{H_1 \qquad H_2 \qquad \ldots \qquad H_n}{C}$$

Inference rules

$$\frac{H_1 \qquad H_2 \qquad \ldots \qquad H_n}{C}$$

$$H_1 \wedge H_2 \wedge \cdots \wedge H_n \Rightarrow C$$

Syntax of $\mathcal{A}$

3 ways: 3/3

$$\frac{i \in \mathbb{Z}}{i \in \mathcal{A}}\ (1) \qquad \frac{e \in \mathcal{A}}{\mathit{Pred}(e) \in \mathcal{A}}\ (2) \qquad \frac{e \in \mathcal{A}}{\mathit{Succ}(e) \in \mathcal{A}}\ (3)$$

$$\frac{e_1 \in \mathcal{A} \qquad e_2 \in \mathcal{A}}{\mathit{Plus}(e_1, e_2) \in \mathcal{A}}\ (4) \qquad \frac{e_1 \in \mathcal{A} \qquad e_2 \in \mathcal{A}}{\mathit{Mult}(e_1, e_2) \in \mathcal{A}}\ (5)$$

Proof of $\mathit{Plus}(4, \mathit{Succ}(2)) \in \mathcal{A}$

$$
\dfrac{\dfrac{4 \in \mathbb{Z}}{4 \in \mathcal{A}} \qquad \dfrac{\dfrac{2 \in \mathbb{Z}}{2 \in \mathcal{A}}}{\mathit{Succ}(2) \in \mathcal{A}}}{\mathit{Plus}(4, \mathit{Succ}(2)) \in \mathcal{A}}
$$

Natural semantics of $\mathcal{A}$

$$\boxed{\Downarrow \subseteq \mathcal{A} \times \mathbb{Z}}$$

$$Plus(4, Succ(2)) \Downarrow 7$$

Natural semantics of $\mathcal{A}$

$$\boxed{\Downarrow \subseteq \mathcal{A} \times \mathbb{Z}}$$

$$\frac{}{i \Downarrow i} \qquad \frac{e \Downarrow i}{\mathit{Pred}(e) \Downarrow i - 1} \qquad \frac{e \Downarrow i}{\mathit{Succ}(e) \Downarrow i + 1}$$

$$\frac{e_1 \Downarrow i_1 \qquad e_2 \Downarrow i_2}{\mathit{Plus}(e_1, e_2) \Downarrow i_1 + i_2} \qquad \frac{e_1 \Downarrow i_1 \qquad e_2 \Downarrow i_2}{\mathit{Mult}(e_1, e_2) \Downarrow i_1 \cdot i_2}$$

Natural semantics of $\mathcal{A}$

$$\frac{e_1 \Downarrow i_1 \qquad e_2 \Downarrow i_2}{\mathit{Plus}(e_1, e_2) \Downarrow i_1 + i_2}$$

$$\frac{e_1 \Downarrow i_1 \qquad e_2 \Downarrow i_2 \qquad i = i_1 + i_2}{\mathit{Plus}(e_1, e_2) \Downarrow i}$$

Proof of $Plus(4, Succ(2)) \Downarrow 7$

$$\dfrac{\dfrac{}{4 \Downarrow 4} \qquad \dfrac{\dfrac{}{2 \Downarrow 2}}{Succ(2) \Downarrow 3}}{Plus(4, Succ(2)) \Downarrow 7}$$

Evaluator semantics of $\mathcal{A}$

```
type arith = Int of int
           | Pred of arith
           | Succ of arith
           | Plus of arith * arith
           | Mult of arith * arith
let rec eval (e : arith) : int =
    match e with
      Int i -> i
    | Pred e -> (eval e) - 1
    | Succ e -> (eval e) + 1
    | Plus (e1, e2) -> (eval e1) + (eval e2)
    | Mult (e1, e2) -> (eval e1) * (eval e2)
```

$\Downarrow \subseteq \mathcal{A} \times \mathbb{Z}$

```
# eval (Plus (Int 4, Succ (Int 2)));;
- : int = 7
```

