Advanced Topics in Programming Languages: Equation List

Josh Felmeden

January 5, 2022

Contents

1 Structural Rules **3**
1.1 Inversion Lemma . . . 3
1.2 Weakening . . . 3
1.3 Substitution . . . 3

2 Type Safety **4**
2.1 Preservation . . . 4
2.2 Progress . . . 4

3 Judgements **5**
3.1 Statics . . . 5
3.2 Dynamics . . . 5

4 Simply-Typed Lambda Calculus **6**
4.1 Products . . . 6
4.1.1 Syntax . . . 6
4.1.2 Statics . . . 6
4.1.3 Dynamics . . . 6
4.2 Sums . . . 7
4.2.1 Syntax . . . 7
4.2.2 Statics . . . 7
4.2.3 Dynamics . . . 7
4.3 Functions . . . 8
4.3.1 Syntax . . . 8
4.3.2 Statics . . . 8
4.3.3 Dynamics . . . 8

5 Programming Computable Functions (PCF) **9**
5.1 Syntax . . . 9
5.2 Statics . . . 9
5.3 Dynamics . . . 10

6 Call by Value/Name **11**
6.1 Process order . . . 11

7 Store **12**
7.1 Syntax . . . 12
7.2 Statics . . . 12
7.3 Transitions . . . 13

1 Structural Rules

1.1 Inversion Lemma

Lemma 1 (Inversion)

Suppose $\Gamma \vdash e : \tau$.

1. If $e =$ plus$(e_1; e_2)$ then it must be that
   - $\tau =$ Num
   - $\Gamma \vdash e_1 :$ Num
   - $\Gamma \vdash e_2 :$ Num

You basically prove this lemma by saying look at the rules there can't be another way. The lemma can also be shown by induction on the typing derivation.

1.2 Weakening

Suppose that $x : \sigma \vdash e : \tau$ (e computes a value of type τ if x is of type σ). It is fair to say that for any **fresh variable** y (a variable that doesn't already appear in term e), the typing judgement $x : \sigma, y : \rho \vdash e : \tau$ should also hold no matter what type ρ is. Essentially, assuming random free variables that are unused should not influence the type of a program. This is called **weakening.** We state and prove by induction on the typing derivation that:

Lemma 2 (Weakening)

If $\Gamma \vdash e : \tau$ and x is fresh then $\Gamma, x : \sigma \vdash e : \tau$

1.3 Substitution

Lemma 3 (Substitution)

If $\Gamma \vdash e : \tau$ and $\Gamma, x : \tau \vdash u : \sigma$ then $\Gamma \vdash u[e/x] : \sigma$

2 Type Safety

Theorem 1 (Type safety)

1. (Preservation) If $\vdash e : \tau$ and $e \mapsto e'$ then $\vdash e' : \tau$
2. (Progress) If $\vdash e : \tau$ then either e val or $e \mapsto e'$ for some e'

2.1 Preservation

Preservation is the statement that types are preserved under evaluation. This is a central **safety** property of type systems: it shows that a step-by-step computation preserves the kind of value that is being computed. We perform this on dynamics.

Theorem 2 (Preservation)

If $\vdash e : \tau$ and $e \mapsto e'$ then $\vdash e' : \tau$
Proof. Using $\vdash e : \tau$, prove that $\vdash e' : \tau$ and if proving some rule $b \longmapsto b'$, also prove $\vdash b' : \tau$.
Example: By induction on the derivation of $e \mapsto e'$. We show the most diffcult case, namely that of D-Let: Suppose that the reduction $e \mapsto e'$ is of the form

$$\frac{}{\text{let}(e_1; x, e_2) \mapsto e_2[e_1/x]}\text{D-Let}$$

We know that $\vdash$ let$(e_1; x.e_2) : \tau$. By **inversion** there must exist σ such that $\vdash e_1 : \sigma$ and $x : \sigma \vdash e_2 : \tau$. By the **substitution lemma**, we obtain $\vdash e_1[e_1/x] : \tau$.

2.2 Progress

Progress is the statement that if a well-typed program is not done computing (aka: is a value), then there must be a step of computation it may take. We perform this on statics.

Lemma 4 (Canonical Forms)

Suppose e val:
1. If $\vdash e$: Num then $e =$ num$[n]$ for some $n \in \mathbb{N}$
2. If $\vdash e$: Str then $e =$ str$[s]$ for some $s \in \Sigma^*$

Theorem 4 (Progress)

If $\vdash e : \tau$ then either e val or $e \mapsto e'$ for some e'
Proof. By induction on the derivation of $\vdash e : \tau$.

