Demand Control-Flow Analysis Technical Report

Kimball Germane[1], Jay McCarthy[2], Michael D. Adams[3], and Matthew Might[4]

[1] Brigham Young University
[2] University of Massachusetts Lowell
[3] University of Utah
[4] University of Alabama

1 Demand 0CFA Correctness

The purpose of this section is to establish that demand 0CFA is sound w.r.t. an exact forward semantics. To do so, we will establish that demand 0CFA is sound w.r.t. an exact *demand* semantics and that this demand semantics has a formal correspondence to an exact forward semantics.

We term the exact demand semantics we define *demand evaluation*. To a first approximation, demand evaluation computes a subderivation of a full derivation of a program's evaluation, where the particular subderivation computed is determined in part by the program subexpression. This is only an approximate description of the action of demand evaluation for a few reasons: first, one may successfully apply demand evaluation to unreachable program subexpressions, computing derivations that don't appear in the derivation of the whole program's evaluation; second, the product of demand evaluation isn't necessarily a single contiguous subderivation but may instead be a set of related subderivations obtained (conceptually) by removing irrelevant judgments from a larger derivation.

Although our intuition is rooted in derivations, we formalize demand evaluation as exact demand analyses related to program configurations, analogous to how we formalized demand 0CFA as approximate demand analyses related to program expressions. Doing so decreases the conceptual distance between demand 0CFA and demand evaluation, making soundness easier to establish. In order to connect demand evaluation to forward evaluation, we reify derivations from exact demand analyses and formally establish a correspondence between those derivations and forward derivations in a technical report [1].

1.1 Demand Evaluation

We define demand evaluation in the same way that we define demand 0CFA: namely, we define an *exact analysis* $(C, \mathcal{E})$ that records exact evaluation facts and two (undecidable) relations, $\models_{eval}$ and $\models_{call}$, over exact analyses and configurations (ρ^d, e, n_c). Both $\models_{eval}$ and $\models_{call}$ relate analyses and configurations in an analogous way to $\models_{fseval}$ and $\models_{fscall}$.

Figure 1 contains formal definitions for the domains of demand evaluation. An exact analysis $(C, \mathcal{E})$ consists of a *cache* C and a caller relation $\mathcal{E}$. In the exact semantics, a caller relation $\mathcal{E}$ is actually a function from called contexts to caller contexts. A cache C is itself a triple $(\$, \sigma, \nu)$ of three functions: $\$$ associates configurations with *names*, σ associates names with values, and ν associates names with calling contexts. A name n serves the function of an address that can be known and transmitted before anything is bound to it; it can be used to obtain the value that may eventually be bound to it in σ. The domain **Name** can be any countably-infinite set; when we must be concrete, it will assume the set of natural numbers $\mathbb{N}$. Demand evaluation environments map variables to names (and not to values contra environments in the exact, big-step semantics presented in Section **??**) which are resolved in the store σ. Similarly, calling contexts are indirected by names which are resolved in the context store ν. (Names within environments and names in contexts come from different address spaces and never interact.) Closures remain the only type of value and remain a pair $(\lambda x.e, \rho^d)$ of a λ-term $\lambda x.e$ and an enclosing environment ρ^d. Configurations are a triple (ρ^d, e, n) of an environment ρ^d closing an expression e in a calling context named by n (and resolved through ν). In the exact semantics, an occurrence is merely a configuration, but we ensure that every configuration treated as such has a value in C.