SOS semantics of $\mathcal{A}$

$$\boxed{\rightarrow \subseteq \mathcal{A} \times \mathcal{A}}$$

$$\sout{\Downarrow \subseteq \mathcal{A} \times \mathbb{Z}}$$

$$\mathit{Plus}(4, \mathit{Succ}(2)) \rightarrow \mathit{Plus}(4, 3)$$

$$\mathit{Plus}(4, 3) \rightarrow 7$$

each step has a proof

SOS semantics of $\mathcal{A}$

$$\boxed{\rightarrow \; \subseteq \mathcal{A} \times \mathcal{A}}$$

~~$\Downarrow \; \subseteq \mathcal{A} \times \mathbb{Z}$~~

$$Plus(4, Succ(2)) \rightarrow Plus(4, 3)$$

$$Plus(4, 3) \rightarrow 7$$

each step has a proof

SOS semantics of $\mathcal{A}$

$$\boxed{\rightarrow \subseteq \mathcal{A} \times \mathcal{A}}$$

Part 1: axioms

$$\frac{}{\mathit{Pred}(i) \;\rightarrow\; i - 1} \qquad \frac{}{\mathit{Succ}(i) \;\rightarrow\; i + 1}$$

$$\frac{}{\mathit{Plus}(i_1, i_2) \;\rightarrow\; i_1 + i_2} \qquad \frac{}{\mathit{Mult}(i_1, i_2) \;\rightarrow\; i_1 \cdot i_2}$$

SOS semantics of $\mathcal{A}$

$$\boxed{\rightarrow \subseteq \mathcal{A} \times \mathcal{A}}$$

Part 2: contexts

$$\frac{e \rightarrow e'}{\mathit{Pred}(e) \rightarrow \mathit{Pred}(e')} \qquad \frac{e \rightarrow e'}{\mathit{Succ}(e) \rightarrow \mathit{Succ}(e')} \qquad \frac{e_1 \rightarrow e_1'}{\mathit{Plus}(e_1, e_2) \rightarrow \mathit{Plus}(e_1', e_2)}$$

$$\frac{e_2 \rightarrow e_2'}{\mathit{Plus}(e_1, e_2) \rightarrow \mathit{Plus}(e_1, e_2')} \qquad \frac{e_1 \rightarrow e_1'}{\mathit{Mult}(e_1, e_2) \rightarrow \mathit{Mult}(e_1', e_2)} \qquad \frac{e_2 \rightarrow e_2'}{\mathit{Mult}(e_1, e_2) \rightarrow \mathit{Mult}(e_1, e_2')}$$

compatible closure

Proof of each step

$$\frac{\overline{Succ(2) \to 3}}{Plus(4, Succ(2)) \to Plus(4, 3)} \qquad \frac{}{Plus(4, 3) \to 7}$$

Different steps

$$\frac{\overline{\mathit{Succ}(4) \to 5}}{\mathit{Plus}(\mathit{Succ}(4), \mathit{Pred}(3)) \to \mathit{Plus}(5, \mathit{Pred}(3))}$$

$$\frac{\overline{\mathit{Pred}(3) \to 2}}{\mathit{Plus}(\mathit{Succ}(4), \mathit{Pred}(3)) \to \mathit{Plus}(\mathit{Succ}(4), 2)}$$

SOS semantics of $\mathcal{A}$

$$\boxed{\rightarrow^\star \subseteq \mathcal{A} \times \mathcal{A}}$$

$$\frac{e \rightarrow e'}{e \rightarrow^\star e'}$$

$$\frac{}{e \rightarrow^\star e}$$

reflexive closure

$$\frac{e \rightarrow^\star e' \qquad e' \rightarrow^\star e''}{e \rightarrow^\star e''}$$

transitive closure

Relating natural and SOS

Claim:

$$e \Downarrow i \iff e \rightarrow^{\star} i$$

Reduction semantics

Every proof of one-step reduction looks like:

$$\dfrac{\dfrac{\dfrac{}{e \to e'}}{\ldots e \ldots \to \ldots e' \ldots}}{\dfrac{\vdots}{\ldots(\ldots e \ldots)\ldots \to \ldots(\ldots e' \ldots)\ldots}}$$

