A Powerdoman Semantics for Indeterminism

David S. Wise

Computer Science Department
Indiana University
Bloomington, IN 47405

Computer Science Department
Oregon State University
Corvallis, OR 97331

A Powerdoman Semantics for Indeterminism

by

David S. Wise

Indiana University, Computer Science Department, Bloomington, IN 47405

Oregon State University, Computer Science Department, Corvallis, OR 97331

TECHNICAL REPORT NO. 142

A POWERDOMAIN SEMANTICS FOR INDETERMINISM

by

David S. Wise

January, 1984 (revised)

Research reported herein was supported by the National Science Foundation under a grant numbered MCS82-03978.

Abstract:

A denotational semantics is presented for a language that includes multiple-valued functions (essentially Lisp S-expressions), which map from ground values into the power domain of ground values. The domain equations are reflexive, and fixed points of all functions are defined. Thus, it is possible to specify an operating system as a function whose codomain is a set of possible behaviors of the system, only one of which is realized under an operational semantics. Such a system can be specified using "pure" applicative programming (recursion equations without side effects) over primitive functions like *amb*, *frons*, *arbiter*, or *arbit*, all of which are formally defined.

Tempered by 'environmental transparency,' we consider a power domain semantics in which the power domain may only occur within the codomain in the equation that defines function space. The problem addressed is how to define the analog of 'fixed point' for a function from the ground domain to thathat power domain, in a semantics that uses natural extension as the axiom for function application. Denotational and equivalent operational semantics are presented for the Smyth-upside-down power domain; a similar denotational semantics is presented for the Plotkin power domain (Egli-Milner order), along with a likely operational semantics.

CR Categories and Subject Descriptors: F3.2 **[Logics and Meanings of Programs]**: Semantics of Programming Languages—*Denotational semantics*; D.4.1 **[Operating Systems]**: Process Management—*Multiprocessing/multiprogramming*; D3.1 **[Programming Languages]**: Formal Definitions and Theory—*Semantics*; D3.2: Language Classifications—*Applicative languages*.

General Terms: Theory, Languages.

Additional Key Words and Phrases: power domain, indeterminism, nondeterminism.

1. Introduction and Notation

Indeterminate programs are those which can return any of a set of many possible results. The archetypical example is an operating system, which takes one or several streams as input and returns one or several streams of output; typically the streams are routed from/to peripherals such as interactive consoles of users who are contending for system resources. The reason that a set of results is acceptable is because users expect any of a set of different behaviors from identical contention. For instance, if A and B both attempt to reserve the last seat on the afternoon plane we allow either A or B to succeed (but not both).

Applicative or functional programming is the style of specifying programs as recursive equations, with function application as the only control structure, and with parameter/argument association as the only binding. Under this discipline, where time or sequentiality of execution is often elided by relaxing input/output synchronization or by lazy evaluation [7], the two input streams that represent the aforementioned reservation attempts are identical. Thus, there can be no single answer. We expect A to succeed if he tries an hour before B, and B to succeed if he reserves well before A. The set of legal system responses must, therefore, include both the pair of streams accepting A and refusing B and the pair that seats B and turns A away. Moreover, if the reservations are made concurrently, no one will be able to predict which pair the correct reservation system will choose. Cases like this, where even the programmer cannot predict machine behavior, are the object of this paper.

Denotational semantics [18, 21] is the mathematics of inferring meanings from a continuous function on an underlying 'domain' through the abstract device of Tarski's fixed points. No mechanical ("operational") model of a computing device, such as Curry's logic , Church's calculus, Turing's machine, or Markov's computation is necessary. Applicative programming is intellectually close to the endeavor of composing a continuous function, because the usual expression of a continuous function appears as a set of recursive equations. No *go-to*'s and no assignment statements appear in either style.

Programs composed under either discipline appear to be a product of the other simply because the syntax and style of expression is so nearly the same. To those used to more classic programming styles (e.g. FORTRAN) these fields are often perceived as one. The confusion should be encouraged because of the consistency between the denotational and expressive power that they offer. If, as its proponents believe, denotational semantics is the most precise way of imparting meaning to programs, and if, similarly, applicative programming is the best way to program for highly parallel architectures, then symbiosis between them will yield deep and precise understanding for highly parallel programs. Without hope for such understanding from any other approach, we must pursue their relationship before we build grandiose software tools for driving the plethora of parallel architectures now electronically possible.

1.1. Domains and Power domains

The reader is referred to Stoy [21] for a precise and readable introduction to this topic. Briefly, domains are the collections of values over which we compute. Here characterized as continuous, countably-based, complete partial orders (Stoy uses lattices, and other characterizations are available [18]), they include both elementary data and the functions, themselves, that programmers construct when they ply their trade. The fact that domains include functions on them—that they are *reflexive*—is a confusion both to computer scientists and to mathematicians. The mathematician is concerned about the cardinality of a set that includes functions *over* it, until he realizes that only the continuous functions are being considered. The computer scientist is confused by all those "extra" functions within the domain of values, forgetting that von Neumann architecture requires programs as data; denotational semantics merely abstracts representation away from programs, leaving behind pure functions.

A domain is ordered by the relation $\sqsubseteq$ which may be interpreted as $A \sqsubseteq B$ if the value B contains more information than A, or represents the result of more computing effort on the same problem than the result A. The "well-below" ordering, $A \sqsubset C$, holds when A and C may be separated by some finitely computable (isolated) element of the domain, B, between them: $A \sqsubseteq B \sqsubseteq C$. The minimum value $\perp$ in every domain contains the least information; it *results* from a computation that is elsewhere interpreted as "divergent". When one applies the theory to impart meaning on a recursive equation for a function, one understands that the intended function is the *least* (-defined with respect to $\sqsubseteq$) fixed point of that equation.

When denotational semantics is extended into the realm of operating systems, the role of a simple result datum is now occupied by a set of legal answers, any of which is acceptable. That is, the elements of the domain are the elements of the power set over an underlying domain. It is the purpose of this paper to explore how fixed points in that power domain might be inferred from recursive equations of any sort. Several domain orderings have been proposed for power domains, two of which are explicitly defined below. The so-called "Smyth-upside-down" or the "other half" of the Egli-Milner order is the one mainly used here.

1.2. Motivation

The goal of this paper is to explore a powerdomain construction in some detail in order to understand the relationship between operational indeterminism and the formal definition of semantics of recursive definitions. Insight into a formalism that supports the intuitive operational procedures is sought. The infered proof rule is subtle and best explained in terms of environments. The results are general, applying to Egli-Milner, Smyth-upside-down, and Smyth domains; they appear as a series of lemmas and theorems that support the formalism and tie it closely to the anticipated procedures for operational semantics (established for Smyth-upside-down).