$$
\begin{array}{rcll}
n & \in & \mathbf{Name} & \text{a countably-infinite set} \\
\rho^d & \in & \mathbf{Env}_d & ::= \quad \bot \mid \rho^d[x \mapsto n] \\
(\lambda x.e, \rho^d) & \in & \mathbf{Value} & = \quad \mathbf{Lam} \times \mathbf{Env}_d \\
c^d & \in & \mathbf{CCtx}_d & ::= \quad \mathrm{mt} \mid \ell :: n \\
(\rho^d, e, n) & \in & \mathbf{Config} & = \quad \mathbf{Env}_d \times \mathbf{Exp} \times \mathbf{Name} \\
 & & \mathbf{Occur} & = \quad \mathbf{Config} \\
\$ & \in & \mathbf{Names} & = \quad \mathbf{Config} \to \mathbf{Name} \\
\sigma & \in & \mathbf{Store} & = \quad \mathbf{Name} \to \mathbf{Value} \\
\nu & \in & \mathbf{CStore} & = \quad \mathbf{Name} \to \mathbf{CCtx}_d \\
C & \in & \mathbf{Cache} & = \quad \mathbf{Names} \times \mathbf{Store} \times \mathbf{Store} \\
\mathcal{E} & \in & \mathbf{Calls} & = \quad \mathbf{Config} \to \mathbf{Config}
\end{array}
$$

Fig. 1. Demand evaluation domains

An exact demand analysis encapsulates the evaluation of a given configuration. However, because configuration environments and calling contexts are threaded through stores, all configurations for a given expression have the same shape. In consequence, a configuration does not uniquely identify a particular evaluation for an expression—that is, the evaluation of a particular instance of the expression in evaluation. Instead, we will define our acceptability relations $\models_{eval}$ and $\models_{call}$ to admit analyses that encapsulate *some* evaluation of the given configuration.

1.2 The $\models_{eval}$ relation

The $\models_{eval}$ relation relates an analysis $(C, \mathcal{E})$ to a judgment $\rho^d, n_c \vdash e \Downarrow^d v^d$ when $(C, \mathcal{E})$ entails the evaluation of the configuration (ρ^d, e, n_c) to the value v^d. Its definition, seen in Figure 2, contains a clause for each type of expression. Each of these clauses functions essentially as its counterpart does in $\models_{fseval}$.

$$
\begin{array}{ll}
 & (C, \mathcal{E}) \models_{eval} \rho^d, n_c \vdash x^\ell \Downarrow^d v^d \\
 & \textit{iff} \\
[\textsc{Ref}] & \begin{array}{l}
C_\$(\rho^d, x^\ell, n_c) = \rho^d(x) \\
(\rho_0^d[x \mapsto n], e, n_{c'}) = \mathrm{bind}(x, \rho^d, x^\ell, n_c) \\
C(\rho_0^d, \lambda x.e, n_{c''}) = (\lambda x.e, \rho_0^d) \\
(C, \mathcal{E}) \models_{call} (\rho_0^d, \lambda x.e, n_{c''}) \Rightarrow_d (\rho_1^d, (e_0\, e_1)^{\ell_0}, n_{c'''}) \implies \\
\quad C_\$(\rho_1^d, e_1, n_{c'''}) = n \\
\quad \mathcal{E}(\rho_0^d[x \mapsto n], e, n_{c'}) = (\rho_1^d, (e_0\, e_1)^{\ell_0}, n_{c'''}) \\
\quad \rho^d(x) = C_\$(\rho_1^d, e_1, n_{c'''}) \\
\quad (C, \mathcal{E}) \models_{eval} \rho_1^d, n_{c'''} \vdash e_1 \Downarrow^d v^d
\end{array} \\
\hline
 & (C, \mathcal{E}) \models_{eval} \rho^d, n_c \vdash (\lambda x.e)^\ell \Downarrow^d (\lambda x.e, \rho^d) \\
[\textsc{Lam}] & \textit{iff} \\
 & C(\rho^d, (\lambda x.e)^\ell, n_c) = (\lambda x.e, \rho^d) \\
\hline
 & (C, \mathcal{E}) \models_{eval} \rho^d, n_c \vdash (e_0\, e_1)^\ell \Downarrow^d v^d \\
 & \textit{iff} \\
 & (C, \mathcal{E}) \models_{eval} \rho^d, n_c \vdash e_0 \Downarrow^d (\lambda x.e, \rho_0^d) \implies \\
[\textsc{App}] & \quad \rho_1^d = \rho_0^d[x \mapsto C_\$(\rho^d, e_1, n_c)] \\
 & \quad C_\nu(n_{c'}) = \ell :: n_c \\
 & \quad \mathcal{E}(\rho_1^d, e, n_{c'}) = (\rho^d, (e_0\, e_1)^\ell, n_c) \\
 & \quad C_\$(\rho_1^d, e, n_{c'}) = C_\$(\rho^d, (e_0\, e_1)^\ell, n_c) \\
 & \quad (C, \mathcal{E}) \models_{eval} \rho_1^d, n_{c'} \vdash e \Downarrow^d v^d
\end{array}
$$

