Domain Theoretic Techniques in an Operational Setting

Ian A. Mason
Stanford University
iam@cs.stanford.edu

Scott Smith
The Johns Hopkins University
scott@cs.jhu.edu

Carolyn Talcott
Stanford University
clt@sail.stanford.edu

1. Introduction

With a domain model comes notions of partial order, directed sets, least upper bound, continuity, finiteness and ω-algebraicity. These properties are critical to proving important properties of programming languages. However, denotational models often fail to adequately capture program equivalence. For example, the problem of constructing a fully abstract denotational semantics for languages with imperative features remains open [9, 10]. In an operational setting the notions of operational approximation and equivalence by definition adequately capture program equivalence. They also satisfy many of the properties of the domain partial ordering. We would like to develop a semantic framework that provides the best of both worlds. In [13] we defined operational analogues of the domain theoretic notions of directed set, least upper bound, and continuity. The latter is accomplished by defining a notion of equivalence between directed sets.

In this paper we study a simple functional language extending the call-by-value lambda calculus with numbers, pairs and sums. We define the syntax, give an operational semantics based on a single step reduction relation, and sketch the development of the theory of operational equivalence and directed set equivalence based on this semantics. In the denotational setting, finite elements play an important role in establishing properties of their infinite counterparts. To construct a purely operational analogue to the finite elements of domain theory we define the syntax and semantics of labelled (indexed) expressions. This labelled computation system generalizes to the above language the notions of labelled terms and reduction as developed by Hyland, Levy, and Wadsworth [14, 15, 4, 5, 6] for the pure lambda calculus. We establish key properties of the labelled computation system and the induced notion of operational equivalence, including an approximation property that relates equivalence of expressions to equivalence of labelled expressions. This property is an operational analogue of ω-algebraicity. We illustrate the use of labelled computations with two applications. In the first application we use the approximation property to establish the least fixed-point theorem. The use of labelled expressions greatly simplifies the previous operational proofs. Secondly we construct a faithful ideal model of types following [7, 1]. Finite domain elements are a key component of these ideal constructions. One advantage of this operational approach is that it extends smoothly to functional languages with imperative features such as ML and Scheme, while not suffering from a severe loss of abstraction.

2. A Simple Functional Language

We define the syntax and semantics for a simple call-by-value functional language with pairs, sums, and boolean and natural numbers.

2.1. Syntax

To present the syntax of our language we assume given a set of variables $\mathbb{X}$. The set of booleans, $\mathbb{B}$, the set of natural numbers, $\mathbb{N}$, the set of λ-abstractions, $\mathbb{L}$, the set of value expressions, $\mathbb{V}$, the set of immutable pairs, $\mathbb{P}$, the set of left summands $\mathbb{I}_{\mathrm{L}}$, the set of right summands $\mathbb{I}_{\mathrm{R}}$, and the set of expressions, $\mathbb{E}$, are defined, mutually recursively, as the least sets satisfying the following equations.

Definition ($\mathbb{L}$ $\mathbb{B}$ $\mathbb{V}$ $\mathbb{N}$ $\mathbb{P}$ $\mathbb{I}_{\mathrm{L}}$ $\mathbb{I}_{\mathrm{R}}$ $\mathbb{E}$):

$$\mathbb{B} = \{\mathtt{t}, \mathtt{f}\} \quad \mathbb{N} = \{0\} + \mathtt{succ}(\mathbb{N}) \quad \mathbb{P} = \mathtt{pr}(\mathbb{V}, \mathbb{V}) \quad \mathbb{I}_{\mathrm{L}} = \mathtt{inl}(\mathbb{V}) \quad \mathbb{I}_{\mathrm{R}} = \mathtt{inr}(\mathbb{V})$$

$$\mathbb{L} = \lambda\mathbb{X}.\mathbb{E} \quad \mathbb{E} = \mathbb{V} + \mathtt{app}(\mathbb{E}, \mathbb{E}) + \mathbb{F}_n(\mathbb{E}^n) \quad \mathbb{V} = \mathbb{X} + \mathbb{B} + \mathbb{N} + \mathbb{L} + \mathbb{P} + \mathbb{I}_{\mathrm{L}} + \mathbb{I}_{\mathrm{R}}$$

The unary, binary and ternary operations are:

$$\{\mathtt{isnat}, \mathtt{isz}, \mathtt{succ}, \mathtt{pred}, \mathtt{ispr}, \mathtt{fst}, \mathtt{snd}, \mathtt{isl}, \mathtt{isr}, \mathtt{inl}, \mathtt{inr}, \mathtt{outl}, \mathtt{outr}, \} = \mathbb{F}_1$$

$$\{\mathtt{pr}\} = \mathbb{F}_2 \quad \{\mathtt{br}\} = \mathbb{F}_3$$

Note that we are treating `succ`, `pr`, `inl` and `inr` as *constructors*, with *recognizers* `isnat`, `ispr`, `isl`, `isr`, and *selectors* `pred`, `fst`, `snd` (both for pairs), `outl`, `outr`. `isz` is a test for zero, while `br` is a branching primitive.

We let n range over $\mathbb{N}$, x, y, z range over $\mathbb{X}$, v range over $\mathbb{V}$, ρ range over $\mathbb{L}$, and e range over $\mathbb{E}$. λ is a binding operator and free and bound variables of expressions are defined as usual. Two expressions are considered equal if they are the same up to renaming of bound variables. $\mathrm{FV}(e)$ is the set of free variables of e. $e^{\{x:=e'\}}$ is the result of substituting e' for x in e taking care not to trap free variables of e'. For any syntactic domain Y and set of variables X we let Y_X be the elements of Y with free variables in X. A *closed expression* is an expression with no free variables. Thus $\mathbb{E}_\emptyset$ is the set of all closed expressions. A value substitution, σ, is a finite map from variables to values (i.e. $\sigma \in \mathbb{X} \xrightarrow{\omega} \mathbb{V}$. We write $\{x_i := v_i \mid i < n\}$ for the substitution σ with domain $\{x_i \mid i < n\}$ such that $\sigma(x_i) = v_i$ for $i < n$. e^σ is the result of simultaneous substitution of free occurrences of $x \in \mathrm{Dom}(\sigma)$ in e by $\sigma(x)$, again taking care not to trap variables.

Definition ($\mathbb{C}$): Contexts are expressions with holes. We use $\bullet$ to denote a hole. The set of contexts, $\mathbb{C}$, is defined by

$$\mathbb{C} = \{\bullet\} + \mathbb{X} + \mathbb{B} + \mathbb{N} + \lambda\mathbb{X}.\mathbb{C} + \mathtt{app}(\mathbb{C}, \mathbb{C}) + \mathbb{F}_n(\mathbb{C}^n)$$

We let C range over $\mathbb{C}$. $C[e]$ denotes the result of replacing any holes in C by e. Free variables of e may become bound in this process, we let $\mathrm{Traps}(C)$ be those variables which may be trapped.

2.2. Semantics

The operational semantics of expressions is given by a reduction relation $\mapsto$ on expressions. A primitive step is either a β-reduction or the application of a primitive operation to a sequence of value expressions.