Reduction semantics

Every proof of one-step reduction looks like:

$$\cfrac{\cfrac{\cfrac{}{Plus(2,3) \to 5}}{Succ(Plus(2,3)) \to Succ(5)}}{\vdots}$$

$$\overline{Mult(Succ(Plus(2,3)), Pred(4)) \to Mult(Succ(5), Pred(4))}$$

Reduction axioms

$$\boxed{\mathbf{a} \subseteq \mathcal{A} \times \mathcal{A}}$$

$$\frac{}{\mathit{Pred}(i)\ \mathbf{a}\ i-1} \qquad \frac{}{\mathit{Succ}(i)\ \mathbf{a}\ i+1}$$

$$\frac{}{\mathit{Plus}(i_1, i_2)\ \mathbf{a}\ i_1 + i_2} \qquad \frac{}{\mathit{Mult}(i_1, i_2)\ \mathbf{a}\ i_1 \cdot i_2}$$

Reduction semantics

Every proof of one-step reduction looks like:

$$
\dfrac{\dfrac{\dfrac{}{e \; \mathbf{a} \; e'}}{\ldots e \ldots \rightarrow \ldots e' \ldots}}{\vdots}
$$

$$
\overline{\ldots(\ldots e \ldots)\ldots \rightarrow \ldots(\ldots e' \ldots)\ldots}
$$

Reduction semantics

$$\begin{array}{lcll} \textit{Context} & \mathcal{C} & = & \square \\ & & \mid & \textit{Pred}(\mathcal{C}) \mid \textit{Succ}(\mathcal{C}) \\ & & \mid & \textit{Plus}(\mathcal{C}, e) \mid \textit{Plus}(e, \mathcal{C}) \\ & & \mid & \textit{Mult}(\mathcal{C}, e) \mid \textit{Mult}(e, \mathcal{C}) \end{array}$$

$$\mathcal{C}[e]$$

$$\dfrac{\dfrac{\dfrac{}{e \;\mathbf{a}\; e'}}{\ldots e \ldots \to \ldots e' \ldots}}{\dfrac{\vdots}{\ldots(\ldots e \ldots)\ldots \to \ldots(\ldots e' \ldots)\ldots}}$$

$$\frac{e \;\mathbf{a}\; e'}{\mathcal{C}[e] \to \mathcal{C}[e']}$$

Reduction semantics

$$\begin{array}{llll} \textit{Context} & \mathcal{C} & = & \square \\ & & \mid & \textit{Pred}(\mathcal{C}) \mid \textit{Succ}(\mathcal{C}) \\ & & \mid & \textit{Plus}(\mathcal{C}, e) \mid \textit{Plus}(e, \mathcal{C}) \\ & & \mid & \textit{Mult}(\mathcal{C}, e) \mid \textit{Mult}(e, \mathcal{C}) \end{array}$$

$$\mathcal{C}[e]$$

$$\dfrac{\dfrac{\dfrac{}{\textit{Plus}(2,3) \to 5}}{\textit{Succ}(\textit{Plus}(2,3)) \to \textit{Succ}(5)}}{\dfrac{\vdots}{\textit{Mult}(\textit{Succ}(\textit{Plus}(2,3)), \textit{Pred}(4)) \to \textit{Mult}(\textit{Succ}(5), \textit{Pred}(4))}}$$

$$\frac{\textit{Plus}(2,3)\ \mathbf{a}\ 5}{\mathcal{C}[\textit{Plus}(2,3)] \to \mathcal{C}[5], \text{ where } \mathcal{C} = \textit{Mult}(\textit{Succ}(\square), \textit{Pred}(4))}$$