Fig. 2. The $\models_{eval}$ relation

The Ref *clause* The Ref clause specifies that an analysis $(C, \mathcal{E})$ entails the evaluation of a variable reference configuration (ρ^d, x^ℓ, n_c). It ensures that such a configuration evaluates to a value v^d when (1) the name of the configuration reflects the name of the environment binding, (2) the closure that created that binding when applied (furnished by bind) is called at a call site, (3) the name of the argument configuration at that call site is consistent with the new environment binding, and (4) the argument configuration evaluates to v^d.

The bind metafunction (defined in Figure 3) reconstructs not simply the binding λ-term $\lambda x.e$ of x but the configuration at which the particular closure over $\lambda x.e$ first appears.

The Lam *clause* The Lam clause of $\models_{eval}$ specifies that an analysis $(C, \mathcal{E})$ entails the evaluation of a λ-term configuration $(\rho^d, (\lambda x.e)^\ell, n_c)$ if $C(\rho^d, (\lambda x.e)^\ell, n_c) =$

$$
\begin{array}{lll}
\mathrm{bind}(x, \rho^d, e, n) = & (\rho_0^d, (\lambda x.e)^\ell, n_0) & \text{where} \quad \mathbb{K}(e) = (\lambda x.\Box)^\ell \text{ and } \rho^d = \rho_0^d[x \mapsto n] \\
\mathrm{bind}(x, \rho^d, e, n) = \mathrm{bind}(x, \rho_0^d, (\lambda x.y)^\ell, n_0) & & \text{where } \mathbb{K}(e) = (\lambda y.\Box)^\ell,\ x \neq y, \text{ and } \rho^d = \rho_0^d[y \mapsto n] \\
\mathrm{bind}(x, \rho^d, e, n) = \mathrm{bind}(x, \rho^d, (e\, e_1)^\ell, n) & & \text{where} \quad \mathbb{K}(e) = (\Box\, e_1)^\ell \\
\mathrm{bind}(x, \rho^d, e, n) = \mathrm{bind}(x, \rho^d, (e_0\, e)^\ell, n) & & \text{where} \quad \mathbb{K}(e) = (e_0\, \Box)^\ell
\end{array}
$$

Fig. 3. Given a variable x and an expression e in which x appears free, along with its closing environment and calling context, the bind metafunction reconstructs the "birth" context of the closure which yields this binding of x when applied. The resultant name of the calling context must be consistent with (and is uniquely identified by) the calling context discovered for this closure.

$(\lambda x.e, \rho^d)$ meaning that $C_\$(\rho^d, (\lambda x.e)^\ell, n_c) = n$ and $\sigma(n) = (\lambda x.e, \rho^d)$ for some n.

The App *clause* The App clause applies to configurations focused on an application expression $(e_0\, e_1)^\ell$. It ensures that such a configuration evaluates to a value v^d when (1) the operator e_0 is evaluated (within its configuration) to some value $(\lambda x.e, \rho_0^d)$, (2) the environment ρ_0^d and calling context n_c are defined appropriately in the configuration of e, (3) the caller of that configuration is defined in $\mathcal{E}$, and (4) that configuration evaluates to v^d.

1.3 The $\models_{call}$ relation

The $\models_{call}$ relation relates an analysis $(C, \mathcal{E})$ to a judgment $(\rho^d, e, n_c) \Rightarrow_d (\rho_0^d, (e_0\, e_1)^\ell, n_{c'})$ when $(C, \mathcal{E})$ entails that the value of the configuration (ρ^d, e, n_c) is applied at the configuration $(\rho_0^d, (e_0\, e_1)^\ell, n_{c'})$. Its definition, seen in Figure 4, contains a clause for each type of expression context. Each of these clauses functions essentially as its counterpart does in $\models_{fscall}$.