Definition ($\mathbb{E}_{\rm r}$): The set of redexes, $\mathbb{E}_{\rm r}$, is defined as

$$\mathbb{E}_{\rm r} = \mathtt{app}(\mathbb{V},\mathbb{V}) + (\mathbb{F}_n(\mathbb{V}^n) - (\mathbb{P} + \mathbb{I}_{\rm L} + \mathbb{I}_{\rm R} + \mathbb{N}))$$

Reduction contexts identify the subexpression of an expression that is to be evaluated next, they correspond to the left-first, call-by-value reduction strategy of Plotkin [11] and were first introduced in [3].

Definition ($\mathbb{R}$): The set of reduction contexts, $\mathbb{R}$, is the subset of $\mathbb{C}$ defined by

$$\mathbb{R} = \{\bullet\} + \mathtt{app}(\mathbb{R},\mathbb{E}) + \mathtt{app}(\mathbb{V},\mathbb{R}) + \mathbb{F}_{m+n+1}(\mathbb{V}^m,\mathbb{R},\mathbb{E}^n)$$

We let R range over $\mathbb{R}$. An expression is either a value expression or decomposes uniquely into a redex placed in a reduction context.

Lemma (Decomposition): If $e \in \mathbb{E}$ then either $e \in \mathbb{V}$ or e can be written uniquely as $R[e']$ where R is a reduction context and $e' \in \mathbb{E}_{\rm r}$.

Definition ($\mapsto_1$):

(beta)	$\mathtt{app}(\lambda x.e, v) \mapsto_1 e^{\{x:=v\}}$	(br)	$\mathtt{br}(v_1, v_2, v) \mapsto_1 \begin{cases} v_1 & \text{if } v \neq \mathtt{f} \\ v_2 & \text{if } v = \mathtt{f} \end{cases}$
(isnat)	$\mathtt{isnat}(v) \mapsto_1 \begin{cases} \mathtt{t} & \text{if } v \in \mathbb{N} \\ \mathtt{f} & \text{if } v \notin \mathbb{N} \end{cases}$	(ispr)	$\mathtt{ispr}(v) \mapsto_1 \begin{cases} \mathtt{t} & \text{if } v \in \mathbb{P} \\ \mathtt{f} & \text{if } v \notin \mathbb{P} \end{cases}$
(iszero)	$\mathtt{isz}(v) \mapsto_1 \begin{cases} \mathtt{t} & \text{if } v = 0 \\ \mathtt{f} & \text{if } v \neq 0 \end{cases}$	(isl)	$\mathtt{isl}(v) \mapsto_1 \begin{cases} \mathtt{t} & \text{if } v \in \mathbb{I}_{\rm L} \\ \mathtt{f} & \text{if } v \notin \mathbb{I}_{\rm L} \end{cases}$
(isr)	$\mathtt{isr}(v) \mapsto_1 \begin{cases} \mathtt{t} & \text{if } v \in \mathbb{I}_{\rm R} \\ \mathtt{f} & \text{if } v \notin \mathbb{I}_{\rm R} \end{cases}$	(pred)	$\mathtt{pred}(\mathtt{succ}(v)) \mapsto_1 v$
(fst)	$\mathtt{fst}(\mathtt{pr}(v_0, v_1)) \mapsto_1 v_0$	(snd)	$\mathtt{snd}(\mathtt{pr}(v_0, v_1)) \mapsto_1 v_1$
(outl)	$\mathtt{outl}(\mathtt{inl}(v)) \mapsto_1 v$	(outr)	$\mathtt{outl}(\mathtt{inr}(v)) \mapsto_1 v$

Definition ($\mapsto$): The reduction relation $\mapsto$ is the reflexive transitive closure of the relation generated by the following clause:

(red) $\quad e \mapsto_1 e' \;\Rightarrow\; R[e] \mapsto R[e']$

In order to enforce a typing discipline later in the paper, we isolate the run-time type errors.

Definition ($\downarrow \wr \ll$): Let e be a closed expression. Then: e is *defined* (written $\downarrow e$) if it evaluates to a value; e is in a *stuck state* iff it is not a value and it does not reduce; e is *stuck* (written $\wr e$) if it reduces to a stuck state. Furthermore a closed expression e_0 is *less defined* than another closed expression e_1 (written $e_0 \ll e_1$) iff whenever e_0 is defined (or stuck), e_1 is also defined (resp. stuck).

We define operational equivalence in such a way as to enforce the distinction between non-termination and *type* errors. This notion of operational equivalence is a slight variation of the traditional notion, however as we shall see, it satisfies the same key properties.

Definition ($\sqsubseteq\!\!\!\sim$):

$$e_0 \lesssim e_1 \;\Leftrightarrow\; (\forall C \in \mathbb{C} \mid C[e_0], C[e_1] \in \mathbb{E}_\emptyset)(C[e_0] \ll C[e_1])$$
$$e_0 \cong e_1 \;\Leftrightarrow\; e_0 \lesssim e_1 \wedge e_1 \lesssim e_0$$

Operational approximation enjoys most of the properties of operational equivalence. In particular it is a congruence and and satisfies the context lemma (called the (**ciu**) theorem in [8]).

Lemma ($\lesssim$-compatible): If $e_0 \lesssim e_1$ then $C[e_0] \lesssim C[e_1]$.

Theorem (ciu): $e_0 \lesssim e_1 \;\Leftrightarrow\; (\forall R \in \mathbb{R}, \sigma \mid R[e_0^\sigma], R[e_1^\sigma] \in \mathbb{E}_\emptyset)(R[e_0^\sigma] \ll R[e_1^\sigma])$

We now review the development of [13], defining $\lesssim$-directed sets of expressions, and notions of ordering and equivalence on those sets. First, some notation. We let E range over subsets of $\mathbb{E}$. Constructors implicitly extend pointwise to sets of expressions, so for instance $\mathsf{app}(e, E)$ abbreviates $\{\mathsf{app}(e, e') \mid e' \in E\}$. Substitutions σ extend to sets of expressions: $E^\sigma = \{e^\sigma \mid e \in E\}$. Recall that σ is a finite map, so some E have no closing substitution. $C[E] = \{C[e] \mid e \in E\}$.

Definition (directed): $E \subseteq \mathbb{E}$ is *directed* iff for all $e_0 \in E, e_1 \in E$, there exists $e_2 \in E$ such that $e_0 \lesssim e_2$ and $e_1 \lesssim e_2$.

Definition ($\lesssim_S$ $\cong_S$):

$$E_0 \lesssim_S E_1 \;\Leftrightarrow\; (\forall e_0 \in E_0)(\forall C \in \mathbb{C} \mid C[E_0], C[E_1] \in \mathcal{P}(\mathbb{E}_\emptyset))(\exists e_1 \in E_1)(C[e_0] \ll C[e_1])$$
$$E_0 \cong_S E_1 \;\Leftrightarrow\; E_0 \lesssim_S E_1 \wedge E_1 \lesssim_S E_0$$

Properties of $\lesssim_S$ include the following, see [13] for motivations and proofs.

Lemma ($\lesssim_S$ properties):

(i) $e_0 \lesssim e_1$ iff $\{e_0\} \lesssim_S \{e_1\}$

(ii) $E \lesssim_S \{e'\}$ iff $e \lesssim e'$ for all $e \in E$.

(iii) $e \in E$ implies $\{e\} \lesssim_S E$.