1.3. Environmental Transparency

The most important concept to be developed is that a function from a ground domain into its powerdomain *dominates* some continuous functions on that ground domain, and that the analog of the fixed point of this function may be described as the powerdomain element defined by

the fixed points of these dominated functions. This idea is sufficiently general to transfer to other powerdomain definitions, and we shall see how it derives from the concept of *environmental transparency* below.

1.3.1. Transparency Considered Twice

"Environmental transparency" is a broad philosophical refinement of "referential transparency," a term attributed to Russell and Whitehead [22], but defined more precisely by Quine[17]. Quine's descriptive definition, perfectly suitable for deteministic computation, centers on the universal safety of so-called β-substitution of the λ-calculus.

Environmental transparency denies this universality, and replaces it with recognition of environment as an implicit object that interprets identifiers. An environment is a function which maps identifiers to unique, elementary (below dubbed *denoted*) values; the image of any single identifier under that map is its *binding in that environment*. Whenever one creates new bindings, one creates a new environment (a conceptually important and operationally expensive feat). Environmental transparency recognizes that many envrionments may "exist" at once and that a burden in computation, evaluation, or semantic interpretation is to recognize the appropriate environment to use for discharging an identifier.

1.3.2. Bindings to Denoted Values

Environmental transparency seems natural to practitioners of denotational semantics (where ρ is generally bound to environment [21]), and of lexically-scoped programming languages like Common LISP [19], where function-values must include their respective environments. Even these colleagues may not, however, subscribe to the elementary nature of the codomain of an environment [4], necessitated by indeterminism[1].

Quite an intuitive situation obtains when the binding of a parameter to an individual argument is considered. A parameter can only be bound to a single function (program) or to a single ground (underlying the power domain) value; meaning of a parameter in any binding environment must be unique. Under indeterminism, environmental transparency holds that environment creation distributes over choice, so that instead of binding X, say, to the *choice* of values indicated by the set {1,2}, we instead consider the choice of two environments, where X is bound to 1 in one and X is bound to 2 in the other. This leads to the use of "natural extension" of traditional application rules in the language axioms below.

Elements from power domains, or sets, of ground values never crawl through the environment into the evaluation of an expression. An identifier X is bound to *exactly* one value (never an element in the power domain), and X=X can never evaluate to **false**. That is, λx.(x=x) applied to the powerdomain element {1,2} cannot return **false** in its answer set. General β-substitution becomes impossible, because β-substitution of-expression-for-identifier here leads to evaluation of {1,2}={1,2}, one of whose constituent interpretations is (1=2) which evaluates **false**.

1.3.3. Power Domains over Ground Values

A major thesis of the present work is that it is wrong simply to build a power domain directly from any underlying reflexive domain. Rather, one should build the power domain from an underlying domain without functions, a domain of *ground* or simple data types, and adjoin to it the appropriate reflexive domain of functions.

At first it appears that this thesis thereby denies functions as "first class citizens", but, while still not first class, their restrictions are not so onorous. Functions, though explicitly excluded from indeterminism, may still be arguments of functions, results of functions, and even elements of data structures (Section 8 below); exclusion from these roles has classically been dubbed "second" class, though I suggest that "third" is better here, definitely beneath the stature of

---

[1]Indeterminism has not yet been sufficiently well understood, but I believe that we are close to a satisfactory formalism that, as will be shown, denies select roles for power domains: as objects of bindings (codomains of environments) and over objects of indeterminsitic choice (over indeterminate functions or other power domains).

functions. A new intermediate "second class" role is rendered the equivalent of "first" in all ways except contention in indeterministic computations — except for membership in power domains. Functions have the rights of this new, intermediate, class.

There are a couple of justifications for this kind of "second class" status for functions. Easiest to understand is the practical observation that indeterminsim, as practiced in operating systems, is restricted to choice among signals — among ground values. A systems programmer places indeterminate control at a unique (and delicate) level in his system, a level where functions (i.e. programs) are static — and very reliable. Pragmatic indeterminism selects, but never constructs, running programs.

Second would be a conflict between syntactic and semantic indeterminism if functions would appear within power domains. As I intend that an element from a powerdomain semantically represent only runtime contention (between incomparable values), I would have a problem with any element that included both a ground value and a function. Any strongly typed system [15] would deny such semantic contention and could syntactically force the choice of the function (if needed as the left operand to an application) or of the ground value (if needed as printable output) regardless of termination behavior[2].

The domain equations that follow, therefore, separate functions (called systems) from ground values (roughly, printable data). Denoted values are either systems or ground values; expressed values (results from systems) are either systems or *sets* of ground values:

$$\mathbf{D} = \mathbf{G} + \mathbf{S} \qquad \text{(denoted values)}$$
$$\mathbf{E} = \mathcal{P}_{EM}(\mathbf{G}) + \mathbf{S} \qquad \text{(expressed values)}$$
$$\mathbf{S} = [\mathbf{D} \to \mathbf{E}] \qquad \text{(systems)}$$

1.3.4. Fixed Points

For reasons to be developed, the remainder of this paper deals with finding the analog to a fixed point for a function in $[\mathbf{G} \to \mathcal{P}(\mathbf{G})]$. That is, if

$$\pi \in \mathbf{P} = [\mathbf{G} \to \mathcal{P}(\mathbf{G})] \qquad \text{(programs)}$$
$$\phi \in \mathbf{F} = [\mathbf{G} \to \mathbf{G}] \qquad \text{(functions)}$$

then from π we want to construct an analog to **fix**, here mispelled $\mathbf{fyx} \in [\mathbf{P} \to \mathcal{P}(\mathbf{G})]$;

$$\mathbf{fyx}(\pi) = \text{The least element, } \epsilon \in \mathcal{P}_{EM}(\mathbf{G}), \text{ such that}$$
$$\epsilon = \left\{ ((\gamma_1 \sqsubseteq \gamma_2) \to \gamma_2, \gamma_1) \mid \gamma_1 \in \epsilon \ \&\ \gamma_2 \in \pi\gamma_1 \right\} .$$

The preceding operational characterization has a formal characterization; the main result of this paper is proving these two equivalent:

$$\mathbf{fyx}(\pi) = \{\mathbf{fix}\ \phi \mid \forall\, \gamma \in \mathbf{G}(\phi\gamma \in \pi\gamma)\} .$$

The first formulation of **fyx** suggests a fixed point and even a proof rule without giving any indication that it might be proved continuous. The second formulation is philosphically justifiable and admits a proof of ω-continuity, but it gives no hint of an operational semantics for discharging the implicit existential on ϕ.

