From Operational to Denotational Semantics

Scott F. Smith
Department of Computer Science
The Johns Hopkins University
Baltimore, MD 21218
scott@cs.jhu.edu

September 2, 1992

Abstract

In this paper it is shown how operational semantic methods may be naturally extended to encompass many of the concepts of denotational semantics. This work builds on the standard development of an operational semantics as an interpreter and operational equivalence. The key addition is an operational ordering on *sets* of terms. From properties of this ordering a closure construction directly yields a fully abstract continuous cpo model. Furthermore, it is not necessary to construct the cpo, for principles such as soundness of fixed-point induction may be obtained by direct reasoning from this new ordering. The end result is that traditional denotational techniques may be applied in a purely operational setting in a natural fashion, a matter of practical importance for developing semantics of realistic programming languages.

1 Introduction

This paper aims to accomplish a degree of unification between operational and denotational approaches to programming language semantics by recasting denotational concepts inside a purely operational framework. These concepts include notions of ordering $a \sqsubseteq b$, directed set, complete partial order, monotonicity, continuity, least fixed point principle, and fixed point induction.

There is an important application of this. The denotational concepts listed above are in fact useful for giving rigorous semantics to large classes of programming languages and logics. Theories of program equivalence arise directly from domain orderings. The notions of directed set, least upper bound and continuity lead to a least fixed-point induction principle. Ideal sets, defined in terms of least upper bounds, are a general notion of data type[MPS84]. However, denotational semantics suffers some shortcomings that limits its usefulness, the most well-known being the full abstraction problem. For most languages, equality of the domain does not exactly correspond to the operational equality. A full discussion of this issue is outside the scope of this paper; see for instance [Sto88, Blo90].

This paper outlines an approach whereby a purely operational theory of a programming language may be enriched in a natural fashion to incorporate the "denotational" properties listed above. In this paper we carry out the program for a small deterministic functional language L. What we see as the interest of the method is not its applicability to small languages, however. We see the method as applicable to a wide class of languages, including nondeterministic, concurrent, object-oriented, imperative, and typed languages (with polymorphic, recursive and module constructors), and plan to demonstrate this applicability in future work.

Let us now consider the paper in more detail. In section 3, a now-standard development of a basic operational semantics for the language L is presented. This consists of an interpreter, an

operational ordering $a \lesssim b$, and corresponding equivalence $a \cong b$. These results are largely known in the literature, although some new, simplified proofs are presented, and a large number of properties are proved about $\lesssim$.

In section 4 new results are presented to extend the basic operational semantics to encompass concepts of domain theory. The objective here is not just to mimic concepts of domain theory, but to derive generally useful semantic principles that extend the basic operational semantics. In fact, the operational concepts developed here differ from the domain development in some significant ways. The operational ordering $\lesssim$ does not form a cpo by simple computability arguments, and "least upper bound" is not very interesting operationally because the operational analogue of continuity fails.

However, there is a new concept that allows analogous principles to be derived. The idea, informally, is to view $\lesssim$-directed sets of terms A as one large cooperating computation. The terms in A will never outright contradict each other, for instance by having one compute to 3 and the other 5, because then the set would not have been directed. So, we can view them as cooperating, ignoring those elements that diverge. This leads to the definitions of an ordering $A \lesssim_S B$ and equivalence $A \cong_S B$ defined on directed sets of computations A and B. For example, for some function f and everywhere diverging function $\lambda x.\bot$, the $\lesssim$-directed set $\{\lambda x.\bot, f(\lambda x.\bot), f(f(\lambda x.\bot)), \ldots, f^k(\lambda x.\bot), \ldots\}$ is $\cong_S$ to the set $\{Y(f)\}$, where Y is an appropriate fixed-point operator. This is because for any particular halting computation that uses $Y(f)$, only finitely many recursive calls of f will be executed, so f unrolled that many times suffices in place of $Y(f)$. A useful theory of the ordering $\lesssim_S$ is developed, showing that $\lesssim_S$ successfully takes the place of notions of least upper bound and continuity found in domain theory.

This allows results (such as Scott fixed-point induction, section 4.3) which have in the past needed continuous cpo models for justification to be proven directly using operational orderings. One concrete example is all the rules of the LCF programming logic [GMW79] can be shown sound by this method. This might at first seem surprising, because the rules of LCF were inspired by Scott's domain model. In fact, to the author's knowledge there is no equational principle which is derivable in a fully abstract domain model and not derivable independently in this setting.

To further justify the power of these concepts, a fully abstract cpo is directly constructed from properties of $\lesssim$ and $\lesssim_S$ in section 5. The net result is evidence for the similarity of this operational approach to denotational semantics and thus the usefulness of these operational principles. We should emphasize however that our purpose in giving this cpo construction is *not* to give another method for constructing fully abstract cpos, for there is really no need to work in a cpo model; the extra elements added by the cpo closure merely complicate proofs, and it is easier to work directly over directed sets of terms. Since researchers are more familiar with cpo models, the utility of this observation will perhaps not be immediately evident.

2 Related Work

There has already been considerable work in developing the basic operational notions of ordering $\lesssim$ and corresponding equivalence $\cong$ that we present in the next section. Numerous basic properties are desired, including the respect of computation, for instance requiring $plus(3,2) \cong 5$, and congruence. A number of researchers have developed methods to directly prove these and other basic properties without recourse to domain theory; see for instance Milner [Mil77], Howe [How89], and Mason and Talcott [MT91]. It also should be mentioned that for simple functional languages like L studied here, denotational models can be altered by various means to achieve fully abstract models [EHdR92, Abr90].

Milner, in one of the classic papers of denotational semantics, constructs a fully abstract cpo starting from an operational ordering on terms $\leq$. Milner proves both respect of computation and congruence by purely syntactic means. His construction of the cpo is quite different than that of this paper and it leaves him with no completely operational characterization as is possible here: he cannot stop short of constructing a cpo and still have a usable theory. Furthermore, his language is typed and based on combinators rather than λ, greatly simplifying the mathematics.

Howe proves congruence for a class of languages with a particular style of operational semantics. This schema captures simple functional programming language features, but does not extend to languages with advanced features such as effects or control operators. Since all such schemas must leave out interesting languages, no schema is attempted here. Instead, we prove the results for one particular language; one can work from the proofs herein to generalize the results to other languages.

Mason and Talcott have proven respect of computation and congruence for more complex languages than the language L presented here—their languages have continuations as first-class objects and a global state. The development of the basic operational semantics of the next section is based on the methods of Mason and Talcott. Their work complements ours because it indicates the new results presented in section 4 may be expected to apply to more complex languages.

3 Basic Operational Semantics

For this paper, a simple untyped call-by-value functional language with numbers and pairing L is studied. We choose call-by-value evaluation because it is slightly more challenging than call-by-name; an earlier version of this work incorporated call-by-name evaluation [Smi92a].

Definition 3.1 The expressions E of L are the least collection constructed from

(i) countably many variables (never directly written),

(ii) value terms $0, 1, 2, \ldots$, $\langle v_0, v_1\rangle$, $\lambda x.a$,

(iii) and computation terms $pred(a)$, $succ(a)$, $if\_zero(a; b; c)$, $\pi_1(a)$, $\pi_2(a)$, $a(b)$,

where a, b, and c are terms, v_0 and v_1 are value terms, and x is a variable.

Let a–f range over terms (f is expected to be a function term), v range over value terms, and x–z range over variables. Conventions of bound and free variables and substitution $a\{x := a'\}$ are as follows: $\lambda x.a$ binds free occurrences of x in a, and $a\{x := a'\}$ denotes the substitution of a' for free occurrences of x in a, renaming bound variables of a to avoid capture. A term is *closed* if all variables occurring are bound. $a = b$ means a and b are identical modulo α-conversion.

First, an operational semantics for untyped computations is given. We present a rewriting interpreter like the $\xrightarrow{v}$ relation of [Plo75], using the more convenient notion of a reduction context (a.k.a. evaluation context) taken from [FFK87].

Closed terms are either computations or values. Call-by-value evaluation is deterministic, so at most one reduction applies. $C[\circ]$ denotes a *context*, a term C with occurrences of atomic holes "$\circ$" in which another term a may be placed, $C[a]$. This may lead to free variables in a being captured by C. We use specialized *reduction contexts* R to isolate the next redex to be reduced.

Definition 3.2 A reduction context $R[\bullet]$ is inductively of the form

- (a hole) or $R'[\bullet](a)$, or $v(R'[\bullet])$, or $if\_zero(R'[\bullet]; a; b)$, or $pred(R'[\bullet])$, or $succ(R'[\bullet])$, or $\pi_1(R'[\bullet])$, or $\pi_2(R'[\bullet])$, or $\langle R'[\bullet], a\rangle$, or $\langle v, R'[\bullet]\rangle$,

where $R'[\bullet]$ is a reduction context, a and b are terms and v is a value.

$R[\bullet]$ can be viewed as a syntactic form of the continuation for the computation in the hole. Reduction contexts are used below as follows. In order to perform one step of computation of some term a, it is factored into $a = R[a']$ for some reduction context $R[\bullet]$ and redex a', and the redex then contracted. "$\bullet$" is used to represent the (unique) hole in a reduction context and not "$\circ$", because it will sometimes be necessary to use reduction contexts that also have occurrences of normal holes $\circ$, written $R[\circ][\bullet]$.