(iv) $(\forall e_0 \in E_0)(\exists e_1 \in E_1)(e_0 \lesssim e_1)$. implies $E_0 \lesssim_S E_1$, but the converse *fails.*

(v) $\lesssim_S$ is compatible, namely if $E_0 \lesssim_S E_1$, then $C[E_0] \lesssim_S C[E_1]$.

(vi) $\cong_S$ is a congruence.

3. Labelled Expressions

In this section we introduce and compute with labelled expressions $\widehat{\mathbb{E}}$. Define $\mathbb{E}^+$ to be $\mathbb{E}$ with $\mathbb{F}_1$ extended to include a new (unary) operation `cut`. A labelled or indexed expression is a pair $\texttt{<}e, \mathrm{L}\texttt{>}$ where $e \in \mathbb{E}^+$ and L is a map from all occurrences of all non-atomic subexpressions of e to *positive* natural numbers. An atomic expression is either a variable or a constant ($\{\mathtt{t}, \mathtt{f}, 0\}$). The set of all labelled expressions is denoted by $\widehat{\mathbb{E}}$, we let $\hat{e}$ range over $\widehat{\mathbb{E}}$. Define $\widehat{\mathbb{V}}$ to be $\texttt{<}v, \mathrm{L}\texttt{>}$ for value v. $\hat{v}$ ranges over $\widehat{\mathbb{V}}$. Note that this definition differs from the traditional indexed λ-terms [2, 14, 15] in that variables and constants are not labelled, and no term is labelled by zero. This is because computation suspends before any term is given a zero label in call-by-value reduction.

Labelled expressions were introduced in [14, 15, 4, 5] to prove the *approximation theorem*. The labels on expressions constitute a form of resource. Evaluation depletes this resource. To define the notion of approximant, a new constant Ω is introduced to indicate fully-depleted resources. Continuing in the call-by-value spirit, we introduce **susp**, to play the analogous role. **susp** is a meta-theoretic constant, not an expression.

In keeping with the tradition, the number or index associated to each occurrence of a subterm is often denoted as a superscript to that occurrence. We also use the convention that $\mathrm{L}(e)$ is the label of the outer-most constructor of e in $\texttt{<}e, \mathrm{L}\texttt{>}$. Substitution is extended to labelled expressions in the obvious fashion. In particular, since variables are not labelled, $x^{\{x:=\hat{e}\}} = \hat{e}$. Given two labellings of e, L_0 and L_1, we write $\mathrm{L}_0 \leq \mathrm{L}_1$ to mean that every occurrence of every expression has a smaller than or equal label under L_0 than under L_1.

The set of finite value expressions $\mathbb{V}^{(\omega)}$ is a subset of the labelled value expressions $\hat{v}$ that corresponds to the finite elements of the value domain. Not all labelled value expressions can be considered to be finite values. The resources implicit in the labelling may not be sufficient to construct them. For example $\mathtt{succ}^2(0)$ is a finite value but $\mathtt{succ}^1(0)$ and $\mathtt{succ}^2(\mathtt{succ}^2(0))$ are not.

To make this notion precise we define a family of functions $\Pi^j \in \widehat{\mathbb{V}} \to (\widehat{\mathbb{V}} \cup \{\mathbf{susp}\})$ for $j \in \mathbb{N}$ by the following clauses.

Definition ($\Pi^j(\hat{v})$):

(base) $\quad \Pi^0(\hat{v}) = \mathbf{susp}$

(atom) $\quad \Pi^{j+1}(a) = a \qquad \text{if } a \in \{0, \mathtt{t}, \mathtt{f}\}$

(lam) $\quad \Pi^{j+1}(\lambda^{k+1}x.\hat{e}) = \lambda^{\min(j,k)+1}x.\hat{e}$

(cons.i) $\quad \Pi^{j+1}(\gamma^{k+1}(\hat{v}_1, \ldots, \hat{v}_n)) = \gamma^{\min(j,k)+1}(\Pi^j(\hat{v}_1), \ldots, \Pi^j(\hat{v}_n))$

$\qquad$ if $\Pi^j(\hat{v}_i) \neq \mathbf{susp}$ for all $1 \leq i \leq n$.

(cons.ii) $\quad \Pi^{j+1}(\gamma^{k+1}(\hat{v}_1, \ldots, \hat{v}_n)) = \mathbf{susp}$

$\qquad$ if $\Pi^j(\hat{v}_i) = \mathbf{susp}$ for some $1 \leq i \leq n$.

where $\gamma \in \{\mathtt{succ}, \mathtt{pr}, \mathtt{inl}, \mathtt{inr}\}$.

Definition ($\mathbb{V}^{(\omega)}$): The set of finite value expressions is defined to be:

$$\mathbb{V}^{(\omega)} = \{\texttt{<}v, \mathrm{L}\texttt{>} \in \widehat{\mathbb{V}} \mid \Pi^{\mathrm{L}(v)}(\texttt{<}v, \mathrm{L}\texttt{>}) \neq \mathbf{susp}\}$$

We let ν range over $\mathbb{V}^{(\omega)}$. Note that a labelled lambda expression is always a finite value. It is only in the case of the structured data where resources may prove insufficient for construction. Let $\mathbb{N}^{(\omega)}$ be those elements of $\mathbb{V}^{(\omega)}$ with outermost constructor `succ` together with 0. $\mathbb{L}^{(\omega)}$, $\mathbb{P}^{(\omega)}$, $\mathbb{I}_{\mathrm{L}}^{(\omega)}$, and $\mathbb{I}_{\mathrm{R}}^{(\omega)}$ are defined similarly.

Single-step reduction $\overset{1}{\mapsto}_1 \subseteq \widehat{\mathbb{E}} \times (\widehat{\mathbb{E}} \cup \{\mathbf{susp}\})$ is now defined.

Definition ($\overset{1}{\mapsto}_1$):

$$\text{(beta)}\quad \mathtt{app}^{j+1}(\lambda^{k+1}x.\hat{e},\nu) \overset{1}{\mapsto}_1 \begin{cases} \mathtt{cut}^{\min(j,k)}(\hat{e}^{\{x:=\nu'\}}) & \text{if } \Pi^{\min(j,k)}(\nu)=\nu' \\ \mathbf{susp} & \text{if } \Pi^{\min(j,k)}(\nu)=\mathbf{susp} \end{cases}$$

$$\text{(cut)}\quad \mathtt{cut}^{j}(\hat{v}) \mapsto_1 \Pi^{j}(\hat{v}) \qquad\qquad \text{(ispr)}\quad \mathtt{ispr}^{j+1}(\nu) \overset{1}{\mapsto}_1 \begin{cases} \Pi^{j}(\mathtt{t}) & \text{if } \nu \in \mathbb{P}^{(\omega)} \\ \Pi^{j}(\mathtt{f}) & \text{if } \nu \notin \mathbb{P}^{(\omega)} \end{cases}$$

$$\text{(isnat)}\quad \mathtt{isnat}^{j+1}(\nu) \overset{1}{\mapsto}_1 \begin{cases} \Pi^{j}(\mathtt{t}) & \text{if } \nu \in \mathbb{N}^{(\omega)} \\ \Pi^{j}(\mathtt{f}) & \text{if } \nu \notin \mathbb{N}^{(\omega)} \end{cases} \qquad \text{(iszero)}\quad \mathtt{isz}^{j+1}(\nu) \overset{1}{\mapsto}_1 \begin{cases} \Pi^{j}(\mathtt{t}) & \text{if } \nu = 0 \\ \Pi^{j}(\mathtt{f}) & \text{if } \nu \neq 0 \end{cases}$$