The Rator *clause* The resultant value of a configuration (ρ^d, e_0, n_c) where e_0 has context $(\Box\, e_1)^\ell$ is applied at $(e_0\, e_1)^\ell$ (assuming the convergence of evaluation of e_1) so its caller configuration is $(\rho^d, (e_0\, e_1)^\ell, n_c)$.

The Rand *clause* The resultant value of a configuration (ρ^d, e_1, n_c) where e_1 has context $(e_0\, \Box)^\ell$ is bound to the parameter x of the value $(\lambda x.e, \rho_0^d)$ of e_0 and appears at every reference to x in e. The Rand clause ensures that the argument value is called at configuration $(\rho_0^d, (e_2\, e_3)^{\ell_0}, n_{c'})$ when (1) the operator expression is evaluated to a value, (2) the environment of that value is extended with the name of the argument and the calling context is extended with the call-site label, and (3) the find metarelation furnishes a configuration whose value is called at $(\rho_0^d, (e_2\, e_3)^{\ell_0}, n_{c'})$.

The find metarelation, defined in Figure 5, relates references to x in a configuration (ρ^d, e, n_c) to configurations $(\rho_0^d, x^\ell, n_{c'})$ wich constitute references to x.

[Rator]	$(C,\mathcal{E}) \models_{call} (\rho^d, e_0, n_c) \Rightarrow_d (\rho^d, (e_0\, e_1)^\ell, n_c)$ for $\mathbb{K}(e_0) = (\square\, e_1)^\ell$ *iff* always
[Rand]	$(C,\mathcal{E}) \models_{call} (\rho^d, e_1, n_c) \Rightarrow_d (\rho_0^d, (e_2\, e_3)^{\ell_0}, n_{c'})$ for $\mathbb{K}(e_1) = (e_0\, \square)^\ell$ *iff* $(C,\mathcal{E}) \models_{eval} \rho^d, n_c \vdash e_0 \Downarrow^d (\lambda x.e, \rho_1^d) \implies$ $\rho_2^d = \rho_1^d[x \mapsto C_\$(\rho^d, e_1, n_c)]$ $C_\nu(n_{c''}) = \ell :: n_c$ $(\rho_3^d, x^{\ell_1}, n_{c'''}) \in \text{find}(x, \rho_2^d, e, n_{c''}) \implies$ $(C,\mathcal{E}) \models_{call} (\rho_3^d, x^{\ell_1}, n_{c'''}) \Rightarrow_d (\rho_0^d, (e_2\, e_3)^{\ell_0}, n_{c'})$
[Body]	$(C,\mathcal{E}) \models_{call} (\rho^d[x \mapsto n], e, n_c) \Rightarrow_d (\rho_0^d, (e_0\, e_1)^{\ell_0}, n_{c'})$ for $\mathbb{K}(e) = (\lambda x.\square)^\ell$ *iff* $(C,\mathcal{E}) \models_{call} (\rho^d, (\lambda x.e)^\ell, n_{c''}) \Rightarrow_d (\rho_1^d, (e_2\, e_3)^{\ell_1}, n_{c'''}) \implies$ $(C,\mathcal{E}) \models_{call} (\rho_1^d, (e_2\, e_3)^{\ell_1}, n_{c'''}) \Rightarrow_d (\rho_0^d, (e_0\, e_1)^{\ell_0}, n_{c'})$
[Top]	$(C,\mathcal{E}) \models_{call} (\rho^d, e, n_c) \Rightarrow_d (\rho_0^d, (e_0\, e_1)^\ell, n_{c'})$ for $\mathbb{K}(e) = \square$ *iff* never