DEFINITION 3.3 $\mapsto_1$, single-step computation, is the least relation on closed terms such that, for closed $R[\bullet], b, c, \lambda x.a, v, v_0, v_1$,

$R[(\lambda x.a)(v)] \mapsto_1 R[a\{x := v\}]$
$R[\pi_1(\langle v_0, v_1\rangle)] \mapsto_1 R[v_0]$
$R[\pi_2(\langle v_0, v_1\rangle)] \mapsto_1 R[v_1]$
$R[succ(v_0)] \mapsto_1 R[v_1]$, where v_1 is one larger than the number v_0
$R[pred(v_0)] \mapsto_1 R[v_1]$, where v_1 is one smaller than v_0, and 0 if v_0 is 0
$R[if\_zero(0; b; c)] \mapsto_1 R[b]$
$R[if\_zero(v; b; c)] \mapsto_1 R[c]$, where $v \in \{1, 2, \ldots\}$.

For each line $R[a] \mapsto_1 R[b]$ in the above definition, a is a *redex* and b its *contractum*. Note that computation is defined for closed terms only; open terms are incomplete programs. We will be able to reason about them, but not directly compute them. It is necessary to show that every term a can be uniquely factored into a form $a = R[a']$ where a' is a redex to guarantee a deterministic evaluator for L has been defined [FFK87].

LEMMA 3.4 (UNIQUE FACTORIZATION) If $a \mapsto_1 b$ then $a = R[c]$ for some *unique* reduction context $R[\bullet]$ and redex c.

PROOF. By induction on the term structure. If a is a value the result is trivial; suppose inductively that terms smaller than a can be factored, and proceed by cases on the outermost constructor of a. If $a = d(e)$, then there are three subcases: if d and e are values, let R be $(\bullet)$, and c be a. If d is a value but not e, inductively $e = R'[f]$ uniquely, so let R be $d(R'[\bullet])$, and c be f. If neither are values, inductively $d = R'[f]$ uniquely, so let R be $(R'[\bullet])(e)$, and c be f. The other possibilities for a proceed similarly.
QED.

In proofs below it is often desirable to factor an instantiated context $C[a]$ into reduction context form; the following lemma defines the proper factoring.

COROLLARY 3.5 (UNIQUE CONTEXT FACTORIZATION) If $C[a] \mapsto_1 a'$, there exists unique $R[\circ][\bullet]$ and $C'[\circ]$ such that $C[\circ] = R[\circ][C'[\circ]]$, and $C'[a]$ is a redex or $C'[\circ] = \circ$.

PROOF. $C[a] = R'[c]$ for redex c uniquely by above. Consider where the occurrences of a in $C[a]$ lie in $R'[c]$. If they all are subterms of either R' or of c, $R'[c]$ may be re-written $R[a][C'[a]]$. The only other possibility is the redex c is a subterm of a, in which case we let $C'[\circ] = \circ$ and have $C[a] = R[a][a]$.
QED.

DEFINITION 3.6 Over closed terms,

(i) $\mapsto^*$ is the reflexive, transitive closure of $\mapsto_1$.

(ii) $\mapsto$ is $\mapsto^*$ with its range restricted to values only.

LEMMA 3.7 $\mapsto_1$ and $\mapsto$ possess the following properties:

(i) (locality of evaluation) If $R[a] \mapsto_1 a'$, then a is closed, $a \mapsto_1 b$, and $a' = R[b]$.

(ii) (reflexivity for values) $v \mapsto v$

(iii) (determinism) If $a \mapsto v$ and $a \mapsto v'$ then $v = v'$.

Termination $a\downarrow$ means $a \mapsto v$ for some v, and nontermination $a\uparrow$ means $\neg a\downarrow$; $\bot \stackrel{\text{def}}{=} (\lambda x.x(x))(\lambda x.x(x))$, $\bot_\lambda \stackrel{\text{def}}{=} \lambda x.\bot$, and call-by-value fixed point combinator $Y_v \stackrel{\text{def}}{=} \lambda y.(\lambda x.\lambda z.y(x(x))(z))(\lambda x.\lambda z.y(x(x))(z))$.

3.1 Ordering and Equivalence

There is only one universally accepted notion of operational equivalence, *observational congruence* (also called operational equivalence) $a \cong_{\text{obs}} b$. a and b are observationally congruent just when a and b behave identically when placed in any closing program context $C[\circ]$. It is trivially a congruence.

DEFINITION 3.8 (i) $a \lesssim_{\text{obs}} b$ iff $C[a]\downarrow$ implies $C[b]\downarrow$, for all contexts $C[\circ]$ such that $C[a]$ and $C[b]$ are closed.

(ii) $a \cong_{\text{obs}} b$ iff $a \lesssim_{\text{obs}} b$ and $b \lesssim_{\text{obs}} a$.

LEMMA 3.9 $\lesssim_{\text{obs}}$ is substitutive, i.e. $a \lesssim_{\text{obs}} b$ implies $C[a] \lesssim_{\text{obs}} C[b]$, and $\cong_{\text{obs}}$ is a congruence.

PROOF. To show $C[a] \lesssim_{\text{obs}} C[b]$, by the definitions we must show $C'[C[a]]\downarrow$ implies $C'[C[b]]\downarrow$, and this is direct from assumption $a \lesssim_{\text{obs}} b$, picking $C[\circ]$ there to be $C'[C[\circ]]$.
QED.

Some alternate notions of operational equivalence $a \cong_{\text{alt}} b$ have been developed that can be characterized as a pruning of the set of contexts for which a and b must behave identically to be considered equal. Since contexts are pruned, $a \cong_{\text{obs}} b$ (same in all contexts) always implies $a \cong_{\text{alt}} b$ (same in a subset of all contexts). But, the significance of these alternate orderings is the pruned contexts are not needed to distinguish terms: $\cong_{\text{alt}}$ is exactly $\cong_{\text{obs}}$. What is gained then is not a different notion of equivalence, but an easier route to proving observational congruence. With fewer contexts, it is easier to show $a \cong_{\text{alt}} b$ than to show $a \cong_{\text{obs}} b$.

We define such an alternate notion in this paper, restricting contexts to be **c**losed **i**nstances of all **u**ses of an expression. This equivalence is thus called **ciu** equivalence $\cong_{\text{ciu}}$, following [MT91]. $a \cong_{\text{ciu}} b$ means a and b behave identically when closed (the *closed instances* part) and placed in any reduction context $R[\bullet]$ (the *uses* part). As alluded to above, we can prove $\cong_{\text{obs}}$ is the same as $\cong_{\text{ciu}}$ (Limited Contexts Lemma, 3.16), so we have gained a simpler characterization of observational congruence. Some other alternate notions are discussed at the end of this section.

Notation is then needed for the closing of a term. Define *closing substitutions* σ to range over finite sequences $\{x_1 := a_1\} \ldots, \{x_n := a_n\}$ where the a_i are closed. a^σ denotes $a\{x_1 := a_1\} \ldots, \{x_n := a_n\}$ and is only well-formed if the result is closed. A value substitution is a substitution where all variables map to value terms. For simplicity $\cong_{\text{ciu}}$ will hereafter be abbreviated $\cong$.

DEFINITION 3.10 (i) For closed a and b, $a \lesssim_0 b$ iff for all reduction contexts $R[\bullet]$, if $R[a]\downarrow$, then $R[b]\downarrow$.

(ii) For arbitrary a and b, $a \lesssim b$ iff for all closing value substitutions σ, $a^\sigma \lesssim_0 b^\sigma$.

(iii) $a \cong b$ iff $a \lesssim b$ and $b \lesssim a$.

LEMMA 3.11 ($\lesssim/\cong$ PROPERITES) (i) $\lesssim$ is transitive and reflexive (a pre-order), and $\cong$ is an equivalence relation;

(ii) $\cong$ is nontrivial, in particular $0 \ncong 1$;

(iii) $\bot \lesssim a$, and $\bot_\lambda \lesssim \lambda x.a$;

(iv) For closed a, $a{\uparrow}$ iff $a \cong \bot$;

(v) $\cong$ respects computation, i.e. $a \cong b$ where a is a redex and b its contractum;

(vi) For closed a and b, if $a \mapsto v$ and $a \lesssim b$, then $b \mapsto v'$, and v and v' have the same outermost constructor;

(vii) $v(a) \cong \bot$ if v is a number or pair;

(viii) $\pi_{\{1,2\}}(v) \cong \bot$ if v is a number or lambda;

(ix) $pred(v) \cong succ(v) \cong if\_zero(v; a; b) \cong \bot$ if v is a pair or lambda;

(x) if $a \cong \bot$ then $R[a] \cong \bot$ for all $R[\bullet]$.

PROOF. Direct from the definitions.
QED.