$$\text{(isl)}\quad \mathtt{isl}^{j+1}(\nu) \overset{1}{\mapsto}_1 \begin{cases} \Pi^{j}(\mathtt{t}) & \text{if } \nu \in \mathbb{I}_{\mathbf{L}}^{(\omega)} \\ \Pi^{j}(\mathtt{f}) & \text{if } \nu \notin \mathbb{I}_{\mathbf{L}}^{(\omega)} \end{cases} \qquad \text{(isr)}\quad \mathtt{isr}^{j+1}(\nu) \overset{1}{\mapsto}_1 \begin{cases} \Pi^{j}(\mathtt{t}) & \text{if } \nu \in \mathbb{I}_{\mathbf{R}}^{(\omega)} \\ \Pi^{j}(\mathtt{f}) & \text{if } \nu \notin \mathbb{I}_{\mathbf{R}}^{(\omega)} \end{cases}$$

$$\text{(br)}\quad \mathtt{br}^{j+1}(\nu_1,\nu_2,\nu_3) \overset{1}{\mapsto}_1 \begin{cases} \Pi^{j}(\nu_1) & \text{if } \nu_3 \neq \mathtt{f}, \\ \Pi^{j}(\nu_2) & \text{if } \nu_3 = \mathtt{f} \end{cases} \qquad \text{(pred)}\quad \mathtt{pred}^{j+1}(\mathtt{succ}^{k+1}(\nu)) \overset{1}{\mapsto}_1 \Pi^{\min(j,k)}(\nu)$$

$$\text{(fst)}\quad \mathtt{fst}^{j+1}(\mathtt{pr}^{k+1}(\nu_1,\nu_2)) \overset{1}{\mapsto}_1 \Pi^{\min(j,k)}(\nu_1) \qquad \text{(snd)}\quad \mathtt{snd}^{j+1}(\mathtt{pr}^{k+1}(\nu_1,\nu_2)) \overset{1}{\mapsto}_1 \Pi^{\min(j,k)}(\nu_2)$$

$$\text{(outl)}\quad \mathtt{outl}^{j+1}(\mathtt{inl}^{k+1}(\nu)) \overset{1}{\mapsto}_1 \Pi^{\min(j,k)}(\nu) \qquad \text{(outr)}\quad \mathtt{outr}^{j+1}(\mathtt{inr}^{k+1}(\nu)) \overset{1}{\mapsto}_1 \Pi^{\min(j,k)}(\nu)$$

Note that $\Pi^{j}(\mathtt{t}) = \mathtt{t}$ if $j > 0$ and **susp** otherwise. Also note that in (**beta**) the label on `cut`, $\min(j,k)$, is non-zero since otherwise $\Pi^{\min(j,k)}(\nu) = \mathbf{susp}$.

Labelled contexts $\widehat{C}$ are tuples $<C, \mathrm{L}>$ for C a context for $\widehat{\mathbb{E}}^{+}$. Note L does not label the hole. Labelled reduction contexts $\widehat{R}$ are tuples $<R, \mathrm{L}>$ where R is a reduction context for $\mathbb{E}^{+}$.

Definition ($\overset{1}{\mapsto}$): The reduction relation $\overset{1}{\mapsto}$ is the reflexive transitive closure of the relation generated by the following clauses:

$$\text{(susp)}\quad \hat{e}_0 \overset{1}{\mapsto}_1 \mathbf{susp} \Rightarrow \widehat{R}[\hat{e}_0] \overset{1}{\mapsto} \mathbf{susp} \qquad \text{(lred)}\quad \hat{e}_0 \overset{1}{\mapsto}_1 \hat{e}_1 \Rightarrow \widehat{R}[\hat{e}_0] \overset{1}{\mapsto} \widehat{R}[\hat{e}_1]$$

Definition (erasure): $(<e, \mathrm{L}>)^{*} = e'$ where e' is e with `cut`'s removed, i.e. all occurrences of $\mathtt{cut}(e_0)$ replaced by e_0.

The sense in which labelled expressions approximate their unlabelled counterparts is expressed in part in the following lemma.

Lemma (labelled computation):

(i) $\hat{e} \in \widehat{\mathbb{E}}_{\emptyset}$ and $\hat{e} \overset{1}{\mapsto} \nu$ implies $e \mapsto v$ where $(\hat{e})^{*} = e$ and $(\nu)^{*} = v$.

(ii) $e \in \mathbb{E}_{\emptyset}$ and $e \mapsto v$ implies $(\exists \mathrm{L}, \nu)(<e, \mathrm{L}> \overset{1}{\mapsto} \nu)$ where $(\nu)^{*} = v$.

(iii) If $<e_0, \mathrm{L}_0> \overset{1}{\mapsto} <e_1, \mathrm{L}_1>$ then for $\mathrm{L}_0' \geq \mathrm{L}_0$, $<e_0, \mathrm{L}_0'> \overset{1}{\mapsto} <e_1, \mathrm{L}_1'>$ and $\mathrm{L}_1' \geq \mathrm{L}_1$.

(iv) If $<e, \mathrm{L}_0> \overset{1}{\mapsto} <v, \mathrm{L}_1>$ then for $\mathrm{L}_0' \leq \mathrm{L}_0$, either $<e, \mathrm{L}_0'> \overset{1}{\mapsto} <v, \mathrm{L}_1'>$ and $\mathrm{L}_1' \leq \mathrm{L}_1$ or $<e, \mathrm{L}_0'> \overset{1}{\mapsto} \mathbf{susp}$.

(v) If $<e, \mathrm{L}_0> \overset{1}{\mapsto} <v, \mathrm{L}_1>$, then $\mathrm{L}_0(e) \geq \mathrm{L}_1(v)$.

4. Equivalence of labelled expressions

In this section we study operational equivalence for labelled expressions and directed sets of labelled expressions. The definitions are the obvious generalizations from the unlabelled case and the key properties, such as the (**ciu**) theorem, continue to hold. The main theorem is the approximation theorem, which states that expressions in the unlabelled language are equal ($e_0 \cong e_1$) if their sets of labelled expressions are equal ($\tau(e_0) \mathrel{\widehat{\cong}}_S \tau(e_1)$). This in turn allows for a simple proof of the least fixed point theorem to be given, not requiring induction on computation length.

Definition ($\downarrow\ \wr\ \Uparrow\ \ll$): Let $\hat{e}$ be closed labelled expression. Then: $\hat{e}$ is *defined* (written $\downarrow \hat{e}$) if it reduces to a finite value. $\hat{e}$, is a *stuck state* iff it is not a value and does not reduce; $\hat{e}$ is *stuck* (written $\wr\, \hat{e}$) if it reduces to a stuck state; $\hat{e}$, *suspends* (written $\Uparrow \hat{e}$) iff $\hat{e} \overset{1}{\mapsto}$ **susp**. A closed labelled expression e_0 is *less defined* than another closed labelled expression e_1 (written $e_0 \ll e_1$) iff whenever e_0 is defined (or stuck), e_1 is also defined (resp. stuck).