Reduction semantics

$$\frac{e \mathbin{\mathbf{a}} e'}{\mathcal{C}[e] \rightarrow \mathcal{C}[e']}$$

Standard reductions

$$\frac{e \;\mathbf{a}\; e'}{e \longmapsto e'}$$

$$\frac{e_1 \longmapsto e_1'}{\mathit{Plus}(e_1, e_2) \longmapsto \mathit{Plus}(e_1', e_2)} \qquad \frac{e \longmapsto e'}{\mathit{Plus}(i, e) \longmapsto \mathit{Plus}(i, e')} \qquad \frac{e_1 \longmapsto e_1'}{\mathit{Mult}(e_1, e_2) \longmapsto \mathit{Mult}(e_1', e_2)}$$

$$\frac{e \longmapsto e'}{\mathit{Mult}(i, e) \longmapsto \mathit{Mult}(i, e')} \qquad \frac{e \longmapsto e'}{\mathit{Succ}(e) \longmapsto \mathit{Succ}(e')} \qquad \frac{e \longmapsto e'}{\mathit{Pred}(e) \longmapsto \mathit{Pred}(e')}$$

Standard reductions

$$\frac{e \ \mathbf{a} \ e'}{\mathcal{E}[e] \longmapsto \mathcal{E}[e']}$$

$$\begin{array}{llll}
\textit{EvalContext} & \mathcal{E} & = & \square \\
 & & \mid & \textit{Pred}(\mathcal{E}) \mid \textit{Succ}(\mathcal{E}) \\
 & & \mid & \textit{Plus}(\mathcal{E}, e) \mid \textit{Plus}(i, \mathcal{E}) \\
 & & \mid & \textit{Mult}(\mathcal{E}, e) \mid \textit{Mult}(i, \mathcal{E})
\end{array}$$

Relating reductions

Claim:

$$e \longmapsto^{\star} i \iff e \longrightarrow^{\star} i$$

Abstract (stack) machine

$$
\begin{array}{rrcl}
\textit{Frame} & \mathcal{F} & = & \textit{Pred}(\Box) \mid \textit{Succ}(\Box) \\
 & & \mid & \textit{Plus}(\Box, e) \mid \textit{Plus}(i, \Box) \\
 & & \mid & \textit{Mult}(\Box, e) \mid \textit{Mult}(i, \Box) \\
\textit{Stack} & \mathcal{S} & = & [\,] \mid \mathcal{F} :: \mathcal{S} \\
\\
\textit{Serious} & s & \in & \mathcal{A} \setminus \mathbb{Z}
\end{array}
$$

Abstract (stack) machine

$$\frac{e \; \mathbf{a} \; e'}{e, \mathcal{S} \rightsquigarrow e', \mathcal{S}}$$

reduce

Abstract (stack) machine

$$\frac{}{\mathit{Pred}(s), \mathcal{S} \rightsquigarrow s, \mathit{Pred}(\square) :: \mathcal{S}}$$

$$\frac{}{\mathit{Mult}(s, e), \mathcal{S} \rightsquigarrow s, \mathit{Mult}(\square, e) :: \mathcal{S}}$$

push

$$\frac{}{\mathit{Mult}(i, s), \mathcal{S} \rightsquigarrow s, \mathit{Mult}(i, \square) :: \mathcal{S}}$$

(Not showing similar rules for Succ, Plus)

Abstract (stack) machine

$$\frac{}{i, \mathit{Pred}(\square) :: \mathcal{S} \leadsto \mathit{Pred}(i), \mathcal{S}}$$

pop

$$\frac{}{i, \mathit{Mult}(\square, e) :: \mathcal{S} \leadsto \mathit{Mult}(i, e), \mathcal{S}}$$

$$\frac{}{i, \mathit{Mult}(e, \square) :: \mathcal{S} \leadsto \mathit{Mult}(e, i), \mathcal{S}}$$

(Not showing similar rules for Succ, Plus)

Relating reductions

Claim:

$$e \longmapsto^\star i \iff e, [\,] \leadsto^\star i, [\,]$$

Functions

$$
\begin{array}{rcl}
e & = & \ldots \\
& | & \mathit{App}(e, e) \\
& | & \mathit{Fun}(x, e) \\
& | & \mathit{Var}(x) \\
\\
x & = & \mathtt{x} \mid \mathtt{y} \mid \mathtt{z} \mid \ldots \\
\\
v & = & i \mid \mathit{Fun}(x, e)
\end{array}
$$