We briefly mention some related alternate notions of equivalence that have been proposed for deterministic languages. Bloom's *applicative congruence* $\cong_{\text{ap}}$ [Blo90] for the simply typed language PCF may be viewed as a further restriction on the observation contexts to be those applicative contexts $(\circ)(a_0)(a_1)\ldots(a_n)$ that drive the term in the hole to ground type. This notion in fact originates with Milner [Mil77], and Milner proves the *Context Lemma* that implies $\cong_{\text{ap}}$ is exactly $\cong_{\text{obs}}$.*

Abramsky [Abr90] and Howe [How89] define an ordering closely related to $\cong_{\text{ciu}}$ and $\cong_{\text{ap}}$, *applicative bisimulation* $a \cong_{\text{bisim}} b$. As with $\cong$, all closed instances of a and b are taken, but instead of using reduction contexts R or applications $(\circ)(a_0)(a_1)\ldots(a_n)$ to signify all the possible uses, there must exist a bisimulation between the closed instances. For the language L of this paper, alternate equivalences of all of the three styles $\cong_{\text{ciu}}$, $\cong_{\text{ap}}$, $\cong_{\text{bisim}}$ may be defined and will in fact be provably identical relations. We choose to use $\cong_{\text{ciu}}$ for a resulting simplicity of proofs. The important point to be made is the difference between the above alternate notions are more technical than substantive, and the new ideas of this paper presented in section 4 could be developed using any of the above alternate equivalences. The best choice depends on the language and applications intended.

Up to this point an ordering $\cong$ has been defined which respects computation. Four important results are next proven about this ordering: the substitutivity of $\lesssim$, which has as a corollary the congruence of $\cong$ and the equivalence of $\cong$ and $\cong_{\text{obs}}$; two properties which are the operational analogues the least fixed-point and continuity properties of domain theory; and a fixed point induction principle.

*Milner's result is actually slightly different because he defines his ordering over an abstract base type and uses combinators instead, but in spirit they are the same.

3.2 Congruence

The desired theorem is as follows.

THEOREM 3.12 ($\cong$ CONGRUENCE) If $a \cong b$ then $C[a] \cong C[b]$.

The proof given here is derived from the proof found in [MT91]. Mason and Talcott prove this property by direct induction on the length of the computation $C[a]$. Their proof is complicated considerably because in the course of computation substitutions may be performed on the term in the hole, and these substitutions may in turn include terms with holes, *ad nauseum*. All of this must be explicitly kept track of using a generalized notion of hole. Here we take a different approach, breaking the theorem into separate chunks and avoiding the need for generalized holes. The key insight is $C[\circ]$ is constructed step-by-step, i.e. the theorem is proven by induction on $C[\circ]$'s size. The first lemma we prove guarantees the result holds if $C[\circ]$ does not capture variables in a and b. Next, we prove that $a \cong b$ implies $\lambda x.a \cong \lambda x.b$. By repeated application of these two lemmas, any substitutive context can be constructed around a and b.

LEMMA 3.13 (CLOSED SUBSTITUTIVITY) $\sqsubseteq_{\approx_0}$ is substitutive, i.e. for closed a, b and $C[\circ]$, if $a \sqsubseteq_{\approx_0} b$ then $C[a] \sqsubseteq_{\approx_0} C[b]$.

PROOF. To show $C[a] \sqsubseteq_{\approx_0} C[b]$, it suffices to pick arbitrary closed $C[\circ]$, and show $C[a]{\downarrow}$ implies $C[b]{\downarrow}$. Proceed by induction on the number of computation steps taken by $C[a]$, generalizing to arbitrary $C[\circ]$. For the base case, either $C[\circ]$ is outermost a value, in which case the result is trivial, or $C[\circ] = \circ$, in which case the result follows directly by hypothesis with $R[\bullet] = \bullet$.

Assume there is a proof for shorter computations, and consider what happens in one step of computation

$$C[a] \mapsto_1 a'.$$

Using the Unique Context Factorization Corollary (3.5), We may factor $C[\circ] = R[\circ][\![C'[\circ]]\!]$, where $C'[a]$ is a redex or $C'[\circ] = \circ$. Consider the next step of computation

$$R[a][\![C'[a]]\!] \mapsto_1 R[a][\![c]\!].$$

Observe that by inspection of the reduction rules, a occurring outside of the reduction context are untouched when one step of computation is performed. Now, consider $C'[a]$ in a reduction context. Taking the case $C'[\circ] \neq \circ$, inspection of the reduction rules shows a inside $C'[\circ]$ are also just carried over to the right side of the rule, except in a few special cases. Putting off the special cases, assume a is not touched; we may write the single step as

$$R[a][\![C'[a]]\!] \mapsto_1 R[a][\![C''[a]]\!]$$

for some $C''[\circ]$. The right side of the reduction is one step shorter, so applying the induction hypothesis gives $R[b][\![C''[b]]\!]{\downarrow}$, and since $R[b][\![C'[b]]\!] \mapsto_1 R[b][\![C''[b]]\!]$, $R[b][\![C'[b]]\!]{\downarrow}$. Returning to the special cases, if $C'[\circ] = (\circ)(C''[\circ])$,

$$R[a][\![a(C''[a])]\!] \mapsto_1 R[a][\![a'\{x := C''[a]\}]\!], \text{ where } a = \lambda x.a'.$$

By induction hypothesis, $R[b][\![a'\{x := C''[b]\}]\!]{\downarrow}$; thus, $R[b][\![a(C''[b])]\!]{\downarrow}$ by reversing the computation step. This term may also be written $R'[b][\![a]\!]$ for $R' = R[\circ][\![(\bullet)(C''[b])]\!]$; observe R' is a reduction context. Then, from assumption $a \sqsubseteq_{\approx_0} b$, $R'[b][\![b]\!]{\downarrow}$, so reverting to previous notation, $R[b][\![b(C''[b])]\!]{\downarrow}$, proving this case. The other special cases where the hole is touched, $C'[\circ] = \pi_{\{1,2\}}(\circ)$, $pred(\circ)$, $succ(\circ)$, and $if\_zero(\circ; C''[\circ]; C'''[\circ])$, are similar.

The only case remaining is $C'[\circ] = \circ$,

$$R[a]\llbracket a \rrbracket \mapsto_1 R[a]\llbracket a' \rrbracket.$$

Applying the induction hypothesis to the right side treating a' as fixed, $R[b]\llbracket a' \rrbracket\downarrow$, so $R[b]\llbracket a \rrbracket\downarrow$ as well. So, since $a \sqsubseteq_0 b$, $R[b]\llbracket b \rrbracket\downarrow$ directly.
QED.

LEMMA 3.14 (LAMBDA SUBSTITUTIVITY) If $a \sqsubseteq b$ then $\lambda x.a \sqsubseteq \lambda x.b$.

PROOF. Since $\lambda x.a$ and $\lambda x.b$ may contain free variables, we must close them with a substitution σ. Without loss of generality we move σ inside the lambda $\lambda x.(a^\sigma)$, with the restriction that x occurs neither in the domain or range of σ. To show $\lambda x.a^\sigma \sqsubseteq_0 \lambda x.b^\sigma$, following the previous lemma it suffices to show $C[\lambda x.a^\sigma]\downarrow$ implies $C[\lambda x.b^\sigma]\downarrow$ for closed $C[\circ]$, by induction on computation length. Again, apply the Unique Context Factorization Corollary (3.5) to give $C[\circ] = R[\circ]\llbracket C'[\circ] \rrbracket$, where $C'[\lambda x.a^\sigma]$ is a redex or $C'[\circ] = \circ$.

Consider what happens in one step of computation

$$R[\lambda x.a^\sigma]\llbracket C'[\lambda x.a^\sigma] \rrbracket \mapsto_1 R[\lambda x.a^\sigma]\llbracket d \rrbracket.$$

Consider $C'[\lambda x.a^\sigma]$ in the reduction context: if $C'[\circ]$ is not of the form $\circ(C''[\circ])$, inspection of the reduction rules shows the $\lambda x.a^\sigma$ inside $C'[\circ]$ is just carried over to the right side of the reduction rule. So, for these $C'[\circ]$, $\lambda x.a^\sigma$ is not touched; we may write the single step as

$$R[\lambda x.a^\sigma]\llbracket C'[\lambda x.a^\sigma] \rrbracket \mapsto_1 R[\lambda x.a^\sigma]\llbracket C''[\lambda x.a^\sigma] \rrbracket$$

for some $C''[\circ]$. The right side of the reduction is one step shorter, so applying the induction hypothesis gives $R[\lambda x.b^\sigma]\llbracket C''[\lambda x.b^\sigma] \rrbracket\downarrow$, and since $R[\lambda x.b^\sigma]\llbracket C'[\lambda x.b^\sigma] \rrbracket \mapsto_1 R[\lambda x.b^\sigma]\llbracket C''[\lambda x.b^\sigma] \rrbracket$, $R[\lambda x.b^\sigma]\llbracket C'[\lambda x.b^\sigma] \rrbracket\downarrow$.

The only case remaining is then when $C'[\circ] = \circ(C''[\circ])$,

$$R[\lambda x.a^\sigma]\llbracket (\lambda x.a^\sigma)(C''[\lambda x.a^\sigma]) \rrbracket \mapsto_1 R[\lambda x.a^\sigma]\llbracket a^{\sigma\cup\{x:=C''[\lambda x.a^\sigma]\}} \rrbracket.$$

Applying the induction hypothesis to the right side treating $a^{\sigma\cup\{x:=C''[\circ]\}}$ as fixed, we have

$$R[\lambda x.b^\sigma]\llbracket a^{\sigma\cup\{x:=C''[\lambda x.b^\sigma]\}} \rrbracket\downarrow.$$

So, since $a^{\sigma\cup\{x:=C''[\lambda x.b^\sigma]\}} \sqsubseteq b^{\sigma\cup\{x:=C''[\lambda x.b^\sigma]\}}$, $R[\lambda x.b^\sigma]\llbracket b^{\sigma\cup\{x:=C''[\lambda x.b^\sigma]\}} \rrbracket\downarrow$ directly. Thus, by uniformly reversing the computation step, $R[\lambda x.b^\sigma]\llbracket (\lambda x.b^\sigma)(C''[\lambda x.b^\sigma]) \rrbracket\downarrow$.
QED.