Definition ($\mathrel{\widehat{\sqsubseteq}}\ \mathrel{\widehat{\cong}}$):

$$\hat{e}_0 \mathrel{\widehat{\sqsubseteq}} \hat{e}_1 \;\Leftrightarrow\; (\forall \widehat{C} \in \widehat{\mathbb{C}} \mid \widehat{C}[\hat{e}_0], \widehat{C}[\hat{e}_1] \in \widehat{\mathbb{E}}_\emptyset)(\widehat{C}[\hat{e}_0] \ll \widehat{C}[\hat{e}_1])$$

$$\hat{e}_0 \mathrel{\widehat{\cong}} \hat{e}_1 \;\Leftrightarrow\; \hat{e}_0 \mathrel{\widehat{\sqsubseteq}} \hat{e}_1 \wedge \hat{e}_1 \mathrel{\widehat{\sqsubseteq}} \hat{e}_0$$

The following are direct consequences of the definitions.

Lemma ($\mathrel{\widehat{\sqsubseteq}}$ / $\mathrel{\widehat{\cong}}$-cong):

(i) $\mathrel{\widehat{\sqsubseteq}}$ is reflexive and transitive, $\mathrel{\widehat{\cong}}$ is reflexive, symmetric and transitive.

(ii) $\mathrel{\widehat{\sqsubseteq}}$ is compatible, $\mathrel{\widehat{\cong}}$ is a congruence.

(iii) If $\downarrow \hat{e}_0$ and $\hat{e}_0 \mathrel{\widehat{\sqsubseteq}} \hat{e}_1$ then $\downarrow \hat{e}_1$.

(**labelled ciu**) is the analog of the context lemma for labelled computation.

Theorem (labelled ciu):

$$\hat{e}_0 \mathrel{\widehat{\sqsubseteq}} \hat{e}_1 \;\Leftrightarrow\; (\forall \widehat{R} \in \widehat{\mathbb{R}}, \hat{\sigma} \mid \widehat{R}[\hat{e}_0^{\hat{\sigma}}], \widehat{R}[\hat{e}_1^{\hat{\sigma}}] \in \widehat{\mathbb{E}}_\emptyset)(\widehat{R}[\hat{e}_0^{\hat{\sigma}}] \ll \widehat{R}[\hat{e}_1^{\hat{\sigma}}])$$

Some easy consequences of (**labelled ciu**) are the following. These properties are used later to establish deeper results.

Lemma ($\mathrel{\widehat{\sqsubseteq}}$ / $\mathrel{\widehat{\cong}}$-properties):

(i) If $\hat{e}_0, \hat{e}_1 \in \widehat{\mathbb{E}}_\emptyset$ and $\hat{e}_0 \overset{1}{\mapsto}_1 \hat{e}_1$, then $\hat{e}_0 \mathrel{\widehat{\cong}} \hat{e}_1$.

(ii) $<e, \mathrm{L}_0> \mathrel{\widehat{\sqsubseteq}} <e, \mathrm{L}_1>$ if $\mathrm{L}_0 \leq \mathrm{L}_1$.

(iii) $<\perp, \mathrm{L}> \mathrel{\widehat{\sqsubseteq}} \hat{e}$ for all $\mathrm{L}, \hat{e}$, where $\perp = \mathsf{app}(\lambda x.\mathsf{app}(x,x), \lambda x.\mathsf{app}(x,x))$.

(iv) If $\hat{e} \notin \widehat{\mathbb{V}}$, $\hat{e}^1 \mathrel{\widehat{\cong}} <\perp, \mathrm{L}>$ for all L.

A simple consequence of (i) is that expressions that reduce to the same expression are equivalent. Property (i) can be extended to non-closed expressions, provided we have a suitable notion of reduction for non-closed expressions. To see what can go wrong consider the following three expressions: (a) $\lambda^2 x.0$; (b) $\lambda^3 x.0$; and (c) $\lambda^3 x.\mathsf{app}^2(\lambda^2 x.0, x)$. Clearly (a) and (b) are not equivalent, since (b) has a bigger domain. (c) is equivalent to (a) but not to (b). In particular, it is not the case that $\mathsf{app}^2(\lambda^2 x.0, x)$ is equivalent to 0, since for some instantiations of x the reduction will suspend (for example $x = \mathsf{succ}^2(0)$). Thus unlike the unlabelled case, lambda applications of the form $\mathsf{app}^{k+1}(\lambda^{j+1}x.\hat{e}, x)$ do not symbollicaly reduce.

To give some more concrete understanding of labelled approximation we analyze the equivalence classes of finite value expressions. In the full paper we first show that $\mathbb{N}^{(\omega)}$ is in one-one correspondence with $\mathbb{N}$ modulo operational equivalence. To do this we define a canonical representation, $\widehat{n}$, for each natural number n as follows: $\widehat{0} = 0$ and $\widehat{n+1} = \mathsf{succ}^{n+2}(\widehat{n})$. Each element of $\mathbb{N}^{(\omega)}$ is equivalent to some canonical element, and distinct canonical expressions are inequivalent. A similar analysis is made of the other forms of data. An important result is extensionality.

Lemma (extensionality):

$$(\forall \nu_0, \nu_1 \in \mathbb{L}^{(\omega)})(\nu_0 \mathrel{\widehat{\sqsubseteq}} \nu_1 \;\Leftrightarrow\; (\forall j \in \mathbb{N})(\forall \nu \in \mathbb{V}^{(\omega)})(\mathsf{app}^{j+1}(\nu_0, \nu) \mathrel{\widehat{\sqsubseteq}} \mathsf{app}^{j+1}(\nu_1, \nu)))$$

As a corollary of (**extensionality**) is that finite values can be introduced in stages so that there are finitely many elements at each stage (modulo operational equivalence). We define $\mathbb{V}^{(k+1)} = \{\hat{v} \mid \Pi^{j+1}(\hat{v}) = \hat{v}\}$ and then prove that each $\mathbb{V}^{(n)}$ has finitely many elements modulo $\widehat{\cong}$.

4.1. Directed sets of labelled expressions

We now define directed sets of labelled expressions and the corresponding ordering, $\widehat{\sqsubseteq}_S$.

Definition (directed): $\widehat{E} \subseteq \widehat{\mathbb{E}}$ directed iff for all $\hat{e}_0 \in \widehat{E}, \hat{e}_1 \in \widehat{E}$, there exists $\hat{e}_2 \in \widehat{E}$ such that $\hat{e}_0 \mathrel{\widehat{\sqsubseteq}} \hat{e}_2$ and $\hat{e}_1 \mathrel{\widehat{\sqsubseteq}} \hat{e}_2$.

Important sets of labelled expressions to consider are the τ-sets. The set $\tau(e)$ is the operational analogue of all the finite elments approximating some domain element e.

Definition (τ): $\tau \in \mathbb{E} \to \mathcal{P}(\widehat{\mathbb{E}})$ s.t. $\tau(e) = \{\hat{e} \mid (\hat{e})^* = e\}$. $\tau(E) = \bigcup_{e \in E} \tau(e)$.

Lemma (τ-directed): $\tau(e)$ directed for all e.