I argue that environmental transparency suggests that **fyx**'s desired image in $\mathcal{P}(\mathbf{G})$ is described pointwise in terms of single ground values in **G**. Just as natural extension of application to argument-values in $\mathcal{P}(\mathbf{G})$ (the application axiom uniformly used below) "distributes application over choice", so also is the **fyx** analog to be defined pointwise; we intrepet π as an assemblage of ω-continuous ground functions, ϕ:

$$\pi = \lambda\hat{\gamma}.\pi\hat{\gamma} = \lambda\hat{\gamma}.\{\gamma \in \pi\hat{\gamma}\} = \lambda\hat{\gamma}.\{\phi\hat{\gamma} \mid \forall\, \gamma(\phi\gamma \in \pi\gamma)\} ,$$

and specify that the set of all their fixed points is the value intended by $\mathbf{fyx}(\pi)$.

---

[2] If forced to create an indeterministic choice among similarly typed functions, I would, instead, write a *single* new function whose *result* reflects the indeterministic choice.

In the following **fyx** will appear as ψ.

1.4. MacQueen's Problem

The direct challenge addressed by this paper is a clean formal denotational semantics (with philosophical insights derived from polished formalism), that provides the intuitive meaning for a problem by David MacQueen, presented completely in Sections 5 and 7. MacQueen poses the problem of deriving "suitable" semantics for evaluation of an expression of the form

(letrec I {1 2 (if (first I)=1 then 3 else 4)})

where the braces here indicate all permutations of the enclosed items. Drawing from the notation of group theory, where S_3 indicates the permutation group of order 3, let

$$S_{<1\,2\,3>} = \{1\ 2\ 3\}$$

here denote the set of the permutations of the three explicit atoms 1, 2, and 3; its cardinality is six. Then the preceding recursive expression might be interpreted as

$$\textbf{(letrec I } S_{<1\,2\,(\,((\text{first I})=1)\text{-}\,3,4)>}\textbf{)}$$

where I stands, in turn, for each one of the possible permutations of 1, 2, and either 3 or 4 depending (circularly) on what I is. It should evaluate to a set of at most six lists (perhaps less if $\perp$ occurs, because $\perp$ may indicate more than one "unrealized" alternative.)

What is a "suitable" permutation; what is possible? Let us consider possible permutations intended by the recursion. Surely (1 2 3), (1 3 2), (2 1 4), and (2 4 1) are allowable. What about a permutation beginning with 3? There are none, because the presence of 3 depends circularly on that permutation beginning with 1.

What about *computing* permutations beginning with 4? By an argument on circularity, we conclude that there is no computationally plausible reasoning that requires 4 to begin a permutation. While such permutations, once "in hand", are included in some fixed-point under the desired semantics, they cannot be computed if they are "in the bush"; that is, they are not included in a (plausible) *least* fixed point of that denotational semantics. The strongest intuitive (least fixed-point) argument we can cast is that ($\perp$ 1 2) and ($\perp$ 2 1) are the last two legitimate permutations, completing the census of six.

1.5. Streams and Fairness

Three other points must be made regarding the intent of this work. First, a principal intention was to allow more than flat underlying domains. Although this might be perceived as a rebuttal against researchers who naively propose implementations of unspecified objects [6,7], the importance of nonflat ground domains, specifically those allowing for LISP list-structures, should not be overlooked for general operating systems. Through streams [13] this domain allows for modeling the true input/output behavior of operating systems. Also, internal data structures of arbitrary complexity thereby become available for system support.

Second is the intention to allow lazy evaluation or call-by-need operational semantics. This provides a way of implementing streams, but it also has subtle implications for the way recursive domain equations are read. Infinite list/nesting structures *are* intended by an equation like

$$\mathbf{L = \{eof\} + A \times L}\ .$$

With infinite list structures somewhere in the domains, it becomes possible to consider issues of *fairness*.

Considerations of fairness are beyond the original scope of this research. It is a concept that has little meaning in denotational semantics, which treats eventualities as realizations because it interprets systems as limit points which (if non-isolated) might only really exist after Armageddon. Bounded fairness makes no sense because (even in a power domain) continuity is defined without any regard to bound, and unbounded fairness is difficult to grasp because of completeness (the difference between $\sqsubseteq$ and $\mathbb{\sqsubseteq}$).

1.6. Power Domain Definitions and Closure

The conventional Egli-Milner power domain [16] (sometimes called the "Plotkin domain") is the principle tool of this paper. The crispest definition, given by Hennessy and Plotkin [9], characterizes it as the initial algebra on the power set with a union function, $\cup$, meeting axioms of associativity, commutativity, and absorbtion:

$$\begin{array}{lll} \forall X,Y,Z & ((X \cup Y) \cup Z) = (X \cup (Y \cup Z)) ; & \text{(Associativity)} \\ \forall X,Y & (X \cup Y) = (Y \cup X) ; & \text{(Commutativity)} \\ \forall X & (X \cup X) = X . & \text{(Absorption)} \end{array}$$

It is weaker than those that follow. The Smyth domain [20] may be characterized by adding this inequality to the rules above:

$$\forall X,Y \qquad (X \cup Y) \sqsubseteq X . \qquad \text{(Inequation)}$$

The Smyth-upside-down-domain, (called by some the "Hoare domain"), is so-called because it derives from the the Egli-Milner ordering: its "other half" with respect to the Smyth ordering Hennessy and Plotkin characterize it by adding the inequality:

$$\forall X,Y \qquad X \sqsubseteq (X \cup Y) . \qquad \text{(Inequation)}$$

The stronger definitions that appear below define domains isomorphic to Hennessy's and Plotkin's, the difference being specific inclusion of non-isolated limit points within each element of the power domain.

Definition: A domain G is an ω-algebraic, complete partial-order.

Definition: (Egli-Milner) Let P be the closed set:

$$P = \left\{ X \subset G \mid \forall \{\gamma_i\}_{i\in\omega} (\forall i \in \omega (\gamma_i \in X \;\&\; \gamma_i \sqsubseteq \gamma_{i+1}) \supset (\sqcup_i \gamma_i \in X)) \;\&\; \forall \gamma_1,\gamma_2,\gamma_3 \in G \; ((\gamma_1,\gamma_3 \in X \;\&\; \gamma_1 \sqsubseteq \gamma_2 \sqsubseteq \gamma_3) \supset (\gamma_2 \in X)) \right\};$$

and let $\sqsubseteq_P$ be the preorder defined on P by

$X_1 \sqsubseteq_P X_2$ if and only if both

$\forall \gamma_1 \in X_1 \; (\exists \gamma_2 \in X_2 \; (\gamma_1 \sqsubseteq_G \gamma_2))$, and $\forall \gamma_2 \in X_2 \; (\exists \gamma_1 \in X_1 \; (\gamma_1 \sqsubseteq_G \gamma_2))$.