THEOREM 3.15 ($\sqsubseteq$ SUBSTITUTIVITY) If $a \sqsubseteq b$ then $C[a] \sqsubseteq C[b]$.

PROOF. Proceed by induction on the structure of $C[\circ]$. The base $C[\circ] = \circ$ is trivial; assume true for smaller contexts. Proceed by cases on the outermost structure of $C[\circ]$.
CASE $C[\circ] = \lambda x.C'[\circ]$: By the induction hypothesis, $C'[a] \sqsubseteq C'[b]$, so $\lambda x.C'[a] \sqsubseteq \lambda x.C'[b]$ by the Lambda Substitutivity Lemma (3.14).
CASE $C[\circ] = C_1[\circ](C_2[\circ])$: Show $(C_1[a](C_2[a]))^\sigma \sqsubseteq_0 (C_1[b](C_2[b]))^\sigma$ for closing σ. By the induction hypothesis we may obtain $(C_i[a])^\sigma \sqsubseteq_0 (C_i[b])^\sigma$ for $i < 2$. Since $(C_1[a])^\sigma$ and $(C_1[b])^\sigma$ are closed, $(C_1[a]^\sigma(C_2[a]^\sigma)) \sqsubseteq_0 (C_1[b]^\sigma(C_2[a]^\sigma))$ by the Closed Substitutivity Lemma (3.13), and by one more application of the Lemma, $(C_1[a]^\sigma(C_2[a]^\sigma)) \sqsubseteq_0 (C_1[b]^\sigma(C_2[b]^\sigma))$. Thus, since application does not introduce any new bindings, $(C_1[a](C_2[a]))^\sigma \sqsubseteq_0 (C_1[b](C_2[b]))^\sigma$.
CASE $C[\circ]$ = outermost other constructors: Proof proceeds similarly to the case for application, since no binding of free variables takes place.
QED.

PROOF OF $\cong$ CONGRUENCE THEOREM (3.12) Direct from the $\sqsubseteq$ Substitutivity Theorem (3.15).
QED.

It is possible to now prove a limited number of contexts suffices for observing differences, and thus $\sqsubseteq$ and $\cong$ are equivalent to $\sqsubseteq_{\text{obs}}$ and $\cong_{\text{obs}}$, respectively. This Lemma is similar to Mason and Talcott's ciu theorem [MT91], Milner's Context Lemma [Mil77], and to what Bloom [Blo90] calls operational extensionality.

LEMMA 3.16 (LIMITED CONTEXTS) (i) $a \sqsubseteq_{\text{obs}} b$ iff $a \sqsubseteq b$

(ii) $a \cong_{\text{obs}} b$ iff $a \cong b$

PROOF. (i) $\Rightarrow$: Show $R[\![a^\sigma]\!]\downarrow$ implies $R[\![b^\sigma]\!]\downarrow$. These terms can equivalently be written as a context $C[\circ]$ with a or b placed in the hole: $C[\circ] = (\lambda x_1, \ldots \lambda x_n.R[\![\circ]\!])(v_1)\ldots(v_n)$, where $\sigma = \{x_1 := v_1\}\ldots\{x_n := v_n\}$; $C[a] \mapsto^* R[\![a^\sigma]\!]$ and $C[b] \mapsto^* R[\![b^\sigma]\!]$. So, since $C[a]\downarrow$ implies $C[b]\downarrow$ by assumption, the result follows.
$\Leftarrow$: If $a \sqsubseteq b$, then $C[a] \sqsubseteq C[b]$ for all closing $C[\circ]$ by the Substitutivity Theorem (3.15), so by the $\sqsubseteq$ / $\cong$ Properties Lemma (3.11) (vi), $a \sqsubseteq_{\text{obs}} b$.
(ii) Direct from (i).
QED.

LEMMA 3.17 (EXTENSIONALITY) (i) All functions are extensional: $\lambda x.a \sqsubseteq \lambda x.b$ iff $(\lambda x.a)(v) \sqsubseteq (\lambda x.b)(v)$ for all values v.

(ii) (η-λ) If $f \cong \lambda x.b$, then $f \cong \lambda y.f(y)$;

(iii) (η-π) If $a \cong \langle b, c\rangle$, then $a \cong \langle \pi_1(a), \pi_2(a)\rangle$;

PROOF. (i), $\Rightarrow$: Direct from the definitions.
(i), $\Leftarrow$: Proof proceeds by the same technique as Lambda Substitutivity Lemma (3.14). Again, we consider what happens in one step of computation

$$R[\lambda x.a^\sigma][\![C'[\lambda x.a^\sigma]]\!] \mapsto_1 R[\lambda x.a^\sigma][\![d]\!].$$

The only difficult case is also when $C'[\circ] = \circ(C''[\circ])$,

$$R[\lambda x.a^\sigma][\![(\lambda x.a^\sigma)(C''[\lambda x.a^\sigma])]\!] \mapsto_1 R[\lambda x.a^\sigma][\![a^{\sigma\cup\{x:=C''[\lambda x.a^\sigma]\}}]\!].$$

Applying the induction hypothesis to the right side treating $a^{\sigma\cup\{x:=C''[\circ]\}}$ as fixed, we have

$$R[\lambda x.b^\sigma][\![a^{\sigma\cup\{x:=C''[\lambda x.b^\sigma]\}}]\!]\downarrow.$$

Thus, by uniformly reversing the computation step, $R[\lambda x.b^\sigma][\![(\lambda x.a^\sigma)(C''[\lambda x.b^\sigma])]\!]\downarrow$. Letting v in the hypothesis be $C''[\lambda x.b^\sigma]$, by the definition of $\sqsubseteq$ we are done. (ii) and (iii) are similar but simpler.
QED.

4 Semantics of Directed Sets

The main contribution of this paper is the definition of an ordering $\sqsubseteq_S$ on directed sets of terms which conceptually replaces the notion of least upper bound of domain theory, because it has a substitutivity property which is an analogue of continuity for functions in a cpo model. That ordering is now defined.

To separate categories, small letters a–f range over terms, capital letters A–F range over sets of terms, and script letters $\mathcal{A}$–$\mathcal{F}$ range over sets of sets of terms. Term constructors implicitly extend pointwise to sets of terms, so for instance $a(B)$ abbreviates $\{a(b) \mid b \in B\}$ and $A(B)$ abbreviates $\{a(b) \mid a \in A, b \in B\}$. Substitutions σ may also act on a set of terms: define $A^\sigma = \{a^\sigma \mid a \in A\}$. A set of terms is closed if all the terms in the set are closed. Recall that σ is a finite sequence of variable substitutions, so some sets of terms A have no closing substitution σ. We thus restrict ourselves hereafter to sets A with finitely many free variables. †

DEFINITION 4.1 a set of terms A is *directed* iff for every $a, b \in A$, $a \precsim c$ and $b \precsim c$ for some $c \in A$.

It is possible to consider directed sets as a collective representation of a single term, because the only incompatibility between terms in a directed set is that one may halt while another diverges. In such a case we ignore the diverging computations, taking the most defined elements. It is then possible to define an ordering $A \precsim_S B$ which is an extension of $\precsim$ to directed sets of terms (look at the definition below carefully, however: it is not the obvious pointwise extension). This ordering is a generalization of $\precsim$ and is useful for characterizing the relation between sets of terms. $\cong_S$ is the equivalence induced by $\precsim_S$. Substitutivity of $\precsim_S$ may then be proved, and numerous applications of this result developed.

DEFINITION 4.2 (i) $A \precsim_{S,0} B$ iff A, B closed and directed and for all closed $R[\bullet]$ and $a \in A$, if $R[a]\downarrow$, then $R[b]\downarrow$ for some $b \in B$.

(ii) $A \precsim_S B$ iff $A^\sigma \precsim_{S,0} B^\sigma$ for all closing substitutions σ.

(iii) $A \cong_S B$ iff $A \precsim_S B$ and $B \precsim_S A$.

Here are some basic properties.

LEMMA 4.3 ($\precsim_S$ PROPERTIES) (i) $\{a\} \precsim_S \{b\}$ iff $a \precsim b$.

(ii) $A \precsim_S \{b\}$ iff $a \precsim b$ for all $a \in A$.

(iii) $a \in A$ implies $\{a\} \precsim_S A$.

(iv) $A \precsim_S B$ does *not* imply that for all $a \in A$ there exists $b \in B$ such that $a \precsim b$.

PROOF. (i)-(iii) are direct from the definitions. For a counterexample to (iv), see the Fixed Point Approximation Lemma (4.6).
QED.

The most important properties of $\precsim_S$ and $\cong_S$ are substitutivity and congruence properties; the proofs are very similar to the proofs of the same properties for $\precsim$ and $\cong$. The main difference is directnedness is needed in a context $C[\circ]$ with multiple holes because there can be a different member of the directed set in each hole, but by directedness there is a single biggest element which can replace all these elements and have at least as strong a termination behavior.