Fig. 4. The $\models_{call}$ relation

$(\rho^d, x^\ell, n_c) \in \text{find}(x, \rho^d, x^\ell, n_c)$	*iff*	always
$(\rho_0^d, x^\ell, n_{c'}) \in \text{find}(x, \rho^d, \lambda y.e^\ell, n_c)$	*iff*	$(\rho_0^d, x^\ell, n_{c'}) \in \text{find}(x, \rho^d[y \mapsto n], e, n_{c''})$ where $x \neq y$ and for some n, $n_{c''}$
$(\rho_0^d, x^\ell, n_c) \in \text{find}(x, \rho^d, (e_0\, e_1)^\ell, n_c)$	*iff*	$(\rho_0^d, x^\ell, n_c) \in \text{find}(x, \rho^d, e_0, n_c)$
$(\rho_0^d, x^\ell, n_c) \in \text{find}(x, \rho^d, (e_0\, e_1)^\ell, n_c)$	*iff*	$(\rho_0^d, x^\ell, n_c) \in \text{find}(x, \rho^d, e_1, n_c)$

Fig. 5. The find relation

The BODY *clause* If a configuration is in a body context, its result becomes the result of the caller of the closure over its enclosing λ-term. The BODY clause ensures that the resultant value of a configuration $(\rho^d[x \mapsto n], e, n_c)$ such that e has syntactic context $(\lambda x.\Box)^\ell$ is called at a configuration $(\rho_1^d, (e_2\, e_3)^{\ell_1}, n_{c'''})$ when (1) the enclosing value is called at configuration $(\rho_1^d, (e_2\, e_3)^{\ell_1}, n_{c'''})$, and (2) the resultant value of $(\rho_1^d, (e_2\, e_3)^{\ell_1}, n_{c'''})$—the value of the initial configuration over e—is called at configuration $(\rho_1^d, (e_2\, e_3)^{\ell_1}, n_{c'''})$.

The TOP *clause* A closure that reaches the top level of the program is not called at any configuration within evaluation.

1.4 Soundness

We can now formally state the correctness of demand 0CFA relative to demand evaluation. Correctness is expressed by two lemmas which each relate a demand evaluation relation to its demand 0CFA counterpart.

Lemma 1. *If* $(C, \mathcal{E}) \models_{eval} \rho^d, n_c \vdash t^\ell \Downarrow^d (\lambda x.e, \rho_0^d)$ *then, if* $(\hat{\mathcal{C}}, \hat{\mathcal{E}}) \models_{fseval} t^\ell$, $\lambda x.e \in \hat{\mathcal{C}}(\ell)$.

Lemma 2. *If* $(C, \mathcal{E}) \models_{call} (\rho^d, t^\ell, n_c) \Rightarrow_d (\rho_0^d, (e_0\, e_1)^{\ell_0}, n_{c'})$ *where* $C(\rho^d, t^\ell, n_c) = (\lambda x.e, \rho_0^d)$ *then, if* $(\hat{\mathcal{C}}, \hat{\mathcal{E}}) \models_{fscall} (\lambda x.e, t^\ell)$, $(e_0\, e_1)^{\ell_0} \in \hat{\mathcal{E}}(e)$.

Lemma 1 states that a demand 0CFA analysis $(\hat{\mathcal{C}}, \hat{\mathcal{E}})$ acceptable by $\models_{fseval}$ for an expression e contains an abstraction of every value for which there is an acceptable (by $\models_{eval}$) exact analysis $(C, \mathcal{E})$. Lemma 2 says that the demand 0CFA specification $\models_{fseval}$ will always include abstractions of calling configurations discovered by the demand evaluation specification $\models_{call}$. Because we took great pains to keep exact demand evaluation close to approximate demand evaluation, the proofs of these lemmas proceed straightforwardly by mutual induction on the definitions of $\models_{eval}$ and $\models_{call}$. The coinductive step proceeds by cases over expressions, in the case of Lemma 1, and syntactic contexts, in the case of Lemma 2. The corresponding clauses in the exact and approximate relations themselves tightly correspond, so each case proceeds without impediment.