Then the domain $\boldsymbol{P}_{EM}(G)$ is the set of the equivalence classes of $\sqsubseteq_P$ with the $\sqsubseteq_{EM}$ partial order defined by

$E_1 \sqsubseteq_{EM} E_2$ if and only if $\forall X_1 \in E_1 (\forall X_2 \in E_2 \; ((X_1 \sqsubseteq_P X_2))$.

Definition: (Smyth) Let P be the closed set:

$$P = \left\{ X \subset G \mid \forall \{\gamma_i\}_{i\in\omega} (\forall i \in \omega (\gamma_i \in X \;\&\; \gamma_i \sqsubseteq \gamma_{i+1}) \supset (\sqcup_i \gamma_i \in X)) \;\&\; \forall \gamma_1,\gamma_2 \in G \; ((\gamma_1 \in X \;\&\; \gamma_1 \sqsubseteq \gamma_2) \supset (\gamma_2 \in X)) \right\};$$

and let $\sqsubseteq_P$ be the preorder defined on P by

$X_1 \sqsubseteq_P X_2$ if and only if $\forall \gamma_2 \in X_2 \; (\exists \gamma_1 \in X_1 \; (\gamma_1 \sqsubseteq_G \gamma_2))$.

Then the domain $\boldsymbol{P}_{Sm}$ is the set of the equivalence classes of $\sqsubseteq_P$ with the $\sqsubseteq_{Sm}$ partial order defined by

$E_1 \sqsubseteq_{Sm} E_2$ if and only if $\forall X_1 \in E_1 (\forall X_2 \in E_2 \; ((X_1 \sqsubseteq_P X_2))$.

Definition: (Smyth-upside-down) Let P be the closed set:

$$P = \left\{ X \subset G \mid \forall \{\gamma_i\}_{i\in\omega} (\forall i \in \omega (\gamma_i \in X \;\&\; \gamma_i \sqsubseteq \gamma_{i+1}) \supset (\sqcup_i \gamma_i \in X)) \;\&\; \forall \gamma_2,\gamma_3 \in G \; ((\gamma_3 \in X \;\&\; \gamma_2 \sqsubseteq \gamma_3) \supset (\gamma_2 \in X)) \right\};$$

and let $\sqsubseteq_P$ be the preorder defined on P by

$X_1 \sqsubseteq_P X_2$ if and only if $\forall \gamma_1 \in X_1 \; (\exists \gamma_2 \in X_2 \; (\gamma_1 \sqsubseteq_G \gamma_2))$.

Then the domain $\boldsymbol{P}_{Sud}(G)$ is the set of the equivalence classes of $\sqsubseteq_P$ with the $\sqsubseteq_{Sud}$ partial order defined by

$E_1 \sqsubseteq_{Sud} E_2$ if and only if $\forall X_1 \in E_1 (\forall X_2 \in E_2 \; ((X_1 \sqsubseteq_P X_2))$.

With the meanings of X and E taken as in these definitions, simple set membership, $\gamma \in E$, is used to abbreviate $\gamma \in X \in E$.

As in lattice theory, the unary up and down arrows are taken to be the closure to upper and lower sets, respectively.

Definition [8]: For $X \subset G$, define

$$\uparrow X = \{\gamma_2 \in G \mid \exists \gamma_1 \in X \ (\gamma_1 \sqsubseteq \gamma_2)\} ;$$
$$\downarrow X = \{\gamma_1 \in G \mid \exists \gamma_2 \in X \ (\gamma_1 \sqsubseteq \gamma_2)\} .$$

The Smyth order might have been defined using $X = \uparrow X$; the Smyth-upside-down order may be described with $X = \downarrow X$.

Definition [8]: Define the upper and lower closures of a function $\pi \in [D \to \boldsymbol{P}(G)]$ by extension:

$$\uparrow\pi = \lambda\delta.\ \uparrow(\pi\delta) ;$$
$$\downarrow\pi = \lambda\delta.\ \downarrow(\pi\delta) .$$

While the singleton $\{\gamma\}$ is a valid element in the Egli-Milner power domain, the arrows on $\uparrow\{\gamma\}$ and $\downarrow\{\gamma\}$ are required for precise specifications of singleton elements in the Smyth and Smyth-upside-down domains, respectively.

Elements of these domains are closed in two ways. First, they are complete (containing limit points of monotone subsets). Secondly, they are expressible as intersections of upper sets with lower sets[3] [8]. Each equivalence class is represented by the largest set in it, which may be derived by taking the appropriate $\uparrow/\downarrow$ closure of any set in the class. In the latter case, $\boldsymbol{P}_{Sud}(G)$, each element in the power domain (a union of lower sets) may alternatively be represented by the set of its *representative elements* in G, which are the (maximal) mutually-incomparable elements of G that occur in all sets of that class.

1.7. Power Domain Well-Below

Each power domain $\sqsubseteq_{\boldsymbol{P}}$ ordering rule can extended to the "well below" relation $\sqsubset_{\boldsymbol{P}}$ by:

Definition: $X_1 \sqsubset_{Sud} X_2$ if and only if $\forall \gamma_1 \in X_1 \ (\ \exists \gamma_2 \in X_2 \ (\ \gamma_1 \sqsubset_G \gamma_2\ ))$.

Definition: $X_1 \sqsubset_{EM} X_2$ if and only if

$$\forall \gamma_1 \in X_1 \ (\exists \gamma_2 \in X_2 \ (\gamma_1 \sqsubset_G \gamma_2)\ ) \text{ , and } \forall \gamma_2 \in X_2 \ (\exists \gamma_1 \in X_1 \ (\gamma_1 \sqsubset_G \gamma_2)\ ).$$

1.8. Summing Domains

Summing of domains is usually effected by the usual disjoint sum operator, +, on complete partial orders. The lattice-theoretic sum, $\boxplus$, is sometimes used below, but only where the specific domain equation is *not* reflexive. The $\boxplus$ sum introduces both $\top$ and $\bot$ from the original lattice theoretic development of domains [21], where $\top$ may be here interpreted as the result of an error from clashing types in indeterministic programs.

Such a $\top$ value may result, for instance, where a program indeterministically returns either a ground value or a function. This kind of behavior is generally the result of poor programming style, which can be avoided by installing a strong (polymorphic [15]) typing mechanism in the language. The languages presented here lack such a mechanism so that we may more clearly consider specific issues of fixed points.