THEOREM 4.4 ($\precsim_S$ SUBSTITUTIVITY/CONGRUENCE) Four substitutivity properties over directed sets hold.

(i) $\precsim_{S,0}$ is substitutive, i.e. if $A \precsim_{S,0} B$ then $C[A] \precsim_{S,0} C[B]$ for closed $C[\circ]$.

†This restriction could be relaxed, but with an added bookkeeping requirement to ensure fresh variables are always available.

(ii) If $A \sqsubseteq_{\precsim S} B$, then $\lambda x.A \sqsubseteq_{\precsim S} \lambda x.B$.

(iii) $\sqsubseteq_{\precsim S}$ is substitutive.

(iv) $\cong_S$ is a congruence.

PROOF. (i) To show $\sqsubseteq_{\precsim S,0}$ substitutive, show for arbitrary $a \in A$ and closed $C[\circ]$ that if $C[a]\downarrow$, $C[b]\downarrow$ for some $b \in B$. The remainder of the proof proceeds by induction on computation length, paralleling the proof of the Closed Substitutivity Lemma (3.13). The only difference is the assumption we are reasoning under, in the other case $a \sqsubseteq_{\precsim 0} b$, and in this case $A \sqsubseteq_{\precsim S,0} B$. To give the difference in proofs, it suffices to consider the case $C'[\circ] = \circ$, where

$$R[a][\![a]\!] \mapsto_1 R[a][\![a']\!].$$

Applying the induction hypothesis to the right side treating a' as fixed, $R[b][\![a']\!]\downarrow$, so $R[b][\![a]\!]\downarrow$ as well. So, since $A \sqsubseteq_{\precsim S,0} B$, $R[b][\![b']\!]\downarrow$ for some b'. Then, by directedness of B, there exists b'' s.t. $b, b' \sqsubseteq_{\precsim 0} b''$, and $R[b''][\![b'']\!]\downarrow$, proving this case.

(ii), (iii), (iv) differ in an analogous manner with proofs of the Lambda Substitutivity Lemma (3.14), the $\sqsubseteq$ Substitutivity Theorem (3.15) and the $\sqsubseteq$ Congruence Theorem (3.12), respectively.
QED.

We show in section 4.2 that the lub of a $\sqsubseteq$-directed set is not always a useful property, because some chains have what we think of as their proper lub missing due to its uncomputability. This means operational least upper bounds may have a discontinuity going from the chain to the lub, and continuity of all functions fails (Theorem 4.13). Fortunately, there is an analogue here that is just as useful. The assertion $\{a\} \cong_S A$ captures the fact that a single computation has the same behaviors as a directed set of computations. This notion will be shown to succesfully take the place of lub. An elegant restatement of this is a is the *greatest* term that is a lower bound of the set A.

LEMMA 4.5 (GREATEST LOWER BOUND) $\{a\} \cong_S A$ iff $\{a\} \sqsubseteq_{\precsim S} A$ and for all a', if $\{a'\} \sqsubseteq_{\precsim S} A$, then $a' \sqsubseteq a$.

PROOF. $\Rightarrow$: $\{a\} \sqsubseteq_{\precsim S} A$ follows trivially. Supposing $\{a'\} \sqsubseteq_{\precsim S} A$, show $a' \sqsubseteq a$. Show for closed terms; the result for open terms is then direct. Suppose $R[\![a']\!]\downarrow$, show $R[\![a]\!]\downarrow$. By the definition of $\sqsubseteq_{\precsim S,0}$, $R[\![a'']\!]\downarrow$ for some $a'' \in A$; thus, since $A \sqsubseteq_{\precsim S,0} \{a\}$, $R[\![a]\!]\downarrow$.
$\Leftarrow$: Show $A \sqsubseteq_{\precsim S} \{a\}$ (the other direction of $\cong_S$ is given) by showing $a' \sqsubseteq a$ for arbitrary $a' \in A$. Since $\{a'\} \sqsubseteq_{\precsim S} A$, this follows directly by assumption.
QED.

4.1 Least fixed point principle

For this section, take f to be a functional $\lambda x.\lambda z.b$. We wish to reason about fixed points of such functionals by considering their finite approximations. The directed set we study is

$$\{\bot_\lambda, f(\bot_\lambda), f(f(\bot_\lambda)), \ldots, f^k(\bot_\lambda), \ldots\},$$

which is shown $\cong_S$ to $\{Y_v(f)\}$. Intuitively, this means in any particular program context, some finite-depth recursion stack will suffice to compute a recursive function properly.

LEMMA 4.6 (FIXED POINT APPROXIMATION) $\{Y_v(f)\} \cong_S \{f^k \mid k \in \mathrm{N}\}$

PROOF. Take f to be closed, for from this case the result follows for arbitrary f. The right-to-left direction is direct by computing; for the other direction, note $Y_v(f) \cong f'(f')$ where $f' \stackrel{\text{def}}{=} \lambda x.\lambda z.f(x(x))(z)$, so it suffices to show $\{f'(f')\} \sqsubseteq_{\precsim S} \{f^k \mid k \in \mathrm{N}\}$. Expanding definitions, the goal becomes

$R[\![f'(f')]\!]\downarrow$ implies $R[\![f^k]\!]\downarrow$ for some k.

The rest of the proof parallels previous proofs such as the Closed Substituvity Lemma (3.13): generalize and induct on computation length. The only interesting case, where $f'(f')$ is touched, is

$$R[f'(f')][\![f'(f')]\!] \mapsto_1 R[f'(f')][\![\lambda z.f(f'(f'))(z)]\!],$$

with the goal to show $R[f^k][\![f^k]\!]\downarrow$. By induction hypothesis, for some k', $R[f^{k'}][\![\lambda z.f(f^{k'})(z)]\!]\downarrow$, so since $f^{k'} \sqsubseteq_{\lesssim} f^{k'+1}$, and $\lambda z.f(f^{k'})(z) \cong f^{k'+1}$ by η, $R[f^{k'+1}][\![f^{k'+1}]\!]\downarrow$, and letting k be $k'+1$, the goal is proven.
QED.

It is now possible to prove a call-by-value least fixed point theorem.

THEOREM 4.7 (LEAST FIXED POINT) $Y_v(f) \cong f(Y_v(f))$ and for all $\lambda x.a$ such that $\lambda x.a \cong f(\lambda x.a)$, $Y_v(f) \sqsubseteq_{\lesssim} \lambda x.a$.

PROOF. The first half follows by computing; for the second half, suppose for arbitrary $\lambda x.a$ that $\lambda x.a \cong f(\lambda x.a)$. $\{f^k | k \in \mathrm{N}\} \sqsubseteq_{\lesssim S} \{\lambda x.a\}$ follows by showing $f^k \sqsubseteq_{\lesssim} \lambda x.a$ by direct induction on k. By the Fixed Point Approximation Lemma (4.6), $\{f^k | k \in \mathrm{N}\} \cong_S \{Y_v(f)\}$. Thus, $\{Y_v(f)\} \sqsubseteq_{\lesssim S} \{\lambda x.a\}$, so $Y_v(f) \sqsubseteq_{\lesssim} \lambda x.a$ by the $\sqsubseteq_{\lesssim S}$ Properties Lemma (4.3).
QED.

4.2 Least upper bounds

Least upper bounds are important phenomena in domains, but since operational theories have fewer points due to absence of uncomputable functions, lubs are not nearly as interesting. In particular, continuity of all functions fails.

Before defining least upper bounds, we note that not all directed sets of terms have upper bounds, so they also will not have least upper bounds. Let $\overline{K}$ be the non-r.e. set

$$\overline{K} \stackrel{\text{def}}{=} \{n \mid n \in \mathrm{N} \text{ and Turing machine } \phi_n \text{ diverges on blank tape }\}.$$

To simplify notation, define metafunctions

$$\rho(n) = \begin{cases} 1 & \text{if } n \in \overline{K} \\ 0 & \text{otherwise} \end{cases}$$
$$\overline{\rho}(n) = 1 - \rho(n)$$

The directed set

$$D_0 = \{f_k \mid f_k(n) \cong \rho(n) \text{ for } n < k, \\ f_k(n) \cong \bot \text{ o.w. }\},$$

where the functions f_k are definable since they have finitely many non-$\bot$ values, cannot have an upper bound, because this would be a function solving the halting problem. This example illustrates the root of the problem with lubs: in a domain, the presence of uncomputable functions gives nice closure conditions we never can obtain in a purely operational setting. We thus must define lub as a relation, not an operation.

DEFINITION 4.8 $\bigsqcup(A, a)$ ("a is the least upper bound of A") iff $A \sqsubseteq_{\lesssim S} \{a\}$ and for all a', if $A \sqsubseteq_{\lesssim S} \{a'\}$, then $a \sqsubseteq_{\lesssim} a'$.

Other sets have upper bounds but no least upper bound. The set

$$D_1 = \{d_k \mid d_k(n) \cong 1 \text{ if } n < k \text{ and } \rho(n) = 1, \\ d_k(n) \cong \bot \text{ o.w. } \}$$

has the upper bound $d = \lambda x.1$, but for instance assuming $\rho(k) = 0$,

$$\lambda x.\text{ if } x = k \text{ then } \bot \text{ else } 1$$

is a smaller upper bound. It is easy to show by a computability argument that no least upper bound of D_1 exists.