Definition ($\widehat{\sqsubseteq}_S$ $\widehat{\cong}_S$):

$$\widehat{E}_0 \mathrel{\widehat{\sqsubseteq}_S} \widehat{E}_1 \;\Leftrightarrow\; (\forall \hat{e}_0 \in \widehat{E}_0)(\forall \widehat{C} \in \widehat{\mathbb{C}} \mid \widehat{C}[\widehat{E}_0], \widehat{C}[\widehat{E}_1] \in \mathcal{P}(\widehat{\mathbb{E}}_\emptyset))(\exists \hat{e}_1 \in \widehat{E}_1)(\widehat{C}[\hat{e}_0] \ll \widehat{C}[\hat{e}_1])$$

$$\widehat{E}_0 \mathrel{\widehat{\cong}_S} \widehat{E}_1 \;\Leftrightarrow\; \widehat{E}_0 \mathrel{\widehat{\sqsubseteq}_S} \widehat{E}_1 \wedge \widehat{E}_1 \mathrel{\widehat{\sqsubseteq}_S} \widehat{E}_0$$

Properties of $\widehat{\sqsubseteq}_S$ are analogous to ($\sqsubseteq_S$ **properties**). The most important tool for using labelled expressions to reason about unlabelled expressions is the following theorem.

Theorem (approximation):

(i) For directed $E_i \subseteq \mathbb{E}$, if $\tau(E_0) \mathrel{\widehat{\sqsubseteq}_S} \tau(E_1)$ then $E_0 \sqsubseteq_S E_1$.

(ii) For $e_i \in \mathbb{E}$, if $\tau(e_0) \mathrel{\widehat{\sqsubseteq}_S} \tau(e_1)$ then $e_0 \sqsubseteq e_1$.

4.2. Least fixed point property

As a simple application of these results we show that the usual fixed point combinator computes the least fixed point.

Definition ($\mathtt{Y}_v$): The fixed point combinator $\mathtt{Y}_v$ is defined as

$$\mathtt{Y}_v = \lambda x.\mathtt{let}\{y := \lambda z.\lambda w.\mathtt{app}(x, \mathtt{app}(z,z), w)\}\mathtt{app}(y,y)$$

Definition (f @ k): For any function expression f, define f @ $0 = \lambda x.\ \bot$ and f @ $n+1 = \lambda x.\mathtt{app}(\mathtt{app}(f, f \mathbin{@} n), x)$.

Lemma (fixed point):

(i) $\{\mathtt{app}(\mathtt{Y}_v, f)\} \cong_S \{f \mathbin{@} n \mid n \in \mathbb{N}\}$

(ii) $\mathtt{app}(\mathtt{Y}_v, f) \cong \mathtt{app}(f, \mathtt{app}(\mathtt{Y}_v, f))$, and $(\forall \lambda x.e)(\lambda x.e \cong \mathtt{app}(f, \lambda x.e) \Rightarrow \mathtt{Y}_v(f) \sqsubseteq_{\approx} \lambda x.e)$

Proof : (ii) follows from (i) by straightforward reasoning, see [13] for a proof. For (i), we first compute to get $\mathtt{app}(\mathtt{Y}_v, f)) \cong \mathtt{app}(U_f, U_f)$, where

$$U_f = \lambda z.\lambda w.\mathtt{app}(\mathtt{app}(f, \mathtt{app}(z,z)), w),$$

and will show

$$\{\tau(\mathtt{app}(U_f, U_f))\} \mathrel{\widehat{\cong}}_S \tau(\{f \mathbin{@} n \mid n \in \mathbb{N}\}).$$

Pick arbitrary $\mathtt{app}^{i_1}(\widehat{U}_f, \widehat{U}_f) \in \tau(\mathtt{app}(U_f, U_f))$. Proceed by induction on i_1 to show

$$\mathtt{app}^{i_1}(\widehat{U}_f, \widehat{U}_f) \mathrel{\widehat{\sqsubseteq}_{\approx}} \hat{f} \mathbin{@} n$$

for some $\hat{f} \mathbin{@} n \in \tau(\{f \mathbin{@} n \mid n \in \mathbb{N}\})$, proving the result.

For $i = 1$ the result is direct from ($\widehat{\sqsubseteq}_{\approx}$ / $\widehat{\cong}$**-properties.iii,iv**). Assume true for smaller i values. We will write labels $< i$ to indicate the label is some value smaller than i. By ($\widehat{\sqsubseteq}_{\approx}$ / $\widehat{\cong}$**-properties.i**), used twice, and then the induction hypothesis, we have

$$\begin{aligned}
&\mathtt{app}^{i_1}(\widehat{U}_f, \widehat{U}_f)\\
&\quad \mathrel{\widehat{\cong}} \lambda^{<i_1} w.\mathtt{cut}^{<i_1}(\mathtt{app}^{i_2}(\mathtt{app}^{i_3}(\hat{f}, \mathtt{app}^{i_4}(\widehat{U}_f^{<i_1}, \widehat{U}_f^{<i_1})), w))\\
&\quad \mathrel{\widehat{\cong}} \lambda^{<i_1} w.\mathtt{cut}^{<i_1}(\mathtt{app}^{i_2}(\mathtt{app}^{i_3}(\hat{f}, \mathtt{app}^{<i_1}(\widehat{U}_f^{<i_1}, \widehat{U}_f^{<i_1})), w))\\
&\quad \mathrel{\widehat{\sqsubseteq}_{\approx}} \lambda^{<i_1} w.\mathtt{cut}^{<i_1}(\mathtt{app}^{i_2}(\mathtt{app}^{i_3}(\hat{f}, \hat{f} \mathbin{@} (i_1 - 1)), w))\\
&\quad = \hat{f} \mathbin{@} i_1
\end{aligned}$$

for some $\hat{f} \mathbin{@} i_1 \in \tau(\{f \mathbin{@} n \mid n \in \mathbb{N}\})$. □

5. Ideals and Types

In this section we present an ideal model of types taking ideals to be certain sets of finite values. The presentation and content is largely based on the papers [7, 1]. Differences include lack of $\bot$ in ideals due to the call-by-value framework we work in, lack of directed closure of ideals (order ideals suffice for us) because labelled expressions are a completely finite language, and differences in proofs of contractiveness of type operators due to our labelled expressions versus their finite domain elements.

Definition (Ideal): An *order ideal* $I \subseteq \mathbb{V}^{(\omega)}$ is a $\widehat{\underset{\sim}{\sqsubset}}$-downward closed set of finite values; we let $\mathbb{I}$ be the set of all ideals.

$$I \in \mathbb{I} \Leftrightarrow (\nu_0 \in I \wedge \nu_1 \mathrel{\widehat{\underset{\sim}{\sqsubset}}} \nu_0 \;\Rightarrow\; \nu_1 \in I)$$

Henceforth we shall use the terms ideal and order ideal synonymously. The rank *rank*(ν) of a finite value expression ν is the least n such that $(\exists \nu' \in \mathbb{V}^{(n)})(\nu \mathrel{\widehat{\cong}} \nu')$. Let I_0, I_1 be ideals. We say that ν is a *witness* for I_0 and I_1 if ν belongs to one but not both, i.e. if $\nu \in (I_0 - I_1) \cup (I_1 - I_0)$. The *closeness* *close*(I_0, I_1) of I_0 and I_1 is the least possible rank of a witness for I_0 and I_1, ∞ if no witness exists. Closeness induces a metric on ideals $\mathit{dist}(I_0, I_1) = 2^{-\mathit{close}(I_0,I_1)}$. The set of ideals with metric *dist* forms a complete metric space. A function $F : \mathbb{I}^n \to \mathbb{I}$ is *contractive* (*nonexpansive*) just if there is a real number $0 \leq r < 1$ $(0 \leq r \leq 1)$ such that for all ideals $I_1, \ldots, I_n, I_1', \ldots, I_n'$

$$\mathit{dist}(F(I_1, \ldots, I_n), F(I_1', \ldots, I_n')) \leq r\max\{\mathit{dist}(I_i, I_i') \mid 1 \leq i \leq n\}$$

The key property of contractive mappings on complete metric spaces is the classic Banach fixed-point theorem. We define the semantic analogues of the usual type constructors.