1.9. Infinite tuples

The reader is again reminded that domain equations of the form

$$L = \{eof\} + A \times L .$$

are used to indicate both finite and infinite nesting. One may use lazy evaluation or call-by-need

---

[3] where the trivial power-domain element, G, is the only intersecting lower (upper) set for Smyth (respectively, Smyth-upside-down) domains.

to achieve the operational behavior required from such denotations. An alternative (not used here) is to interpret the Cartesian product, above, as a coalesced product [2] generating a flat domain, any of whose elements is finite.

Infinite products are desirable in order to admit streams [13]. They are, however, undesirable as objects of *fair* contention between processors; fairness is not considered in this paper and so any tuple can be infinite.

Section 8 presents a domain of infinite structures not underlying a power domain, and for which the issue of fairness does not, therefore, arise. With such deterministic structures available, the stronger theory (required by this weak restriction on tuple nesting) may not be necessary and fairness might become tractable.

1.10. String Notation

Strings are used later in Section 4.4, Procedure 3, to construct extended subscripts of arbitrary length. Lower case Roman x is used to represent an arbitrary string of integers. The empty string is denoted by Λ . Infix period indicates string concatenation.

1.11. Outline

The remainder of this paper is divided into ten parts. Parts 2, 3, and 4 develop the denotational semantics used, and link it to procedures for implementation. Section 2 presents the definition of a skeletal semantics that is established by a continuity proof in Section 3. Section 4 ties it to operational procedures. Section 5 presents and solves MacQueen's problem in an artificial language that uses triples for data structures. A full blown language definition [7,10,12] appears in Section 6, which is used in Section 7 to solve MacQueen's problem with more surgical results using available strictness. Sections 8, and 9 present peripheral observations: Section 8 introduces a data structure that *can* contain programs unlike the earlier ones, Section 9 presents abbreviated examples of stream merge, and codes for expressing the several indeterminate operators in terms of one another. Finally, Section 10 offers some conclusions for powerdomain semantics that should influence design of this sort of programming language.

2. Denotational semantics for an indeterminate language

The notation in this section follows Stoy [21].

2.1. Syntactic Categories

I ∈ **Ide**	(the usual identifiers)
K ∈ **Con**	(constants)
E ∈ **Exp**	(expressions)

2.2. Syntax

K :: if | amb | *etc.* (See Section 5.1.1 or 6.2)
E :: I | K | (E_0 E_1) | (**lambda** I E) | (**letrec** I E)

2.3. Value Domains

$\rho \in U = [Ide \to D]$ (environments)

$\delta \in D = G \boxplus S$ (denoted values)
$\epsilon \in E = \mathcal{P}_{EM}(G) \boxplus S$ (expressed values)
$\sigma \in S = [D \to E]$ (systems)

$\pi \in P = [G \to \mathcal{P}_{EM}(G)]$ (programs)
$\phi \in F = [G \to G]$ (functions)
$\gamma \in G$ (ground values)

Environments map identifiers onto denoted values which are "elementary" to us; those are the basic "chunks" of computation. Such values are not indeterminate as conceived; indeterminism deals with the situation where more than one such simple value is an acceptable computed result, throwing the answer into a power domain. That is, arguments are these elementary chunks; results might be indeterminate.

One chunk that we perceive as elementary contains "packaged" indeterminism: the system functions that we write and perceive as a single unit, but which may elicit indeterminate behavior on identical input. An operating system that can generate any of several results is still perceived as a *unique* object instead of an indeterminate set of functions that, together, represent the system.

Expressed values are the result of such systems, and often include a set of ground values, whereas denoted values are, at worst, single ground values. If an element of either domain is interpreted as a system, then a unique system is implied.

The substance of this paper is determining how to interpret the **fix** operator on $\sigma \in S$, on systems. The problem can be interpreted as finding fixed points of functions in S, which are either trivial — $\perp_E$ — or are to be found as fixed points in terms from the following domain quasi-distribution[4]:

$$S = [D \to E] = [G \boxplus S] \to [\mathcal{P}_{EM}(G) \boxplus S]$$
$$\simeq [G \to \mathcal{P}_{EM}(G)] \boxplus [G \to S] \boxplus [S \to \mathcal{P}_{EM}(G)] + [S \to S] .$$

Based on the knowledge that all we seek are fixed points from S, only the first of the four terms distributed from S appears puzzling. Conventional semantics provides a denotational and operational semantics for fixed points of $[S \to S]$, and fixing in either of the middle two terms in E results in $\perp_E$. What follows, then is a study of "fixed points" from $[G \to \mathcal{P}_{EM}(G)] = P$.

The additional domains P and F are explicitly defined because the object of the following study is $[G \to \mathcal{P}_{EM}(G)]$. We need these domains labeled because they become the center of out attention as the theory unfolds. These domains do *not*, however, appear in any semantics of any particular language; the later definitions of the evaluation functions $\mathcal{E}$ and $\mathcal{K}$ do not use them. Only the domains, D, E, and S, as well as whatever is necessary for structure within G, will be specified for a particular language.

2.4. Notational Conventions

Definition: $\nabla : D \to E$ is defined as a type coercer:
$\nabla = \lambda\delta.(\delta \in G \to \{\delta|G\}$**in** $E, (\delta \in S \to \delta|S$ **in** $E, (\delta = \top_D \to \top_E, \perp_E)))$.

Definition: $\phi \leq \pi$, read "ϕ is dominated by π", means $\forall \gamma \in G\ (\phi\gamma \in \pi\gamma)$.

Notation: When $\sigma \perp_D \in \mathcal{P}(G)$ we write $\phi \leq \sigma$ to mean $\phi \leq \lambda\gamma.(\sigma(\gamma$ **in** $D)|\mathcal{P}(G))$.

---

[4] I do not claim that $\to$ literally (cf. use of "=" to "$\simeq$" here) distributes over $\boxplus$, but I suggest that the problem be considered as if it did, under a kind of subdomain embedding.

The concept of $\pi \in \mathbf{P}$ dominating $\phi \in \mathbf{F}$ derives from the perspective that each π really stands for the amorphous collection of such ϕ; hence, the centrality of the $\preccurlyeq$ relation whenever **P** occurs below.

The symbol $\preccurlyeq$ is at once suggestive of set membership (ϵ) and, to a lesser degree, of a partial ordering ($<$). An ordering because, under the Smyth-upside-down power domain, it is certainly true that

$\phi_1 \sqsubseteq \phi_2$, $\phi_2 \preccurlyeq \pi_1$, and $\pi_1 \sqsubseteq \pi_2$ together imply that $\phi_1 \preccurlyeq \pi_1$, $\phi_2 \preccurlyeq \pi_2$, and (hence) $\phi_1 \preccurlyeq \pi_2$.