The continuity property of functions in domain theory is defined in an environment where lubs always exist, and takes the form $\bigsqcup_{a \in A} f(a) = f(\bigsqcup A)$, with $\bigsqcup A$ an operation returning the lub of A. Without the presence of such an operation, continuity of f is phrased as follows.

DEFINITION 4.9 f is *continuous* iff given a directed set A and a term a with $\bigsqcup(A, a)$, $\bigsqcup(f(A), f(a))$.

The failure of continuity is shown by counterexample. For some function f, there is a directed set L and upper bound l such that $\bigsqcup(L, l)$, but $\bigsqcup(f(L), f(l))$ fails. The counterexample is highly language-dependent, but this should not be taken to mean non-continuity is a product of this language; it is really a product of the "lack of points" in an operational semantics, but showing this is difficult and requires using much specific knowledge of the language.

Before giving f, l and L, let us motivate their construction. As already observed, the key problem is the missing points corresponding to uncomputable functions. We demonstrated an upper bound d of the set D_1 that was not least. We modify that example by defining l and L such that it is impossible to define any smaller upper bounds of L, making l a least upper bound. However, l is least for artificial reasons, i.e. because the "real" least upper bound is uncomputable. We can then expose this artificiality by applying a function f to L and l that makes the real least upper bound of $f(L)$ computable again, demonstrating a discontinuity in f's behavior.

DEFINITION 4.10 Terms L, l, and f are defined as follows:

$$L = \{l_k | k \in \mathrm{N}\}, \text{ where} \\ \quad l_k = \lambda x_0, x_1, x_2.\mathit{if\_zero}(a_k(x_2); 1; 1), \\ \quad\quad a_0 = \bot_\lambda, \\ \quad\quad a_{k+1} = a_k\{\bot_\lambda := \lambda z.x_{\rho(k+1)}(\bot_\lambda)(z)\}; \\ l = \lambda x_0, x_1, x_2.1; \text{ and} \\ f = \lambda x.x(\lambda y.y).$$

The l_k encode an undecidable problem in an unusual way, by alternating successive self-application of two functions x_0 and x_1 according to the dictates of some non-r.e. set. L is trivially directed, because each successive term replaces $\bot_\lambda$ with some non-$\bot_\lambda$ function.

LEMMA 4.11 $\bigsqcup(L, l)$.

PROOF. $L \sqsubseteq_S \{l\}$ trivially. Suppose $L \sqsubseteq_S \{a\}$ for some a, show $l \sqsubseteq a$. We in fact show something stronger, that $l \cong a$. Assume $l \ncong a$, derive a contradiction by showing a must define a semi-decision procedure for the non-r.e. problem $\overline{K}$.

The assumption $L \sqsubseteq_S \{a\}$ means $l_k(t_0)(t_1)(t_2)\downarrow$ implies $a(t_0)(t_1)(t_2)\downarrow$ for any terms t_0, t_1, t_2. We show that it suffices to look at only a small subset of all possible t_0, t_1, t_2 to deduce a must semi-decide $\overline{K}$. The triples t_0, t_1, t_2 we are interested in are what we call the *core triples*. They are made up of four k-indexed families of triples, now defined.

DEFINITION 4.12 The set of core triples, $\mathcal{T}$, is $\mathcal{T}_a \cup \mathcal{T}_b \cup \mathcal{T}_c \cup \mathcal{T}_d$,

$$\begin{aligned}
\mathcal{T}_a &\stackrel{\text{def}}{=} \{t_{\text{num}}[\bot]\llbracket k\rrbracket, t_{\text{num}}[k]\llbracket k\rrbracket, n \mid n \in \text{N}, k \in \text{N}\},\\
\mathcal{T}_b &\stackrel{\text{def}}{=} \{t_{\text{num}}[k]\llbracket k\rrbracket, t_{\text{num}}[\bot]\llbracket k\rrbracket, n \mid n \in \text{N}, k \in \text{N}\},\\
\mathcal{T}_c &\stackrel{\text{def}}{=} \{t_{\text{pr}}[\bot]\llbracket k\rrbracket, t_{\text{pr}}[k]\llbracket k\rrbracket, \langle n, 0\rangle \mid n \in \text{N}, k \in \text{N}\},\\
\mathcal{T}_d &\stackrel{\text{def}}{=} \{t_{\text{pr}}[k]\llbracket k\rrbracket, t_{\text{pr}}[\bot]\llbracket k\rrbracket, \langle n, 0\rangle \mid n \in \text{N}, k \in \text{N}\},\\
&t_{\text{num}}[\circ]\llbracket\bullet\rrbracket \stackrel{\text{def}}{=} \lambda y.\lambda x.\ \mathit{if}\ x = 0\ \mathit{then}\ (\circ)\ \mathit{else}\\
&\qquad (\mathit{if}\ y(\mathit{pred}(x)) = (\bullet)\ \mathit{then}\ (\bullet)\ \mathit{else}\ \bot)^{\ddagger}\\
&t_{\text{pr}}[\circ]\llbracket\bullet\rrbracket \stackrel{\text{def}}{=} \lambda y.\lambda x.\ \mathit{if}\ \pi_1(x) = 0\ \mathit{then}\ (\circ)\ \mathit{else}\\
&\qquad (\mathit{if}\ y(\langle \mathit{pred}(\pi_1(x)), 0\rangle) = (\bullet)\ \mathit{then}\ (\bullet)\ \mathit{else}\ \bot)
\end{aligned}$$

$\mathcal{T}_a$ and $\mathcal{T}_b$ have the third component of the triple numeric, and the first two components correspondingly expect numerical arguments for x; $\mathcal{T}_b$ and $\mathcal{T}_c$ have the third component of the triple $\langle n, 0\rangle$, a number hiding inside a pair, and the first two components expect this sort of argument. Other than this small difference, $\mathcal{T}_a$ is $\mathcal{T}_c$, and $\mathcal{T}_b$ is $\mathcal{T}_d$. Another thing to note about the structure of the triples is the first two components return at most the value k when applied, and furthermore, when y is applied, a result of k is expected, or else will diverge. Both of these features are used to constrain the behaviors a can have. We show that a either doesn't evaluate any of the arguments, in which case it must be equal to l, or the minute it touches any of the arguments it is necessarily forced to behave like a semi-decision procedure for $\overline{K}$.

To streamline notation, $\lceil n\rceil$ will be used as context-sensitive notation to either indicate n or $\langle n, 0\rangle$, depending on whether it is the component of a tuple in $\mathcal{T}_{a/b}$, or $\mathcal{T}_{c/d}$, respectively. Thus, for $t_0, t_1, \lceil n\rceil \in \mathcal{T}_d$, $\lceil n\rceil$ is $\langle n, 0\rangle$. Similarly, $\lceil \mathit{pred}(a)\rceil$ is either $\mathit{pred}(a)$ or $\langle \mathit{pred}(\pi_1(a)), 0\rangle$ depending on context. $\lceil \mathit{pred}^k(a)\rceil$ denotes $\overbrace{\lceil \mathit{pred}(\ldots \lceil \mathit{pred}}^{k}(a)\rceil \ldots)\rceil$. Given a triple t_0, t_1, t_2, u_0 and u_1 are defined such that $t_0 = \lambda y.\lambda x.u_0$, and $t_1 = \lambda y.\lambda x.u_1$.

For $t_0, t_1, \lceil n\rceil \in \mathcal{T}_{a/c}$, observe $l_n(t_0)(t_1)(\lceil n\rceil)\downarrow$ iff $n \in \overline{K}$, so L collectively gives a semi-decidable characterization of $\overline{K}$. Similarly, $\mathcal{T}_{b/d}$ gives a co-semi-decidable characterization of $\overline{K}$. One corollary of this that will be used below is if $l_i(t_0)(t_1)(\lceil n\rceil)\downarrow$ for some $t_0, t_1, \lceil n\rceil \in \mathcal{T}_{a/c}$, $l_i(t_0')(t_1')(\lceil n\rceil)\uparrow$ for any $t_0', t_1' \in \mathcal{T}_{b/d}$, and conversely.

With all the definitions in place, we assert if $a \not\cong l$, $a(t_0)(t_1)(\lceil n\rceil) \mapsto R\llbracket u_{\rho(n)}\{y := a'\}\{x := \lceil \mathit{pred}^n(\lceil n\rceil)\rceil\}\rrbracket$. This is proved by showing a can only choose one of the many possibilities it could otherwise take in evaluating. We will start the argument by showing that of t_0, t_1 and $\lceil n\rceil$, it must be $t_{\rho(1)}$ that is touched first. Suppose none of the arguments are touched. Then, a must be l, contradicting our assumption. Suppose that $\lceil n\rceil$ is touched first. Since this is the first touch, up to this point the computation proceeds uniformly. But, since $\lceil n\rceil$ may either be a number or a pair, any touch must treat it as either one or the other but not both, causing divergence for half of the core triples, a contradiction. For the case $t_{\overline{\rho}(1)}$ was touched first, if $t_{\overline{\rho}(1)}$ were $\bot_\lambda$, there still would be cases for which some l_k would converge, but this computation would diverge, a contradiction. Thus, by a process of elimination we have shown that $a(t_0)(t_1)(\lceil n\rceil) \mapsto R\llbracket t_{\rho(1)}(v)\rrbracket$ for some v. The remainder of the argument proceeds in a similar fashion of exhaustively eliminating all alternative computation paths and depends critically on the definition of the core triples to rule out paths.