3 Judgements

3.1 Statics

$$\frac{}{\Gamma, x : \sigma \vdash x : \sigma}\ \textsc{Var}$$

$$\frac{n \in \mathbb{N}}{\Gamma \vdash \mathsf{num}[n] : \mathsf{Num}}\ \textsc{Num}$$

$$\frac{s \in \Sigma^*}{\Gamma \vdash \mathsf{str}[s] : \mathsf{Str}}\ \textsc{Str}$$

$$\frac{\Gamma \vdash e_1 : \mathsf{Num} \qquad \Gamma \vdash e_2 : \mathsf{Num}}{\Gamma \vdash \mathsf{plus}(e_1; e_2) : \mathsf{Num}}\ \textsc{Plus}$$

$$\frac{\Gamma \vdash e_1 : \mathsf{Num} \qquad \Gamma \vdash e_2 : \mathsf{Num}}{\Gamma \vdash \mathsf{times}(e_1; e_2) : \mathsf{Num}}\ \textsc{Times}$$

$$\frac{\Gamma \vdash e_1 : \mathsf{Str} \qquad \Gamma \vdash e_2 : \mathsf{Str}}{\Gamma \vdash \mathsf{cat}(e_1; e_2) : \mathsf{Str}}\ \textsc{Cat}$$

$$\frac{\Gamma \vdash e : \mathsf{Str}}{\Gamma \vdash \mathsf{len}(e) : \mathsf{Num}}\ \textsc{Len}$$

$$\frac{\Gamma \vdash e_1 : \sigma_1 \qquad \Gamma, x : \sigma_1 \vdash e_2 : \sigma_2}{\Gamma \vdash \mathsf{let}(e_1; x.\, e_2) : \sigma_2}\ \textsc{Let}$$

3.2 Dynamics

$$\frac{n_1 + n_2 = n}{\mathsf{plus}(\mathsf{num}[n_1]; \mathsf{num}[n_2]) \longmapsto \mathsf{num}[n]}\ \textsc{D-Plus}$$

$$\frac{e_1 \longmapsto e_1'}{\mathsf{plus}(e_1; e_2) \longmapsto \mathsf{plus}(e_1'; e_2)}\ \textsc{D-Plus-1}$$

$$\frac{e_1\ \mathsf{val} \qquad e_2 \longmapsto e_2'}{\mathsf{plus}(e_1; e_2) \longmapsto \mathsf{plus}(e_1; e_2')}\ \textsc{D-Plus-2}$$

$$\frac{s_1 \mathbin{+\!\!+} s_2 = s}{\mathsf{cat}(\mathsf{str}[s_1]; \mathsf{str}[s_2]) \longmapsto \mathsf{str}[s]}\ \textsc{D-Cat}$$

$$\frac{e_1 \longmapsto e_1'}{\mathsf{cat}(e_1; e_2) \longmapsto \mathsf{cat}(e_1'; e_2)}\ \textsc{D-Cat-1}$$

$$\frac{e_1\ \mathsf{val} \qquad e_2 \longmapsto e_2'}{\mathsf{cat}(e_1; e_2) \longmapsto \mathsf{cat}(e_1; e_2')}\ \textsc{D-Cat-2}$$

$$\frac{|s| = n}{\mathsf{len}(\mathsf{str}[s]) \longmapsto \mathsf{num}[n]}\ \textsc{D-Len}$$

$$\frac{e \longmapsto e}{\mathsf{len}(e) \longmapsto \mathsf{len}(e')}\ \textsc{D-Len-1}$$

$$\frac{}{\mathsf{let}(e_1; x.\, e_2) \longmapsto e_2[e_1/x]}\ \textsc{D-Let}$$

$$\frac{}{e \longmapsto^* e}\ \textsc{D-Multi-Refl}$$

$$\frac{e \longmapsto e' \qquad e' \longmapsto^* e''}{e \longmapsto^* e''}\ \textsc{D-Multi-Step}$$

4 Simply-Typed Lambda Calculus

4.1 Products

4.1.1 Syntax

$$
\begin{array}{llll}
\text{types} & \tau & ::= & \dots \\
 & & & \tau_1 \times \tau_2 \quad \text{product type} \\
 & & & \mathbf{1} \quad \text{unit type} \\
\text{pre-terms} & e & ::= & \dots \\
 & & & \langle e_1, e_2 \rangle \quad \text{pair constructor} \\
 & & & \pi_1(e) \quad \text{first projection} \\
 & & & \pi_2(e) \quad \text{second projection}
\end{array}
$$