Set membership reads directly from the definition of that symbol by "cancelling" the universally quantified γ.

2.5.
Semantic Functions

$$\boldsymbol{K} : [\mathbf{Con} \to \mathbf{D}] \qquad \text{Definition detailed later}$$
$$\boldsymbol{E} : [\mathbf{Exp} \to [\mathbf{U} \to \mathbf{E}]]$$

$$\boldsymbol{E}\,[\![\mathrm{I}]\!]\rho \equiv \nabla\,(\rho\,[\![\mathrm{I}]\!]) \qquad \text{(parameter)}$$

$$\boldsymbol{E}\,[\![\mathrm{K}]\!]\rho \equiv \nabla\,(\boldsymbol{K}\,[\![\mathrm{K}]\!]) \qquad \text{(constant)}$$

$$\boldsymbol{E}\,[\![(\mathbf{lambda}\ \mathrm{I}\ \mathrm{E})]\!]\rho \equiv (\lambda\delta.\boldsymbol{E}\,[\![\mathrm{E}]\!]\rho[\delta/\mathrm{I}])\ \mathbf{in}\ \mathbf{E} \qquad \text{(function)}$$

$$\begin{aligned}\boldsymbol{E}\,[\![(\mathrm{E}_0\ \mathrm{E}_1)]\!]\rho \equiv\ & (\mathbf{Let}\ \sigma = ((\boldsymbol{E}\,[\![\mathrm{E}_0]\!]\rho)|\mathbf{S})\ \mathbf{and}\ \epsilon = (\boldsymbol{E}\,[\![\mathrm{E}_1]\!]\rho)\ \mathbf{in} \\ & (\epsilon \in \boldsymbol{P}_{\mathrm{EM}}(\mathbf{G}) \to \bigsqcup_{\mathbf{E}}\{\sigma(\gamma\ \mathbf{in}\ \mathbf{D}) \mid \gamma \in (\epsilon|\boldsymbol{P}_{\mathrm{EM}}(\mathbf{G}))\}, \\ & (\epsilon \in \mathbf{S} \to \sigma(\epsilon|\mathbf{S}\ \mathbf{in}\ \mathbf{D}), \\ & (\epsilon = \top_{\mathbf{E}} \to \sigma\top_{\mathbf{D}}, \\ & \sigma\perp_{\mathbf{D}}))))\end{aligned} \qquad \text{(application)}$$

$$\begin{aligned}\boldsymbol{E}\,[\![(\mathbf{letrec}\ \mathrm{I}\ \mathrm{E})]\!]\rho \equiv\ & (\mathbf{Let}\ \sigma = (\lambda\delta.\boldsymbol{E}\,[\![\mathrm{E}]\!]\rho[\delta/\mathrm{I}])\ \mathbf{in} \\ & (\sigma\perp_{\mathbf{D}} \in \boldsymbol{P}_{\mathrm{EM}}(\mathbf{G}) \to \{\mathbf{fix}\ \phi \mid \phi \preccurlyeq \sigma\}\mathbf{in}\ \mathbf{E}, \\ & (\sigma\perp_{\mathbf{D}} \in \mathbf{S} \to \mathbf{fix}\ \lambda\epsilon.\sigma(\epsilon|\mathbf{S}\ \mathbf{in}\ \mathbf{D}), \\ & (\sigma\perp_{\mathbf{D}} = \top_{\mathbf{E}} \to \top_{\mathbf{E}}, \\ & \perp_{\mathbf{E}})))\end{aligned} \qquad \text{(recursion)}$$

The first three axioms, above, are not remarkable. The last two provide for four cases each. The application rule uses "natural extension" of ordinary application to bind one δ at a time in applying a system σ; the only interesting situation arises from an argument in $\boldsymbol{P}_{\mathrm{EM}}(\mathbf{G})$, from which one γ at a time is taken through **D**. This uniqueness of binding achieves environmental transparency.

(For $\epsilon = \perp_{\mathbf{E}}$, the result $\sigma\perp_{\mathbf{D}}$ is available operationally through lazy evaluation of the body of σ before testing whether $\epsilon \in \boldsymbol{P}_{\mathrm{EM}}(\mathbf{G})$ or $\epsilon \in \mathbf{S}$; this tack is not necessary in general, but it preserves our ability to specify constant-functions σ.)

The first alternative in the axiom for recursion motivates Section 3, its justification. The rest is straightforward and would support a proof of the continuity of $\boldsymbol{E}$ by structural induction on $\mathrm{E} \in \mathbf{Exp}$.

---

3. Continuity of $\boldsymbol{E}$

In the proofs that follow the relations $\sqsubseteq_{\mathrm{Sm}}$ and $\sqsubseteq_{\mathrm{Sud}}$ are used over the $\boldsymbol{P}_{\mathrm{EM}}(\mathbf{G})$ domain to suggest the corresponding half of the $\sqsubseteq_{\mathrm{EM}}$ relation. These relations are used outside their

associated Smyth and Smyth-upside-down domains, so that no additional closures are necessary; their definitions are the same as it is under those domains.

Definition: $\eta : [\omega \to \mathbf{P} \to \mathbf{P}]$;
$\eta = \lambda n \lambda \pi \lambda \gamma . \{\phi^n \gamma \mid \phi \ll \pi\}$.

Theorem 1: For all $n \in \omega$, ηn is monotone.

Proof: The relations $\sqsubseteq_{Sm}$ and $\sqsubseteq_{Sud}$ are used over the $\boldsymbol{P}_{EM}(\mathbf{G})$ domain to suggest the corresponding half of the $\sqsubseteq_{EM}$ relation.
Let $\pi_1 \sqsubseteq_{EM} \pi_2$ be both in **P**. Then for all γ :

$$\begin{aligned}
\eta n \pi_1 \gamma &= \{\phi^n \gamma \mid \phi \ll \pi_1\} \\
&\sqsubseteq_{Sm} \uparrow\{\phi^n \gamma \mid \forall \gamma (\phi\gamma \in \pi_1 \gamma)\} = \{\phi^n \gamma \mid \forall \gamma (\phi\gamma \in (\uparrow \pi_1 \gamma))\} \\
&\supset \{\phi^n \gamma \mid \forall \gamma (\phi\gamma \in (\uparrow \pi_2 \gamma))\} = \uparrow\{\phi^n \gamma \mid \forall \gamma (\phi\gamma \in \pi_2 \gamma)\} \\
&\sqsubseteq_{Sm} \{\phi^n \gamma \mid \forall \gamma (\phi\gamma \in \pi_2 \gamma)\} = \{\phi^n \gamma \mid \phi \ll \pi_2\} = \eta n \pi_2 .
\end{aligned}$$