Definition (type constructors):

$$I_0 \mathbin{\hat{+}} I_1 = \{\mathtt{inl}^i(\nu) \mid \nu \in I_0, i \in \mathbb{N}\} \cup \{\mathtt{inr}^i(\nu) \mid \nu \in I_1, i \in \mathbb{N}\}$$

$$I_0 \mathbin{\hat{\times}} I_1 = \{\mathtt{pr}^i(\nu_0, \nu_1) \mid \nu_0 \in I_0, \nu_1 \in I_1, i \in \mathbb{N}\}$$

$$I_0 \mathbin{\widehat{\rightarrow}} I_1 = \{\nu \in \hat{\mathbb{L}} \mid (\forall \nu_0 \in I_0)(\forall i \in \mathbb{N})(\forall \nu_1 \in I_1)((\mathtt{app}^i(\nu, \nu_0) \mapsto \nu_1) \;\Rightarrow\; \nu_1 \in I_1)\}$$

Theorem (contractive): $\hat{+}$, $\hat{\times}$, and $\widehat{\rightarrow}$ are contractive.

We illustrate the techniques in the case of $\widehat{\rightarrow}$.

Proof ($\widehat{\rightarrow}$): Suppose that for some $i < 2$ $I_i \neq I_i'$ and let $J = I_0 \mathbin{\widehat{\rightarrow}} I_1$ and $J' = I_0' \mathbin{\widehat{\rightarrow}} I_1'$, we show that

$$\mathit{close}(J, J') > \min\{\mathit{close}(I_i, I_i') \mid i < 2\}$$

Suppose that ν is a witness of least rank for J and J'. By symmetry we may assume without loss of generality that $\nu \in J$ but $\nu \in J'$. Thus

(a) $(\forall \nu_0 \in I_0)(\forall i \in \mathbb{N})(\forall \nu_1 \in I_1)((\mathtt{app}^i(\nu, \nu_0) \mapsto \nu_1) \;\Rightarrow\; \nu_1 \in I_1)$

(b) $(\exists \nu_0 \in I_0')(\exists i \in \mathbb{N})(\exists \nu_1 \in I_1')((\mathtt{app}^i(\nu, \nu_0) \mapsto \nu_1) \wedge \nu_1 \notin I_1')$

So choose a $\nu_0 \in I_0'$ of least rank with the properties given by (**b**). Note that by ($\widehat{\underset{\sim}{\sqsubset}}$ / $\widehat{\cong}$-**properties.i**) this implies that *rank*$(\nu_0) <$ *rank*(ν). We consider two cases, depending on whether or not $\nu_0 \in I_0$.

Case $\nu_0 \notin I_0$: In this case ν_0 is a witness for I_0 and I_0' of less rank than ν. $\square$

Case $\nu_0 \in I_0$: In this case we may assume by (**a**) and (**b**) that $\mathsf{app}^i(\nu, \nu_0) \mapsto \nu_1$, $\nu_1 \in I_0$ and $\nu_1 \notin I_1'$. Furthermore by (**labelled computation.v**) $\mathit{rank}(\nu_1) < \mathit{rank}(\nu)$. Thus ν_1 is a witness for I_1 and I_1' of less rank than ν. $\square_{\widehat{\rightarrow}}$

Suppose that $f : \mathbb{I}^{n+1} \to \mathbb{I}$ is a function of $n+1$ arguments. We define two functions $\widehat{\forall} f$ and $\widehat{\exists} f$ which map $\mathbb{I}^n$ to $\mathbb{I}$, by quantifying over f's first argument:

$$(\widehat{\forall} f)(J_1, \ldots, J_n) = \bigcap_{I \in \mathbb{I}} f(I, J_1, \ldots, J_n) \quad (\widehat{\exists} f)(J_1, \ldots, J_n) = \bigcup_{I \in \mathbb{I}} f(I, J_1, \ldots, J_n)$$

In a similar vein if $f : \mathbb{I}^{n+1} \to \mathbb{I}$ is contractive in its first argument then we define $(\widehat{\mu} f)$, which map $\mathbb{I}^n$ to $\mathbb{I}$, by

$$(\widehat{\mu} f)(J_1, \ldots, J_n) = I \qquad \text{where } I \in \mathbb{I} \text{ is the unique fixed point of } \lambda X. f(X, J_1, \ldots, J_n)$$

The following theorems appear in [7], their proofs in our situation are identical.

Lemma ($\widehat{\forall} f$ $\widehat{\exists} f$ $\widehat{\mu} f$): If $f : \mathbb{I}^{n+1} \to \mathbb{I}$ is contractive (non-expansive) in its last n arguments, then $\widehat{\forall} f$ and $\widehat{\exists} f$ are contractive (non-expansive) in all their arguments. If $f : \mathbb{I}^{n+1} \to \mathbb{I}$ is contractive then so is $\widehat{\mu} f$.

6. Type Syntax and Semantics

We define type syntax and relations for type meaning and type membership. The difference with the standard ideal construction is types are initially defined to have labelled elements only, and infinite elements are defined to be in a type iff all their labelled versions are. Let $X \in \mathbb{X}_\tau$ be type variables, let the type syntax $\mathbb{T}$ be

$$\mathbb{T} = \mathbf{bool} + \mathbf{nat} + \mathbb{X}_\tau + (\mathbb{T} \times \mathbb{T}) + (\mathbb{T} \to \mathbb{T}) + (\mathbb{T} + \mathbb{T}) + \mu X.\mathbb{T} + \forall X.\mathbb{T} + \exists X.\mathbb{T}$$

Define the well-formed types $\mathbb{T}^{\mathrm{w}}$ as $\mathbb{T}$ excluding types containing subexpressions of the form $\mu X.Q_1 X_1 \ldots Q_n X_n.X$ for $Q_i = \forall$ or $\exists$ and $n \geq 0$. These types are not contractive. For all other types the definition below is sensible ($\widehat{\mu}$ is always applied to a contractive function).

Type environments ρ are finite maps from $\mathbb{X}_\tau$ to $\mathbb{I}$. Define $[\![T]\!]\rho$ to be the meaning function on types, mapping type $T \in \mathbb{T}^{\mathrm{w}}$ and type environment ρ to some $I \in \mathbb{I}$.