4.1.2 Statics

$$
\textsc{Unit}\ \frac{}{\Gamma \vdash \langle\rangle : \mathbf{1}}
\qquad
\textsc{Prod}\ \frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \langle e_1, e_2 \rangle : \tau_1 \times \tau_2}
\qquad
\textsc{Proj-1}\ \frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \pi_1(e) : \tau_1}
\qquad
\textsc{Proj-2}\ \frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \pi_2(e) : \tau_2}
$$

4.1.3 Dynamics

$$
\textsc{Val-Unit}\ \frac{}{\langle\rangle\ \mathsf{val}}
\qquad
\textsc{Val-Pair}\ \frac{}{\langle e_1, e_2 \rangle\ \mathsf{val}}
\qquad
\textsc{D-Proj-Tuple-1}\ \frac{}{\pi_1(\langle e_1, e_2 \rangle) \longmapsto e_1}
\qquad
\textsc{D-Proj-Tuple-2}\ \frac{}{\pi_1(\langle e_1, e_2 \rangle) \longmapsto e_2}
$$

$$
\textsc{D-Proj-1}\ \frac{e \longmapsto e'}{\pi_1(e) \longmapsto \pi_1(e')}
\qquad
\textsc{D-Proj-2}\ \frac{e \longmapsto e'}{\pi_2(e) \longmapsto \pi_2(e')}
$$

4.2 Sums

4.2.1 Syntax

types	τ	$::=$	$\ldots$	
			$\tau_1 + \tau_2$	sum type
			$\mathbf{0}$	void type
pre-terms	e	$::=$	$\ldots$	
			$\mathsf{inl}(e)$	left injection
			$\mathsf{inr}(e)$	right injection
			$\mathsf{case}(e; x.\, e_1; y.\, e_2)$	case analysis

4.2.2 Statics

Abort
$$\frac{\Gamma \vdash e : \mathbf{0}}{\Gamma \vdash \mathsf{abort}(e) : \tau}$$

Inl
$$\frac{\Gamma \vdash e : \tau_1}{\Gamma \vdash \mathsf{inl}(e) : \tau_1 + \tau_2}$$

Inr
$$\frac{\Gamma \vdash e : \tau_2}{\Gamma \vdash \mathsf{inr}(e) : \tau_1 + \tau_2}$$

Case
$$\frac{\Gamma \vdash e : \tau_1 + \tau_2 \qquad \Gamma, x : \tau_1 \vdash e_1 : \tau \qquad \Gamma, y : \tau_2 \vdash e_2 : \tau}{\Gamma \vdash \mathsf{case}(e; x.\, e_1; y.\, e_2) : \tau}$$

4.2.3 Dynamics

Val-Inl
$$\frac{}{\mathsf{inl}(e)\ \mathsf{val}}$$

Val-Inr
$$\frac{}{\mathsf{inr}(e)\ \mathsf{val}}$$

D-Abort-1
$$\frac{e \longmapsto e'}{\mathsf{abort}(e) \longmapsto \mathsf{abort}(e')}$$

D-Case-Inl
$$\frac{}{\mathsf{case}(\mathsf{inl}(e); x.\, e_1; y.\, e_2) \longmapsto e_1[e/x]}$$

D-Case-Inr
$$\frac{}{\mathsf{case}(\mathsf{inr}(e); x.\, e_1; y.\, e_2) \longmapsto e_2[e/y]}$$

D-Case-1
$$\frac{e \longmapsto e'}{\mathsf{case}(e; x.\, e_1; y.\, e_2) \longmapsto \mathsf{case}(e'; x.\, e_1; y.\, e_2)}$$

4.3 Functions

4.3.1 Syntax

types	τ	$::=$	$\dots$	
			$\tau_1 \to \tau_2$	function type
pre-terms	e	$::=$	$\dots$	
			$\lambda x : \tau.\, e$	abstraction
			$e_1(e_2)$	application

4.3.2 Statics

$$\textsc{Lam}\quad \frac{\Gamma, x : \sigma \vdash e : \tau}{\Gamma \vdash \lambda x : \sigma.\, e : \sigma \to \tau}$$

$$\textsc{App}\quad \frac{\Gamma \vdash e_1 : \sigma \to \tau \qquad \Gamma \vdash e_2 : \sigma}{\Gamma \vdash e_1(e_2) : \tau}$$