Substitution

$$
\begin{aligned}
\mathit{Var}(x')[e/x] &= \begin{cases} e, & \text{if } x = x' \\ \mathit{Var}(x'), & \text{otherwise} \end{cases} \\
\mathit{Succ}(e_0)[e/x] &= \mathit{Succ}(e_0[e/x]) \\
\mathit{Plus}(e_0, e_1)[e/x] &= \mathit{Plus}(e_0[e/x], e_1[e/x]) \\
&\;\vdots \qquad \vdots \\
\mathit{Fun}(x', e_0)[e/x] &= \ldots
\end{aligned}
$$

the tricky part

Natural semantics

$$\frac{e_0 \Downarrow \mathit{Fun}(x, e) \qquad e[e_1/x] \Downarrow v}{\mathit{App}(e_0, e_1) \Downarrow v}$$

call-by-*name*

Natural semantics

$$\frac{e_0 \Downarrow \mathit{Fun}(x, e) \qquad e_1 \Downarrow v_1 \qquad e[v_1/x] \Downarrow v}{\mathit{App}(e_0, e_1) \Downarrow v}$$

call-by-*value*

Reduction semantics

$$\frac{}{\mathit{App}(\mathit{Fun}(x, e), e')\ \boldsymbol{\beta}\ e[e'/x]}$$

$$\begin{array}{llll} \mathit{Context} & \mathcal{C} & = & \ldots \\ & & \mid & \mathit{Fun}(x, \mathcal{C}) \\ & & \mid & \mathit{App}(\mathcal{C}, e) \mid \mathit{App}(e, \mathcal{C}) \end{array}$$

$$\frac{e\ (\mathbf{a} \cup \boldsymbol{\beta})\ e'}{\mathcal{C}[e] \rightarrow \mathcal{C}[e']}$$

Reduction semantics

$$\frac{}{\mathit{App}(\mathit{Fun}(x, e), v) \; \boldsymbol{\beta}_{\mathbf{v}} \; e[v/x]}$$

$$\begin{array}{llll} \mathit{Context} & \mathcal{C} & = & \ldots \\ & & \mid & \mathit{Fun}(x, \mathcal{C}) \\ & & \mid & \mathit{App}(\mathcal{C}, e) \mid \mathit{App}(e, \mathcal{C}) \end{array}$$

$$\frac{e \; (\mathbf{a} \cup \boldsymbol{\beta}_{\mathbf{v}}) \; e'}{\mathcal{C}[e] \rightarrow_{\mathbf{v}} \mathcal{C}[e']}$$

Standard reductions

$$\frac{}{\mathit{App}(\mathit{Fun}(x, e), e')\ \boldsymbol{\beta}\ e[e'/x]}$$

~~$\mathit{Context}\ \ \mathcal{C} \ = \ \ldots \ \mid \ \mathit{Fun}(x, \mathcal{C}) \ \mid \ \mathit{App}(\mathcal{C}, e) \mid \mathit{App}(e, \mathcal{C})$~~

$$\begin{array}{llll} \mathit{EvalContext} & \mathcal{E} & = & \ldots \\ & & \mid & \mathit{App}(\mathcal{E}, e) \end{array}$$

$$\frac{e\ (\mathbf{a} \cup \boldsymbol{\beta})\ e'}{\mathcal{E}[e] \longmapsto \mathcal{E}[e']}$$