2 Demand Evaluation Correctness

Before we formally establish the correctness of demand evaluation, let's informally discuss some of the subtleties involved. Essentially, we cannot directly compare a demand evaluation to the forward evaluation of a whole program because we can demand-evaluate unreachable expressions. In fact, an expression may be reachable but its evaluation might diverge in forward evaluation but converge in demand evaluation.

As an example, consider the evaluation of the call $(e\, \Omega)$ in an environment ρ_0^d such that the value of e would ignore its argument (if applied) but Ω diverges.

The forward call-by-value evaluation of the call diverges but the demand evaluation of it doesn't. Now consider that evaluation in an environment ρ_1^d such that the value of e wouldn't ignore its argument and Ω still diverges. There is no analysis $(C, \mathcal{E})$ and value v^d such that $(C, \mathcal{E}) \models_{eval} \rho_1^d, n_c \vdash (e\,\Omega) \Downarrow^d v^d$ holds.

This leaves us with two conclusions: first, the "footprint" of every demand analysis is finite because it describes a finite evaluation; second, within this finite evaluation may be unevaluated configurations with values that, while not used, still appear in evaluation (e.g. in environments).

To cope with unevaluated configurations, we will introduce an indeterminate value $\square$ to our semantics. Indeterminate values v^i can take the form of closures or be indeterminate; indeterminate environments ρ^i map to indeterminate values.

$$v^i ::= (\lambda x.e, \rho^i) \mid \square \qquad\qquad \rho^i ::= \bot \mid \rho^i[x \mapsto v^i]$$

We also introduce indeterminate calling contexts c^i of the form

$$c^i ::= \mathrm{mt} \mid \ell :: c^i \mid \square_c$$

where a calling context with an indeterminate tail reflects the fact that we consider evaluations with unknown provenance.

Now we can introduce semantic judgments $\rho^i, c^i \vdash e \Downarrow^i v^i$. The Ref and Lam rules for this semantics are identical to the "fully-ground" semantics. We will not need to deal with cases in which the operator value is indeterminate, so we will not need to devise an App rule. Instead, we will have the rule

$$\textsc{Ind}\frac{}{\rho^i, c^i \vdash e \Downarrow^i \square}$$

that can apply to any expression e when particular external conditions are met (which we discuss below).

With the indeterminate value, we can define a partial order over values

$$\square \sqsubseteq (\lambda x.e, \rho^d) \qquad\qquad (\lambda x.e, \rho_0^d) \sqsubseteq (\lambda x.e, \rho_1^d) \text{ iff } \rho_0^d \sqsubseteq \rho_1^d$$

and environments

$$\bot \sqsubseteq \bot \qquad\qquad \rho_0^d[x \mapsto v_0^i] \sqsubseteq \rho_1^d[x \mapsto v_1^i] \text{ iff } \rho_0^d \sqsubseteq \rho_1^d \text{ and } v_0^i \sqsubseteq v_1^i$$

For evaluations of configurations in which no operator value is indeterminate, it is straightforward to prove that evaluation reflects an ordering on configurations to results. That is, that, if $(\rho_0^i, e, c_0^i) \sqsubseteq (\rho_1^i, e, c_1^i)$, $\rho_0^i, c_0^i \vdash e \Downarrow^i v_0^i$, and $\rho_1^i, c_1^i \vdash e \Downarrow^i v_1^i$, then $v_0^i \sqsubseteq v_1^i$. Since fully-ground configurations are maximal elements, their evaluation results are also.

We can connect demand evaluation to the indeteriminate semantics by first *reconstructing* indeterminate configurations (ρ^i, e, c^i) from demand configurations (ρ^d, e, c^d). Reconstruction resolves values through the value store σ and calling context store ν via the metafunctions $recons_v$ to reconstruct values, $recons_\rho$ to reconstruct environments, and $recons_c$ to reconstruct calling contexts, each defined as

$$recons_v(\sigma, n) = \begin{cases} (\lambda x.e, recons_\rho(\sigma, \rho^d)) & \text{if } \sigma(n) = (\lambda x.e, \rho^d) \\ \square & \text{if } \sigma(n) \text{ is not defined} \end{cases}$$