4.3.3 Dynamics

$$\textsc{Val-Lam}\quad \frac{}{\lambda x : \tau.\, e \ \mathsf{val}}$$

$$\textsc{D-App-1}\quad \frac{e_1 \longmapsto e_1'}{e_1(e_2) \longmapsto e_1'(e_2)}$$

$$\textsc{D-Beta}\quad \frac{}{(\lambda x : \tau.\, e_1)(e_2) \longmapsto e_1[e_2/x]}$$

5 Programming Computable Functions (PCF)

5.1 Syntax

types	τ	$::=$	Nat	natural numbers
			$\tau_1 \rightharpoonup \tau_2$	(partial) function type
pre-terms	e	$::=$	x	variables
			zero	zero
			$\mathsf{succ}(e)$	successor
			$\mathsf{ifz}(e; e_0; x.\, e_1)$	zero test
			$\lambda x : \tau.\, e$	abstraction
			$e_1(e_2)$	application
			$\mathsf{fix}(x : \tau.\, e)$	fixed point

5.2 Statics

$$\textsc{Var}\quad \frac{}{\Gamma, x : \sigma \vdash x : \sigma}$$

$$\textsc{Zero}\quad \frac{}{\Gamma \vdash \mathsf{zero} : \mathsf{Nat}}$$

$$\textsc{Succ}\quad \frac{\Gamma \vdash e : \mathsf{Nat}}{\Gamma \vdash \mathsf{succ}(e) : \mathsf{Nat}}$$

$$\textsc{Lam}\quad \frac{\Gamma, x : \sigma \vdash e : \tau}{\Gamma \vdash \lambda x : \sigma.\, e : \sigma \rightharpoonup \tau}$$

$$\textsc{App}\quad \frac{\Gamma \vdash e_1 : \sigma \rightharpoonup \tau \qquad \Gamma \vdash e_2 : \sigma}{\Gamma \vdash e_1(e_2) : \tau}$$

$$\textsc{IfZero}\quad \frac{\Gamma \vdash e : \mathsf{Nat} \qquad \Gamma \vdash e_0 : \tau \qquad \Gamma, x : \mathsf{Nat} \vdash e_1 : \tau}{\Gamma \vdash \mathsf{ifz}(e; e_0; x.\, e_1) : \tau}$$

$$\textsc{Fix}\quad \frac{\Gamma, x : \tau \vdash e : \tau}{\Gamma \vdash \mathsf{fix}(x : \tau.\, e) : \tau}$$

5.3 Dynamics

$$\frac{}{\mathsf{zero}\ \mathsf{val}}\ \textsc{Val-Zero} \qquad \frac{e\ \mathsf{val}}{\mathsf{succ}(e)\ \mathsf{val}}\ \textsc{Val-Succ} \qquad \frac{}{\lambda x:\tau.\,e\ \mathsf{val}}\ \textsc{Val-Lam} \qquad \frac{e \longmapsto e'}{\mathsf{succ}(e) \longmapsto \mathsf{succ}(e')}\ \textsc{D-Succ}$$

$$\frac{e_1 \longmapsto e_1'}{e_1(e_2) \longmapsto e_1'(e_2)}\ \textsc{D-App-1} \qquad \frac{}{(\lambda x:\tau.\,e_1)(e_2) \longmapsto e_1[e_2/x]}\ \textsc{D-Beta}$$

$$\frac{}{\mathsf{fix}(x:\tau.\,e) \longmapsto e[\mathsf{fix}(x:\tau.\,e)/x]}\ \textsc{D-Fix} \qquad \frac{e \longmapsto e'}{\mathsf{ifz}(e;e_0;x.\,e_1) \longmapsto \mathsf{ifz}(e';e_0;x.\,e_1)}\ \textsc{D-Ifz-1}$$

$$\frac{}{\mathsf{ifz}(\mathsf{zero};e_0;x.\,e_1) \longmapsto e_0}\ \textsc{D-Ifz-Zero} \qquad \frac{\mathsf{succ}(e)\ \mathsf{val}}{\mathsf{ifz}(\mathsf{succ}(e);e_0;x.\,e_1) \longmapsto e_1[e/x]}\ \textsc{D-Ifz-Succ}$$