Standard reductions

$$\frac{}{\mathit{App}(\mathit{Fun}(x, e), v)\ \boldsymbol{\beta}_{\mathbf{v}}\ e[v/x]}$$

$$\begin{array}{llll} \mathit{Context} & \mathcal{C} & = & \ldots \\ & & \mid & \mathit{Fun}(x, \mathcal{C}) \\ & & \mid & \mathit{App}(\mathcal{C}, e) \mid \mathit{App}(e, \mathcal{C}) \end{array}$$

$$\begin{array}{llll} \mathit{EvalContext} & \mathcal{E} & = & \ldots \\ & & \mid & \mathit{App}(\mathcal{E}, e) \mid \mathit{App}(v, \mathcal{E}) \end{array}$$

$$\frac{e\ (\mathbf{a} \cup \boldsymbol{\beta}_{\mathbf{v}})\ e'}{\mathcal{E}[e] \longmapsto_{\mathbf{v}} \mathcal{E}[e']}$$

Abstract machine

$$\begin{array}{llll} \mathit{Frame} & \mathcal{F} & = & \ldots \\ & & \mid & \mathit{App}(\Box, e) \mid \mathit{App}(v, \Box) \end{array}$$

remove for CbN

Push, pop, reduce same as before *mutatis mutandis*

Exceptions

$$\begin{array}{rcl} e & = & \ldots \\ & | & \mathit{Raise}(e) \\ & | & \mathit{Try}(e, x, e) \end{array}$$

$$\begin{array}{llcl} \mathit{EvalContext} & \mathcal{E} & = & \ldots \\ & & | & \mathit{Raise}(\mathcal{E}) \\ & & | & \mathit{Try}(\mathcal{E}, x, e) \end{array}$$

$$\begin{array}{llcl} \mathit{TryContext} & \mathcal{T} & \in & \mathcal{E} \setminus \mathit{Try}(\mathcal{E}, x, e) \end{array}$$

Exceptions

$$\frac{}{\mathit{Try}(v, x, e) \ \boldsymbol{\tau} \ v}$$

$$\frac{}{\mathit{Try}(\mathcal{T}[\mathit{Raise}(v)], x, e) \ \boldsymbol{\tau} \ e[v/x]}$$

$$\frac{e \ (\mathbf{a} \cup \boldsymbol{\beta} \cup \boldsymbol{\tau}) \ e'}{\mathcal{E}[e] \longmapsto \mathcal{E}[e']}$$

Call/cc

$$\begin{array}{rcl} e & = & \ldots \\ & | & \mathit{Callcc}(x, e) \mid \mathit{Halt}(e) \\ \mathcal{E} & = & \ldots \\ & | & \mathit{Halt}(\mathcal{E}) \end{array}$$

$$\frac{}{\mathcal{E}[\mathit{Halt}(v)] \longmapsto v}$$

$$\frac{}{\mathcal{E}[\mathit{Callcc}(x, e)] \longmapsto e[\mathit{Fun}(x', \mathit{Halt}(\mathcal{E}[x']))/x]}$$

Operational semantics: A method of defining the meaning of programs by describing the actions carried out during execution.

Useful for:

- Specifying a PL
- Communicating ideas
- Validating claims
- ...

What you've seen:

- Syntax
- Semantics
  - Natural, big-step
  - Evaluator
  - Structured, small-step
  - Reduction
  - Standard reduction
  - Abstract machine

Effects for Efficiency

Asymptotic Speedup with First-Class Control

DANIEL HILLERSTRÖM, The University of Edinburgh, UK
SAM LINDLEY, The University of Edinburgh and Imperial College London and Heriot-Watt University, UK
JOHN LONGLEY, The University of Edinburgh, UK

$$
\begin{array}{ll}
\textsc{S-App} & (\lambda x^A.\, M)V \rightsquigarrow M[V/x] \\
\textsc{S-App-Rec} & (\mathbf{rec}\ f^A\ x.\, M)V \rightsquigarrow M[(\mathbf{rec}\ f^A\ x.\, M)/f, V/x] \\
\textsc{S-Const} & c\ V \rightsquigarrow \mathbf{return}\ (\ulcorner c \urcorner (V)) \\
\textsc{S-Split} & \mathbf{let}\ \langle x, y\rangle = \langle V, W\rangle\ \mathbf{in}\ N \rightsquigarrow N[V/x, W/y] \\
\textsc{S-Case-inl} & \mathbf{case}\ (\mathbf{inl}\ V)^B\ \{\mathbf{inl}\ x \mapsto M; \mathbf{inr}\ y \mapsto N\} \rightsquigarrow M[V/x] \\
\textsc{S-Case-inr} & \mathbf{case}\ (\mathbf{inr}\ V)^A\ \{\mathbf{inl}\ x \mapsto M; \mathbf{inr}\ y \mapsto N\} \rightsquigarrow N[V/y] \\
\textsc{S-Let} & \mathbf{let}\ x \leftarrow \mathbf{return}\ V\ \mathbf{in}\ N \rightsquigarrow N[V/x] \\
\textsc{S-Lift} & \mathcal{E}[M] \rightsquigarrow \mathcal{E}[N], \qquad \text{if } M \rightsquigarrow N
\end{array}
$$