$l_n(t_0)(t_1)(\lceil n\rceil)$ also forces evaluation of a similar term $u_{\rho(n)}\{y := \bot\}\{x := \lceil \mathit{pred}^n(\lceil n\rceil)\rceil\}$, and if this value is non-$\bot$, the whole computation converges. Thus, $l_n(t_0)(t_1)(\lceil n\rceil)$ must converge exactly when $a(t_0)(t_1)(\lceil n\rceil)$ does.

[‡] *if a = b then c else d* is defined as *if_zero*(*subtract*(*a*)(*b*); *if_zero*(*subtract*(*b*)(*a*); *c*; *d*); *d*), and *subtract* is $Y_v(\lambda y.\lambda x.\lambda x'.\mathit{if\_zero}(x; 0; \mathit{if\_zero}(x'; x; y(\mathit{pred}(x))(\mathit{pred}(x')))))$.

However, for $t_0, t_1, n \in \mathcal{T}_a$, $l_n(t_0)(t_1)(\lceil n \rceil)\downarrow$ iff $a(t_0)(t_1)(\lceil n \rceil)\downarrow$ iff $n \in \overline{K}$, making $a(t_0)(t_1)$ a semi-decision procedure for $\overline{K}$, contradicting the fact that $\overline{K}$ is non-r.e.

Therefore, $L \sqsubseteq_S \{a\}$ and $a \not\cong l$ is impossible, meaning $l \cong a$, i.e. l is equivalent to all upper bounds, so l is also least.
QED.

THEOREM 4.13 $\bigsqcup(f(L), f(l))$ fails, and since $\bigsqcup(L, l)$, f is not continuous.

PROOF. Given L, l, and f as defined above, we have $\bigsqcup(L, l)$ by the previous lemma. Consider $f(L)$: it is equivalent, by computing, to

$$\{\lambda x_1, x_2.\mathit{if\_zero}(x_1^{j_k}(x_2); 1; 1) | k \in \mathrm{N}\},$$

where j_k is the cardinality of the set $\{n \mid n \in \overline{K} \text{ and } n \leq k\}$. $f(L)$ thus removes all x_0 terms from the infinite chain. Since there are infinitely many members of $\overline{K}$, the chain now has upper bound $Y(x_1)$. Thus, $f(L) \sqsubseteq_S \{\lambda x_1, x_2.\mathit{if\_zero}(Y(x_1)(x_2); 1; 1)\}$, and

$$f(l) \cong \lambda x_1, x_2.1 \not\sqsubseteq \lambda x_1, x_2.\mathit{if\_zero}(Y(x_1)(x_2); 1; 1),$$

meaning $f(l)$ is not least.
QED.

Since $\cong_S$ is a congruence (Theorem 4.4), $\bigsqcup(A, a)$ cannot be the same as $A \cong_S \{a\}$. In fact, it is a stronger relation.

LEMMA 4.14 $A \cong_S \{a\} \Rightarrow \bigsqcup(A, a)$, and the converse fails.

PROOF. It suffices to show for the case A, a closed. $A \sqsubseteq_S \{a\}$ is trivial; suppose $A \sqsubseteq_S \{a'\}$, show $a \sqsubseteq a'$. Expanding the definition of $\sqsubseteq$, assume $R[\![a]\!]\downarrow$, show $R[\![a']\!]\downarrow$. $\{a\} \sqsubseteq_S A$, so $R[\![a'']\!]\downarrow$ for some $a'' \in A$; thus, by assumption, $R[\![a']\!]\downarrow$. Suppose the converse held; then, supposing $\bigsqcup(A, a)$, we have $A \cong_S \{a\}$ and then have $f(A) \cong_S \{f(a)\}$ and applying the first case of this lemma, $\bigsqcup(f(A), f(a))$, so we have just proved continuity, contradicting Theorem 4.13.
QED.

The moral of this story is not the weakness of operational reasoning, it is only the weakness of the concept of lub for operational reasoning. Where you wished you could say $\bigsqcup(A, a)$, say $A \cong_S \{a\}$ instead. This will be born out in the ideal completion construction in section 5, where the standard definition of ideal completion is modified exactly as just described to construct a cpo.

4.3 Fixed-point induction

One of the most useful induction principles is the Scott fixed-point induction principle ([dS69]; see also [Man74]). The justification of fixed-point induction necessitates functions be continuous in a domain. All that is needed to justify fixed-point induction here is the Fixed Point Approximation Lemma (4.6) and the $\sqsubseteq_S$ Substitutivity Theorem (4.4).

THEOREM 4.15 (ATOMIC FIXED POINT INDUCTION) For $f = \lambda x.\lambda z.b$, if for all k, $C[f^k] \sqsubseteq C'[f^k]$, then $C[Y_v(f)] \sqsubseteq C'[Y_v(f)]$.

PROOF. By the $\sqsubseteq_S$ Substitutivity Theorem (4.4) and Fixed Point Approximation Lemma (4.6), $\{C[f^k] \mid k \in \mathrm{N}\} \cong_S \{C[Y_v(f)]\}$ and $\{C'[f^k] \mid k \in \mathrm{N}\} \cong_S \{C'[Y_v(f)]\}$. Then, using the fact $\{C[f^k] \mid k \in \mathrm{N}\} \sqsubseteq_S \{C'[f^k] \mid k \in \mathrm{N}\}$ (by definition of $\sqsubseteq_S$) and the above equivalences, the result is immediate.
QED.

It is a simple matter to extend this theorem to logical formulas in which statements $C[Y_v(f)] \sqsubseteq_\approx C'[Y_v(f)]$ occur, although only certain *admissible* formulae admit to fixed-point induction; see [Pau87, Iga72]. One alternative for a programming logic the above proof suggests is to axiomatize $\cong_S$ in a theory, and then derive instances of fixed-point induction as necessary; such a logic is defined in [Smi92b].

4.4 An Operational Theory of L

Now that the mathematical development is complete, we may collect together the basic results which form the core of a purely operational theory of L.

(i) $\sqsubseteq_\approx/\cong$ Properties Lemma (3.11).

(ii) $\sqsubseteq_\approx$ Substitutivity Theorem (3.15).

(iii) $\sqsubseteq_\approx$ Extensionality Lemma (3.17).

(iv) $\sqsubseteq_{\approx S}$ Properties Lemma (4.3).

(v) $\sqsubseteq_{\approx S}$ Substitutivity Theorem (4.4).

(vi) Fixed Point Approximation Lemma (4.6).

5 Constructing a cpo

In this section we show how a computation system for which the basic results have been proven may be extended by a minor variation on the standard ideal completion to give a fully abstract cpo model of the language. We carry out the construction for the language L of the previous section, but it only needs proofs of certain of the basic results, so is in fact quite general.

We construct a completion embedding of $\sqsubseteq_\approx$ into a cpo, using a minor variation on the standard ideal completion construction, following the development of [Sto88]. It should be emphasized there is no reason to carry out the completion of $\sqsubseteq_\approx$ for purposes of giving programs meaning; the main purpose of this argument is to show how close a rich operational theory can be to a good cpo model.

The elements of the cpo are closed sets of terms, which are defined as follows.

Definition 5.1 The closure $cl(A)$ of a set of terms A is the following least fixed-point

$$\begin{array}{l} c \in cl(A) \text{ iff } c \in A \text{ or } c \sqsubseteq_\approx b \text{ and } b \in cl(A), \text{ or} \\ \quad D \text{ directed and } D \subseteq cl(C) \text{ and } D \cong_S \{c\}. \end{array}$$

These sets are thus closed downward, and closed under $\cong_S$-equivalent terms (replacing the standard closure under lubs). Such sets will be called *closed sets*. The cpo is inductively constructed from closed sets.

Lemma 5.2 (Closure Properties) Closures have the following properties.

(i) cl is monotonic, i.e. $A \subseteq B$ implies $cl(A) \subseteq cl(B)$.

(ii) $cl(cl(A)) = cl(A)$.

(iii) $cl(\bigcup \mathrm{X}) = cl(\bigcup_{X \in \mathrm{X}} cl(X))$, where each $X \in \mathrm{X}$ is a set of terms.

(iv) $cl(\{a\}) = \{a' | a' \sqsubseteq_\approx a\}$

PROOF. Straightforward.

DEFINITION 5.3 (CPO MODEL) (i) C is the least set such that

$$cl(\{a\}) \in \mathrm{C}, \forall a \in \mathrm{E}$$
$$\text{if } \mathrm{X} \subseteq \mathrm{C}, \text{ then } cl(\bigcup \mathrm{X}) \in \mathrm{C}$$

(ii) $\sqsubseteq \stackrel{\text{def}}{=} \subseteq$, $\bigsqcup \mathrm{X} \stackrel{\text{def}}{=} cl(\bigcup \mathrm{X})$, and $[\![a]\!] = cl(\{a\})$.

LEMMA 5.4 $\subseteq$ is a cpo on C.

Function application on closed sets is defined as the closure of application applied pointwise. The resulting closed set must be shown to be in the cpo.

DEFINITION 5.5 $F(A) = cl(\{f(a) \mid f \in F, a \in A\})$, where F and A are closed sets.

LEMMA 5.6 For all $F, A \in \mathrm{C}$, $F(A) \in \mathrm{C}$.