6 Call by Value/Name

$$\textsc{Val-Inl}\ \dfrac{v\ \mathsf{val}}{\mathsf{inl}(v)\ \mathsf{val}} \qquad \textsc{Val-Inr}\ \dfrac{v\ \mathsf{val}}{\mathsf{inr}(v)\ \mathsf{val}} \qquad \textsc{Val-Lam}\ \dfrac{}{\lambda x:\tau.\,e\ \mathsf{val}} \qquad \textsc{D-Let}\ \dfrac{v\ \mathsf{val}}{\mathsf{let}(v;x.\,e_2) \longmapsto e_2[v/x]}$$

$$\textsc{D-Inl}\ \dfrac{e \longmapsto e'}{\mathsf{inl}(e) \longmapsto \mathsf{inl}(e')} \qquad \textsc{D-Inr}\ \dfrac{e \longmapsto e'}{\mathsf{inr}(e) \longmapsto \mathsf{inr}(e')} \qquad \textsc{D-Case-Inl}\ \dfrac{v\ \mathsf{val}}{\mathsf{case}(\mathsf{inl}(v);x.\,e_1;y.\,e_2) \longmapsto e_1[v/x]}$$

$$\textsc{D-Case-Inr}\ \dfrac{v\ \mathsf{val}}{\mathsf{case}(\mathsf{inr}(v);x.\,e_1;y.\,e_2) \longmapsto e_2[v/y]} \qquad \textsc{D-Case-1}\ \dfrac{e \longmapsto e'}{\mathsf{case}(e;x.\,e_1;y.\,e_2) \longmapsto \mathsf{case}(e';x.\,e_1;y.\,e_2)}$$

$$\textsc{D-App-1}\ \dfrac{e_1 \longmapsto e_1'}{e_1(e_2) \longmapsto e_1'(e_2)} \qquad \textsc{D-App-2}\ \dfrac{e_1\ \mathsf{val} \qquad e_2 \longmapsto e_2'}{e_1(e_2) \longmapsto e_1(e_2')} \qquad \textsc{D-Beta}\ \dfrac{v\ \mathsf{val}}{(\lambda x:\tau.\,e)(v) \longmapsto e[v/x]}$$

6.1 Process order

In CBV:

$$\begin{aligned}
(\lambda x:\mathsf{Num}.\,\mathsf{plus}(x;x))(\underline{\mathsf{print}(\mathsf{hi};\mathsf{num}[1])}) &\xmapsto{\mathsf{hi}}_{\mathsf{v}} \underline{(\lambda x:\mathsf{Num}.\,\mathsf{plus}(x;x))(\mathsf{num}[1])} \\
&\xmapsto{\varepsilon}_{\mathsf{v}} \mathsf{plus}(\mathsf{num}[1];\mathsf{num}[1]) \\
&\xmapsto{\varepsilon}_{\mathsf{v}} \mathsf{num}[2]
\end{aligned}$$

In contrast, in CBN:

$$\begin{aligned}
\underline{(\lambda x:\mathsf{Num}.\,\mathsf{plus}(x;x))(\mathsf{print}(\mathsf{hi};\mathsf{num}[1]))} &\xmapsto{\varepsilon}_{\mathsf{n}} \mathsf{plus}(\underline{\mathsf{print}(\mathsf{hi};\mathsf{num}[1])};\mathsf{print}(\mathsf{hi};\mathsf{num}[1])) \\
&\xmapsto{\mathsf{hi}}_{\mathsf{n}} \mathsf{plus}(\mathsf{num}[1];\underline{\mathsf{print}(\mathsf{hi};\mathsf{num}[1])}) \\
&\xmapsto{\mathsf{hi}}_{\mathsf{n}} \mathsf{plus}(\mathsf{num}[1];\mathsf{num}[1]) \\
&\xmapsto{\varepsilon}_{\mathsf{n}} \mathsf{num}[2]
\end{aligned}$$

7 Store

7.1 Syntax

types	τ	$::=$	Nat		natural numbers
			$\tau_1 \rightharpoonup \tau_2$		(partial) function type
			Cmd		unevaluated commands
pre-terms	e	$::=$	$\ldots$		
			$\mathsf{cmd}(m)$		unevaluated command
pre-commands	m	$::=$	$\mathsf{ret}(e)$	$\mathsf{ret}\ e$	return value
			$\mathsf{bnd}(e; x.\,m)$	$\mathsf{bind}\ x \leftarrow e; m$	sequence
			$\mathsf{dcl}(e; a.\,m)$	$\mathsf{decl}\ a := e\ \mathsf{in}\ m$	allocate
			$\mathsf{get}[a]$	$@\,a$	fetch location contents
			$\mathsf{set}[a](e)$	$a := e$	set location contents