$$\begin{aligned}
\eta n \pi_1 \gamma &= \{\phi^n \gamma \mid \phi \ll \pi_1\} \\
&\sqsubseteq_{Sud} \downarrow\{\phi^n \gamma \mid \forall \gamma (\phi\gamma \in \pi_1 \gamma)\} = \{\phi^n \gamma \mid \forall \gamma (\phi\gamma \in (\downarrow \pi_1 \gamma))\} \\
&\subset \{\phi^n \gamma \mid \forall \gamma (\phi\gamma \in (\downarrow \pi_2 \gamma))\} = \downarrow\{\phi^n \gamma \mid \forall \gamma (\phi\gamma \in \pi_2 \gamma)\} \\
&\sqsubseteq_{Sud} \{\phi^n \gamma \mid \forall \gamma (\phi\gamma \in \pi_2 \gamma)\} = \{\phi^n \gamma \mid \phi \ll \pi_2\} = \eta n \pi_2 .
\end{aligned}$$

Thus, $\eta n \pi_1 \sqsubseteq_{Sm} \eta n \pi_2$ and $\eta n \pi_1 \sqsubseteq_{Sud} \eta n \pi_2$; hence $\eta n \pi_1 \sqsubseteq_{EM} \eta n \pi_2$. □

Lemma 1: If $\phi \ll \pi$, ϕ is isolated, and $\pi = \sqcup_i \pi_i$, where $\{\pi_i\}_{i \in \omega}$ satisfies $\forall i (\pi_i \sqsubseteq_{EM} \pi_{i+1})$, then there exists a k such that $\phi \ll \pi_k$.

Proof: Given ϕ isolated, define $\hat{\pi} = \lambda \gamma . \{\phi\gamma\}$. Obviously, $\phi \ll \hat{\pi}$ and $\hat{\pi}$ is isolated because ϕ is.
Because $\phi \ll \pi$, we have $\pi \sqsubseteq_{Sm} \hat{\pi} \sqsubseteq_{Sud} \pi = \sqcup_i \pi_i$.
But since $\hat{\pi}$ is isolated and $\hat{\pi} \sqsubseteq_{Sud} \sqcup_i \pi_i$, there must exist some $m \in \omega$ such that $\hat{\pi} \sqsubseteq_{Sud} \pi_m$;
similarly $\sqcup_i \pi_i \sqsubseteq_{Sm} \hat{\pi}$, so there must exist some $n \in \omega$ such that $\pi_n \sqsubseteq_{Sm} \hat{\pi}$.
Let $k = \max(m,n)$ so that: $\pi_n \sqsubseteq_{Sm} \pi_k \sqsubseteq_{Sm} \hat{\pi} \sqsubseteq_{Sud} \pi_m \sqsubseteq_{Sud} \pi_k$.
Thus, $\phi \ll \pi_k$. □

Lemma 2: If $\phi \lessdot \pi$ and π is approximated by $\{\pi_i\}_{i\in\omega}$:

$$\pi = \sqcup_i \pi_i \ , \ \pi_0 = \bot_{\mathbf{P}} \ , \text{ and } \forall i \in \omega(\pi_i \sqsubseteq \pi_{i+1}),$$

then there exists $\{\phi_i\}_{i\in\omega}$ such that $\phi_0 = \bot_{\mathbf{F}}$, $\sqcup_i \phi_i = \phi$, and

$$\forall i \in \omega(\ \phi_i \in \mathbf{F} \ \& \ \phi_i \sqsubseteq \phi_{i+1} \ \& \ \phi_i \text{ isolated } \& \ \forall \gamma(\ \pi_i\gamma \sqsubseteq_{Sm} \{\phi_i(\gamma)\} \ )) \ .$$

Proof (after one by Clinger): Let $\{\hat{\phi}_j\}_{j\in\omega}$ be an increasing sequence of isolated points approximating ϕ.
$\phi = \sqcup_j \hat{\phi}_j$, $\phi_0 = \bot_{\mathbf{F}}$, and $\forall j(\hat{\phi}_j \text{ isolated } \& \ \hat{\phi}_j \sqsubseteq \hat{\phi}_{j+1})$.
Define $\phi_i = \phi_k$ where $k = \min\{ \ j \mid \pi_i \sqsubseteq_{Sm} \lambda\gamma.\{\hat{\phi}_k\gamma\} \ \}$.
It is claimed that

- ϕ_i is well-defined, since for all i:

  $$\pi_i \sqsubseteq_{Sm} \pi \sqsubseteq_{Sm} \lambda\gamma.\{\phi\gamma\} = \sqcup_j(\lambda\gamma.\{\hat{\phi}_j\gamma\}) \ ,$$

  because $\phi \lessdot \pi$. Because every $\hat{\phi}_j$ is isolated, there must exist j_i such that

  $$\pi_i \sqsubseteq_{Sm} \lambda\gamma.\{\hat{\phi}_{j_i}\} \ .$$

  Thus, k in the definition of ϕ_i is at most j_i .
- ϕ_i is isolated, since every $\hat{\phi}_j$ is.
- $\phi_i \in \mathbf{F}$ is monotone and continuous, as are all $\phi \in \mathbf{F}$.
- $\phi_i \sqsubseteq \hat{\phi}_i$ since $\pi_i \sqsubseteq \pi_{i+1}$ and $\hat{\phi}_j \sqsubseteq \hat{\phi}_{j+1}$ in the definition of ϕ_i.
- $\pi_i\gamma \sqsubseteq_{Sm} \{\phi_i\gamma\}$ by the definition of ϕ_i .
- $\phi = \sqcup_i \phi_i$ according to the argument that follows.

  From Lemma 1 for every $\hat{\phi}_j$ isolated, there is some k_j such that

  $$\pi_{k_j}\gamma \sqsubseteq_{Sm} \lambda\gamma.\{\hat{\phi}_j\gamma\} \ .$$

  So $\hat{\phi}_j \sqsubseteq \phi_{k_j}$.

  From the definition of ϕ_i we already have $\phi_i \sqsubseteq \sqcup_j \hat{\phi}_j$.