$$
\begin{array}{ll}
\textsc{S-Ret} & \mathbf{handle}\ (\mathbf{return}\ V)\ \mathbf{with}\ H \rightsquigarrow N[V/x], \qquad \text{where } H^{\mathrm{val}} = \{\mathbf{val}\ x \mapsto N\} \\
\textsc{S-Op} & \mathbf{handle}\ \mathcal{E}[\mathbf{do}\ \ell\ V]\ \mathbf{with}\ H \rightsquigarrow N[V/p,\ (\lambda y.\mathbf{handle}\ \mathcal{E}[\mathbf{return}\ y]\ \mathbf{with}\ H)/r], \\
& \qquad \text{where } H^{\ell} = \{\ell\ p\ r \mapsto N\}
\end{array}
$$

Effects for Efficiency

Asymptotic Speedup with First-Class Control

DANIEL HILLERSTRÖM, The University of Edinburgh, UK
SAM LINDLEY, The University of Edinburgh and Imperial College London and Heriot-Watt University, UK
JOHN LONGLEY, The University of Edinburgh, UK

$$
\begin{array}{ll}
\textsc{S-App} & (\lambda x^A.\, M)V \leadsto M[V/x] \\
\textsc{S-App-Rec} & (\mathbf{rec}\ f^A\ x.\, M)V \leadsto M[(\mathbf{rec}\ f^A\ x.\, M)/f, V/x] \\
\textsc{S-Const} & c\ V \leadsto \mathbf{return}\ (\ulcorner c \urcorner (V)) \\
\textsc{S-Split} & \mathbf{let}\ \langle x, y\rangle = \langle V, W\rangle\ \mathbf{in}\ N \leadsto N[V/x, W/y] \\
\textsc{S-Case-inl} & \mathbf{case}\ (\mathbf{inl}\ V)^B\ \{\mathbf{inl}\ x \mapsto M; \mathbf{inr}\ y \mapsto N\} \leadsto M[V/x] \\
\textsc{S-Case-inr} & \mathbf{case}\ (\mathbf{inr}\ V)^A\ \{\mathbf{inl}\ x \mapsto M; \mathbf{inr}\ y \mapsto N\} \leadsto N[V/y] \\
\textsc{S-Let} & \mathbf{let}\ x \leftarrow \mathbf{return}\ V\ \mathbf{in}\ N \leadsto N[V/x] \\
\textsc{S-Lift} & \mathcal{E}[M] \leadsto \mathcal{E}[N], \qquad \text{if } M \leadsto N
\end{array}
$$

$$
\begin{array}{ll}
\textsc{S-Ret} & \mathbf{handle}\ (\mathbf{return}\ V)\ \mathbf{with}\ H \leadsto N[V/x], \qquad \text{where } H^{\mathrm{val}} = \{\mathbf{val}\ x \mapsto N\} \\
\textsc{S-Op} & \mathbf{handle}\ \mathcal{E}[\mathbf{do}\ \ell\ V]\ \mathbf{with}\ H \leadsto N[V/p, (\lambda y.\mathbf{handle}\ \mathcal{E}[\mathbf{return}\ y]\ \mathbf{with}\ H)/r], \\
& \qquad \text{where } H^{\ell} = \{\ell\ p\ r \mapsto N\}
\end{array}
$$