7.2 Statics

$$\textsc{Cmd}\ \frac{\Gamma \vdash_\Sigma m\ \mathsf{ok}}{\Gamma \vdash_\Sigma \mathsf{cmd}(m) : \mathsf{Cmd}} \qquad \textsc{Ret}\ \frac{\Gamma \vdash_\Sigma e : \mathsf{Nat}}{\Gamma \vdash_\Sigma \mathsf{ret}(e)\ \mathsf{ok}} \qquad \textsc{Bind}\ \frac{\Gamma \vdash_\Sigma e : \mathsf{Cmd} \qquad \Gamma, x : \mathsf{Nat} \vdash_\Sigma m\ \mathsf{ok}}{\Gamma \vdash_\Sigma \mathsf{bnd}(e; x.\,m)\ \mathsf{ok}}$$

$$\textsc{Decl}\ \frac{\Gamma \vdash_\Sigma e : \mathsf{Nat} \qquad \Gamma \vdash_{\Sigma,a} m\ \mathsf{ok}}{\Gamma \vdash_\Sigma \mathsf{dcl}(e; a.\,m)\ \mathsf{ok}} \qquad \textsc{Fetch}\ \frac{}{\Gamma \vdash_{\Sigma,a} \mathsf{get}[a]\ \mathsf{ok}} \qquad \textsc{Assign}\ \frac{\Gamma \vdash_{\Sigma,a} e : \mathsf{Nat}}{\Gamma \vdash_{\Sigma,a} \mathsf{set}[a](e)\ \mathsf{ok}}$$

7.3 Transitions

D-Get

$$\frac{}{\mathsf{get}[a] \parallel \mu \otimes \{a \hookrightarrow e\} \longmapsto_{\Sigma,a} \mathsf{ret}(e) \parallel \mu \otimes \{a \hookrightarrow e\}}$$

D-Set-1

$$\frac{e \longmapsto e'}{\mathsf{set}[a](e) \parallel \mu \longmapsto_{\Sigma,a} \mathsf{set}[a](e') \parallel \mu}$$

D-Set

$$\frac{e\ \mathsf{val}}{\mathsf{set}[a](e) \parallel \mu \otimes \{a \hookrightarrow \_\} \longmapsto_{\Sigma,a} \mathsf{ret}(e) \parallel \mu \otimes \{a \hookrightarrow e\}}$$

D-Ret-1

$$\frac{e \longmapsto e'}{\mathsf{ret}(e) \parallel \mu \longmapsto_{\Sigma} \mathsf{ret}(e') \parallel \mu}$$

D-Bnd-1

$$\frac{e \longmapsto e'}{\mathsf{bnd}(e; x.\, m) \parallel \mu \longmapsto_{\Sigma} \mathsf{bnd}(e'; x.\, m) \parallel \mu}$$

D-Bnd-Cmd

$$\frac{m_1 \parallel \mu \longmapsto_{\Sigma} m_1' \parallel \mu'}{\mathsf{bnd}(\mathsf{cmd}(m_1); x.\, m_2) \parallel \mu \longmapsto_{\Sigma} \mathsf{bnd}(\mathsf{cmd}(m_1'); x.\, m_2) \parallel \mu'}$$

D-Bnd-Ret

$$\frac{e\ \mathsf{val}}{\mathsf{bnd}(\mathsf{cmd}(\mathsf{ret}(e)); x.\, m) \parallel \mu \longmapsto_{\Sigma} m[e/x] \parallel \mu}$$

D-Dcl-1

$$\frac{e \longmapsto e'}{\mathsf{dcl}(e; a.\, m) \parallel \mu \longmapsto_{\Sigma} \mathsf{dcl}(e'; a.\, m) \parallel \mu}$$

D-Dcl-2

$$\frac{e\ \mathsf{val} \qquad m \parallel \mu \otimes \{a \hookrightarrow e\} \longmapsto_{\Sigma,a} m' \parallel \mu' \otimes \{a \hookrightarrow e'\}}{\mathsf{dcl}(e; a.\, m) \parallel \mu \longmapsto_{\Sigma} \mathsf{dcl}(e'; a.\, m') \parallel \mu'}$$

D-Dcl-Ret

$$\frac{e\ \mathsf{val} \qquad e'\ \mathsf{val}}{\mathsf{dcl}(e; a.\, \mathsf{ret}(e')) \parallel \mu \longmapsto_{\Sigma} \mathsf{ret}(e') \parallel \mu}$$