Definition (type meaning):

$$\begin{array}{ll}
[\![\mathbf{bool}]\!]\rho = \mathbb{B} & [\![\mathbf{nat}]\!]\rho = \mathbb{N}^{(\omega)} \\
[\![T_0 \to T_1]\!]\rho = [\![T_0]\!]\rho \mathrel{\widehat{\to}} [\![T_1]\!]\rho & [\![T_0 \times T_1]\!]\rho = [\![T_0]\!]\rho \mathrel{\widehat{\times}} [\![T_1]\!]\rho \\
[\![T_0 + T_1]\!]\rho = [\![T_0]\!]\rho \mathrel{\widehat{+}} [\![T_1]\!]\rho & [\![\forall X.T]\!]\rho = \widehat{\forall}((\lambda I \in \mathbb{I})[\![T]\!]\rho\{X := I\}) \\
[\![\exists X.T]\!]\rho = \widehat{\exists}((\lambda I \in \mathbb{I})[\![T]\!]\rho\{X := I\}) & [\![\mu X.T]\!]\rho = \widehat{\mu}((\lambda I \in \mathbb{I})[\![T]\!]\rho\{X := I\})
\end{array}$$

Lemma (μ-unrolling): $[\![\mu X.T]\!]\rho = [\![T^{\{X := \mu X.T\}}]\!]\rho$

We define what it means for closed expressions to inhabit types; the open case is the standard generalization.

Definition ($\widehat{\models}^{\nu}$ $\widehat{\models}$ $\models$)**:**

(i) For closed ν, $\widehat{\models}^{\nu}\ \nu : T$ iff $\nu \in [\![T]\!]$.

(ii) $\widehat{\models}\ \hat{e} : T$ iff $\hat{e} \overset{1}{\mapsto} \nu$ implies $\widehat{\models}^{\nu}\ \nu : T$.

(iii) For closed e, $\models e : T$ iff $\widehat{\models}\ \hat{e} : T$ for all $\hat{e} \in \tau(e)$.

Lemma (faithful):

(i) $\hat{e}_0 \mathrel{\widehat{\underset{\sim}{\sqsubset}}} \hat{e}_1$ and $\widehat{\models}\ \hat{e}_1 : T$ implies $\widehat{\models}\ \hat{e}_0 : T$.

(ii) $e_0 \mathrel{\underset{\sim}{\sqsubset}} e_1$ and $\models e_1 : T$ implies $\models e_0 : T$.

Lemma (fixed point typing): If $\models f @ n : T$ for all $n \in \mathbb{N}$ then $\models \mathtt{Y}_v(f) : T$.

Proof : For all $\hat{e} \in \tau(\mathtt{Y}_v(f))$, $\hat{e} \mathrel{\widehat{\underset{\sim}{\sqsubset}}} \hat{e}'$ for some $\hat{e}' \in \{f @ n \mid n \in \mathbb{N}\}$ by (**fixed point.i**). $\models \hat{e} : T$ thus follows by (**faithful.i**). $\square$

7. Conclusions

We have defined a new tool for semantic analysis of programs by operational means. Labelled expressions are an operational analogue of finite elements of domain theory. This enriches the power of operational semantics, by enabling new constructions such as the ideal model of types to be recast inside a completely operational framework.

We conclude with some conjectures on which we are currently working. These are important properties, but perhaps surprisingly are not necessary for any uses of the labelled expressions we came across.

Conjecture (normalization): For all $\hat{e} \in \widehat{\mathbb{E}}_{\emptyset}$, $\downarrow \hat{e}$ or $\wr\, \hat{e}$ or $\Uparrow \hat{e}$.

Conjecture (finiteness): $\hat{e} \mathrel{\widehat{\underset{\sim}{\sqsubset}}}_S \widehat{E}$ implies $(\exists \hat{e}_0 \in \widehat{E})\hat{e} \mathrel{\widehat{\underset{\sim}{\sqsubset}}} \hat{e}_0$.

An open problem we had hoped this method would solve but has not helped with is

Conjecture (uniqueness): All $\mathtt{F}$ such that $\mathtt{app}(\mathtt{F}, f) \cong \mathtt{app}(f, \mathtt{app}(\mathtt{F}, f))$ are equivalent.

Another desirable result would be to have the converse of the approximation property. This would prove very useful in axiomatizing labeled expressions, allowing to go from equivalences on unlabeled expressions to equivalences on sets of labelled expressions.

Conjecture (approximation): For $e_i \in \mathbb{E}$, $\tau(e_0) \mathrel{\widehat{\underset{\sim}{\sqsubset}}}_S \tau(e_1)$ if $e_0 \mathrel{\underset{\sim}{\sqsubset}} e_1$.

Further work needs to be done to establish the precise connection between labelled expressions and the finite projection operations π_n^m and π_n of domain theory. These operations are derived from the limit construction of solutions to domain equations [12]. These projection operations may be represented by closed expressions of our language (see [1]), and thus serve to establish a useful connection between labelled computations and their unlabelled counterparts.

8. References

[1] M. Abadi, B. Pierce, and G. Plotkin. Faithful ideal models for recursive polymorphic types. ***International Journal of Foundations of Computer Science***, 2(1):1–21, 1991.

[2] H. Barendregt. ***The lambda calculus: its syntax and semantics***. North-Holland, 1981.

[3] M. Felleisen. ***The Calcului of Lambda-v-cs Conversion: A Syntactic Theory of Control and State in Imperative Higher-Order Programming Languages***. PhD thesis, Indiana University, 1987.

[4] J.M.E. Hyland. A survey of some useful partial order relations on terms of the lambda calculus. In Böhm C, editor, ***Lambda Calculus and Computer Science Theory***, number 37 in Lecture Notes in Computer Science, pages 83–95. Springer Verlag, 1975.

[5] J.M.E. Hyland. A syntactic characterization of the equality in some models for the lambda calculus. ***Journal of the London Mathematical Society***, 12(2):361–370, 1976.

[6] J.-J. Levy. An algebraic interpretation of the λ-β-k–calculus and a labelled λ-calculus. In Böhm C, editor, ***Lambda Calculus and Computer Science Theory***, number 37 in Lecture Notes in Computer Science, pages 147–165. Springer Verlag, 1975.

[7] D. MacQueen, G. Plotkin, and R. Sethi. An ideal model for recursive polymorphic types. ***Information and Computation***, 71:95–130, 1986.

[8] I. A. Mason and C. L. Talcott. Equivalence in functional languages with effects. ***Journal of Functional Programming***, 1:287–327, 1991.

[9] A. R. Meyer and K. Sieber. Towards fully abstract semantics for local variables: Preliminary report. In ***15th ACM Symposium on Principles of Programming Languages***, pages 191–208, 1988.

[10] P.W. O'Hearn and R.D. Tennent. Semantics of Local Variables. Technical Report ECS-LFCS-92-192, Laboratory for foundations of computer science, University of Edinburgh, 1992.

[11] G. Plotkin. Call-by-name, call-by-value and the lambda-v-calculus. ***Theoretical Computer Science***, 1:125–159, 1975.

[12] D. S. Scott. Continuous lattices. In F. W. Lawvere, editor, ***Toposes: Algebraic Geometry and Logic***, volume 500 of ***Lecture notes in mathematics***, pages 97–136. Springer, Berlin, 1972.

[13] S. F. Smith. From operational to denotational semantics. In ***MFPS 1991***, volume 598 of ***Lecture Notes in Computer Science***, pages 54–76. Springer-Verlag, 1992.

[14] C. P. Wadsworth. The Relation between Computational and Denotational Properties for Scotts D_∞-models of the Lambda Calculus. ***SIAM Journal of Computing***, 5(3):489–521, 1976.

[15] C. P. Wadsworth. Approximate reduction and the lambda calculus models. ***SIAM Journal of Computing***, 7(3):337–356, 1978.