$$recons_\rho(\sigma, \rho^d) = \begin{cases} \bot & \text{if } \rho^d = \bot \\ recons_\rho(\sigma, \rho_0^d)[x \mapsto recons_v(\sigma, n)] & \text{if } \rho^d = \rho_0^d[x \mapsto n] \end{cases}$$

$$recons_c(\nu, n_c) = \begin{cases} \text{mt} & \text{if } \nu(n_c) = \text{mt} \\ \ell :: recons_c(\nu, n_{c'}) & \text{if } \nu(n_c) = \ell :: n_{c'} \\ \square_c & \text{if } \nu(n_c) \text{ is not defined} \end{cases}$$

To establish that demand evaluation respects the indeterminate semantics, we establish that (1) $\models_{eval}$ respects the intermediate semantics in the sense that a reconstructed configuration yields the expected reconstructed value, and (2) $\models_{call}$ respects the intermediate semantics in the sense that the value of a reconstructed configuration is applied at the expected reconstructed calling context. To achieve this, we reconstruct or reify the evaluation encapsulated in an exact analysis $(C, \mathcal{E})$. Given an exact analysis $(C, \mathcal{E})$, a demand evaluation configuration (ρ^d, e, n_c), and its reconstruction (ρ^i, e, c^i), we denote the reification of the derivation by $(C, \mathcal{E}) \models \rho^i, c^i \vdash e \Downarrow^i v^i$. For instance we reify a Ref rule by

$$\textsc{Ref}\frac{}{\rho^i, c^i \vdash x^\ell \Downarrow v^i}$$

for $e = x^\ell$ and $v^i = recons_v(\sigma, n)$ if $(\hat{\mathcal{C}}, \hat{\mathcal{E}}) \models_{eval} \rho^d, n_c \vdash x^\ell \Downarrow^d v^d$ and $C_\$(\rho^d, x^\ell, n_c) = n$ and reify a Lam rule similarly. We reify an App rule by recursively reifying its premises. If a configuration (ρ^d, e, n_c) is encountered such that neither $(\hat{\mathcal{C}}, \hat{\mathcal{E}}) \models_{eval} \rho^d, n_c \vdash e \Downarrow^d v^d$ holds for any v^d nor $(\hat{\mathcal{C}}, \hat{\mathcal{E}}) \models_{call} (\rho^d, e, n_c) \Rightarrow_d (\rho_0^d, (e_0\, e_1)^{\ell_0}, n_{c'})$ holds for any $(\rho_0^d, (e_0\, e_1)^{\ell_0}, n_{c'})$, then we reify an Ind rule. From here, it is straightforward to demonstrate (1) and (2) by the following lemmas:

Lemma 3. *If* $(C, \mathcal{E}) \models_{eval} \rho^d, n_c \vdash e \Downarrow^d v^d$ *and* $(C, \mathcal{E}) \models recons_\rho(\sigma, \rho^d), recons_c(\nu, n_c) \vdash e \Downarrow^i v^i$*, then* $recons_v(\sigma, C_\$(\rho^d, e, n_c)) = v^i$*.*

Lemma 4. *If* $(C, \mathcal{E}) \models_{call} (\rho^d, e, n_c) \Rightarrow_d (\rho_0^d, (e_0\, e_1)^{\ell_0}, n_{c'})$ *where* $\sigma(\$(\rho^d, e, n_c)) = (\lambda x.e', \rho_1^d)$*, then* $(C, \mathcal{E}) \models recons_\rho(\sigma, \rho_0^d), recons_c(\nu, n_{c'}) \vdash e_0 \Downarrow^i v^i$ *where* $v^i = (\lambda x.e', recons_\rho(\sigma, \rho_1^d))$*.*

The proofs of these lemmas proceed by mutual induction and induction over the definitions of $\models_{eval}$ and $\models_{call}$.

References

1. Germane, K., Might, M.: Demand control-flow analysis. Tech. rep. (January 2019), http://kimball.germane.net/germane-dcfa-techreport.pdf