PROOF. $F(A)$ is $cl(\{f(a) \mid f \in F, a \in A\})$, which is equal to $cl(\bigcup_{f(a) \in F(A)} cl(\{f(a)\}))$ by the Closure Properties Lemma (5.2) (iii) (letting $\mathcal{X}$ be $\{\{f(a)\} | f \in F, a \in A\}$), which is in C by the construction of C.
QED.

LEMMA 5.7 $[\![f(a)]\!] = ([\![f]\!])([\![a]\!])$.

Similar operations can be defined and compositional meaning given for the other constructors of the language.

Full abstraction is direct.

LEMMA 5.8 ($\lesssim$-FULL ABSTRACTION) $a \lesssim b$ iff $[\![a]\!] \sqsubseteq [\![b]\!]$.
PROOF. $[\![a]\!] \sqsubseteq [\![b]\!]$ iff $cl(\{a\}) \subseteq cl(\{b\})$ iff $a \in cl(\{b\})$ iff $a \lesssim b$.
QED.

Next we show the cpo inherits monotonicity and continuity properties from substitutivity properties of $\lesssim$ and $\lesssim_S$, respectively.

LEMMA 5.9 (MONOTONICITY) All functions in the cpo C are monotonic.
PROOF. Suppose $A \sqsubseteq A'$, show $F(A) \sqsubseteq F(A')$: it suffices to show

$$\{f(a) \mid f \in F, a \in A\} \subseteq \{f(a') \mid f \in F, a' \in A'\},$$

because the closure operation is itself monotone (Lemma 5.2). Since $A \subseteq A'$, this result is trivial.
QED.

LEMMA 5.10 (LEAST FIXED POINT) Letting D_f be $\{[\![f^k]\!] | k \in \mathrm{N}\}$, $\bigsqcup D_f = [\![Y(f)]\!]$.
PROOF. Closed sets are downward-closed, so the $\subseteq$ direction is trivial. For the $\supseteq$ direction, expanding definitions and using Lemma 5.2 (iii), $\bigsqcup D_f = cl(\bigcup_{k \in \mathrm{N}} cl(f^k))$. So all to be shown is $Y(f) \in cl(\bigcup_{k \in \mathrm{N}} cl(f^k))$ by lemma 5.2 (iv). $\{f^k | k \in \mathrm{N}\}$ is a subset of D_f, so from the Fixed Point Approximation Lemma (4.6) $\{f^k | k \in \mathrm{N}\} \cong_S \{Y(f)\}$, and by the definition of closure the proof is complete.
QED.

Continuity of all functions is somewhat harder to show, requiring an induction on the definition of cl. All that is needed is the substitutivity of $\lesssim$ and $\lesssim_S$. This brings the picture full circle, showing substitutivity of $\lesssim_S$ justifies continuity in the cpo.

LEMMA 5.11 (CONTINUITY) all functions $F \in \mathrm{C}$ are continuous: $F(\bigsqcup \mathrm{X}) = \bigsqcup_{X \in \mathrm{X}} F(X)$.

PROOF. Expanding definitions, we want to show

$$cl(\{f(b) \mid f \in F, b \in cl(\bigcup \mathrm{X})\}) = cl(\bigcup_{X \in \mathrm{X}} cl(\{f(b) \mid f \in F, b \in X\})).$$

By Lemma 5.2, the inner closure may be removed from the right-hand-side, replacing it with $cl(\bigcup_{X \in \mathrm{X}} \{f(b) \mid f \in F, b \in X\})$. We now prove the equivalence in each direction.

$\supseteq$: this direction is easy; by monotonicity of cl, it suffices to show this fact before taking the closure of both sides. Suppose $a \in \{f(b) \mid f \in F, b \in X\}$ for some X; this means that $a \in \{f(b) \mid f \in F, b \in cl(\bigcup \mathrm{X})\}$.

$\subseteq$: This direction is more difficult, requiring an induction on the definition of cl. Since $cl(cl(A)) = cl(A)$, we can remove the outer closure from the left-hand equality. So, we suppose $f(b) \in \{f(a) \mid f \in F, a \in A\}$ where $b \in cl(\bigcup \mathrm{X})$, and show $f(b) \in D \stackrel{\text{def}}{=} cl(\bigcup_{X \in \mathrm{X}} \{f(b) \mid f \in F, b \in X\})$ by induction on the definition of $cl(\bigcup \mathrm{X})$, definition 5.1. We make the inductive assumption

$$\text{if } c \in cl(\bigcup \mathrm{X}) \text{ then } f(c) \in D$$

for smaller c. Suppose $b \in cl(\bigcup \mathrm{X})$, show $f(b) \in D$. There are three cases in the definition of closure:

CASE $b \in \bigcup \mathrm{X}$: Straightforward.

CASE $D \subseteq cl(\bigcup \mathrm{X}), D \cong_S \{b\}$: By the induction hypothesis, if $d \in D$, $d \in cl(\bigcup \mathrm{X})$, so $f(d) \in D$. By the congruence of $\cong_S$, following directly from substitutivity of $\precsim_S$, $f(D) \cong_S \{f(b)\}$; since D is a closure, this means $f(b) \in D$.

CASE $b \precsim b', b' \in cl(\bigcup \mathrm{X})$: By substitutivity of $\precsim$, $f(b) \precsim f(b')$; $f(b') \in D$, so by the properties of closure, $f(b) \in D$.

QED.

Summarizing these results, we have constructed a compositional cpo model for L for which a least fixed point property holds and all functions are continuous.

6 Concluding remarks

This paper treats directed sets of computations at a purely operational level, proving substitutivity of an ordering $\precsim_S$ on directed sets of terms which then leads to direct proofs of least fixed point and fixed point induction theorems, two theorems that have tradionally required domain theory to prove (with the exception of Talcott's direct proof of the least fixed point principle [Tal89]). The relations $\precsim$ and $\precsim_S$ and associated properties can then be used to construct fully abstract cpos, showing a rich operational theory has expressive power comparable to domain theory. Languages for which domain construction (fully abstract domain construction in particular) is intractable or infeasible thus have another route by which full and faithful semantics may be developed.

The methods used to develop the operational theory herein are simple, and we plan to show in future papers applicability to a wide range of languages, including typed, nondeterministic, and imperative languages. Concrete evidence of feasibility of this task is the work of Talcott and Mason [Tal89, MT91], where, working with languages wth first-class continuations and memories, they use the same general techniques to prove a subset of the basic results obtained here.

Acknowledgements

I owe many thanks to Carolyn Talcott for important suggestions and comments. Thanks also to Albert Meyer for useful comments.

References

[Abr90] S. Abramsky. The lazy lambda calculus. In *Research Topics in Functional Programming*, pages 65–116. Addison-Wesley, 1990.

[Blo90] B. Bloom. Can LCF be topped? *Information and Computation*, 87:264–301, 1990.

[dS69] J. W. deBakker and D. Scott. A theory of programs. unpublished notes, 1969.

[EHdR92] L. Egidi, F. Honsell, and S. R. della Rocca. Operational, denotational, and logical descriptions: a case study. *Fundamenta Informaticae*, 1992. (to appear).

[FFK87] M. Felleisen, D. Friedman, and E. Kohlbecker. A syntactic theory of sequential control. *Theoretical Computer Science*, 52:205–237, 1987.

[GMW79] M. J. Gordon, R. Milner, and C. P. Wadsworth. *Edinburgh LCF: A Mechanized Logic of Computation*, volume 78 of *Lecture notes in Computer Science*. Springer-Verlag, 1979.

[How89] D. J. Howe. Equality in lazy computation systems. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science*, pages 198–203. IEEE, 1989.

[Iga72] S. Igarashi. Admissibility of fixed-point induction in first-order logic of typed theories. Technical Report Stan-CS-72-287, Stanford University Computer Science Department, 1972.

[Man74] Z. Manna. *Mathematical Theory of Computation*. McGraw-Hill, 1974.

[Mil77] R. Milner. Fully abstract models of typed λ-calculi. *Theoretical Computer Science*, 4:1–22, 1977.

[MPS84] D. B. MacQueen, G. Plotkin, and R. Sethi. An ideal model of types. In *Conference Record of the Eleventh Annual ACM Symposium on Principles of Programming Languages*, 1984.

[MT91] I. A. Mason and C. L. Talcott. Equivalence in functional languages with effects. *Journal of Functional Programming*, 1:287–327, 1991.

[Pau87] L. C. Paulson. *Logic and Computation: Interactive Proof with Cambridge LCF*. Cambridge, 1987.

[Plo75] G. Plotkin. Call-by-name, call-by-value, and the λ-calculus. *Theoretical Computer Science*, pages 125–159, 1975.

[Plo77] G. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5:223–255, 1977.

[Sco76] D. Scott. Data types as lattices. *SIAM J. Computing*, 5:522–587, 1976.

[Smi92a] 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.

[Smi92b] S. F. Smith. Partial computations in constructive type theory. Submitted to *Journal of Logic and Computation*, 1992.

[Sto88] A. Stoughton. *Fully abstract models of programming languages.* Research notes in theoretical computer science. Pitman, 1988.

[Tal89] C. L. Talcott. Programming and proving with function and control abstractions. Technical Report STAN-CS-89-1288, Stanford University, Stanford, CA 94305, 1989.