  Taking the join:

  $$\phi = \sqcup_j \hat{\phi}_j \sqsubseteq \sqcup_i \phi_i \sqsubseteq \sqcup_j \hat{\phi}_j = \phi \ .$$

□

Lemma 3: If $\phi \lessdot \pi$ and π is approximated by $\{\pi_i\}_{i\in\omega}$:

$$\pi = \sqcup_i \pi_i \ , \ \pi_0 = \bot_{\mathbf{P}} \ , \text{ and } \forall i \in \omega(\pi_i \sqsubseteq \pi_{i+1}),$$

then there exists $\{\phi_i\}_{i\in\omega}$ such that $\phi_0 = \bot_{\mathbf{F}}$, $\sqcup_i \phi_i = \phi$, and

$$\forall i \in \omega(\ \phi_i \in \mathbf{F} \ \& \ \phi_i \sqsubseteq \phi_{i+1} \ \& \ \phi_i \text{ isolated } \& \ \phi_i \lessdot \pi_i) \ .$$

Proof: Let $\{\hat{\phi}_j\}_{j\in\omega}$ be an increasing sequence of isolated points approximating ϕ as in the proof of Lemma 2 above, but where this decomposition is sufficiently refined that, for all γ and for all isolated π_i , there is some j such that $\phi_j\gamma \in \pi_i\gamma$. This is a reasonable requirement; it says that every γ is mapped into every set $\pi_i\gamma$ by one of the ascending $\hat{\phi}_j$'s. The alternative is that some such $\pi_i\gamma$ are jumped over by the ascending $\{\hat{\phi}_j\}_{j\in\omega}$.

Define $\phi_i = \lambda\gamma.(\hat{\phi}_j\gamma$ where j is minimum such that $\hat{\phi}_j\gamma \in \pi_i\gamma)$.

It is claimed that

- ϕ_i is well defined since the refinement of $\{\hat{\phi}_j\}_{j\in\omega}$ guarantees that a minimum j exists as required in the definition of ϕ_i.
- ϕ_i is isolated since all $\hat{\phi}_j$ are and, by Lemma 2, there is an upper bound to j in the definition of any ϕ_i. Thus, each ϕ_i is defined from finitely many isolated functions $\hat{\phi}_j$.
- ϕ_i is monotone because π_i is:
  If $\gamma_1 \sqsubseteq \gamma_2$ then
  $\phi_i\gamma_1 = \hat{\phi}_j\gamma_1$ where j is minimum s.t. $\hat{\phi}_j\gamma_1 \in \pi_i\gamma_1 \sqsubseteq \pi_i\gamma_2$.
  $\phi_i\gamma_2 = \hat{\phi}_k\gamma_2$ where k is minimum s.t. $\hat{\phi}_k\gamma_2 \in \pi_i\gamma_2$.
  Therefore, as qualified above, the smallest j is less than the smallest k , and $\phi_i\gamma_1 \sqsubseteq \phi_i\gamma_2$.
- ϕ_i is continuous, also because π_i is.
  Let $\gamma = \sqcup_k \gamma_k$ for $\{\gamma_k\}_{k\in\omega}$ ascending.
  By monotonicity
  $\sqcup_{k\in\omega}(\phi_i\gamma_k) \sqsubseteq \sqcup_k(\phi_i\sqcup_k\gamma_k) = \phi_i(\sqcup_k\gamma_k)$.
  Alternatively
  $\phi_i(\sqcup_k\gamma_k) = \hat{\phi}_j(\sqcup_k\gamma_k)$
  where j is smallest such that $\hat{\phi}_j(\sqcup_k\gamma_k) \in \pi_i(\sqcup_k\gamma_k)$
  iff $\sqcup_m(\hat{\phi}_j\gamma_m) \in \sqcup_n(\pi_i\gamma_n)$.
  So
  $\phi_i(\sqcup_k\gamma_k) \sqsubseteq \sqcup_{k,m,n\in\omega}\hat{\phi}_j(\gamma_k)$
  where j is the smallest such that $\hat{\phi}_j\gamma_m \in \pi_i\gamma_n$,
  but since $\{\gamma_k, \hat{\phi}_j, \pi_i\}$ are ascending and continuous:
  $\sqsubseteq \sqcup_{k\in\omega}\phi_i\gamma_k$.
- $\phi_i \sqsubseteq \phi_{i+1}$ because $\pi_i \sqsubseteq \pi_{i+1}$ and $\hat{\phi}_j \sqsubseteq \hat{\phi}_{j+1}$.
- $\phi_i \ll \pi_i$ by definition.
- $\phi = \sqcup_i\phi_i$ because
  $\sqcup_i\phi_i = \sqcup_i(\lambda\gamma.(\hat{\phi}_j\gamma$ where j is minimum such that $\hat{\phi}_j\gamma \in \pi_i\gamma))$;
  $= \lambda\gamma.\sqcup_i(\hat{\phi}_j\gamma$ where j is minimum such that $\hat{\phi}_j\gamma \in \pi_i\gamma))$;
  $= \lambda\gamma.\ \bigsqcup\{\hat{\phi}_j\gamma \mid \{\hat{\phi}_j\gamma\} \sqsubseteq_{Sm} \sqcup_i\pi_i\}$.
  But all $\hat{\phi}_j \ll \pi = \sqcup_i\pi_i$, and so
  $\sqcup_i\phi_i = \lambda\gamma.(\sqcup_j\hat{\phi}_j\gamma) = \sqcup_j\hat{\phi}_j = \phi$.

□

Theorem 2: For all n, ηn is continuous.

Proof: Let $\pi \in [G \to \mathcal{P}_{EM}(G)]$ be approximated by an increasing sequence $\{\pi_i\}_{i\in\omega}$.
Thus, $\sqcup_i \pi_i = \pi$, $\pi_0 = \perp_P$, and $\forall i(\pi_i \sqsubseteq_{EM} \pi_{i+1})$.
Monotonicity of ηn (Theorem 1) implies $\sqcup_i \eta n \pi_i \sqsubseteq_{EM} \eta n(\sqcup_i \pi_i)$.
In order to establish continuity, we only need to show $\eta n(\sqcup_i \pi_i) \sqsubseteq_{EM} \sqcup_i(\eta n \pi_i)$.
We show this in two steps: the first for $\sqsubseteq_{Sud}$ and the second for $\sqsubseteq_{Sm}$.

Step 1 (Clinger): For all $\gamma_1 \in G$, $\eta n \pi \gamma_1 = \eta n(\sqcup_i \pi_i)\gamma_1 = \{\phi^n \gamma_1 \mid \phi \ll \sqcup_i \pi_i\}$.
If $\gamma_2 = \phi^n \gamma_1$ for $\phi \ll \sqcup_i \pi_i$, then Lemma 3 guarantees an ascending sequence
$\{\phi_i\}_{i\in\omega}$ of isolated $\phi_i \ll \pi_i$ such that $\sqcup_i \phi_i = \phi$.
Thus, $\gamma_2 = \phi^n \gamma_1 = (\sqcup_i \phi_i)^n \gamma_1 = \sqcup_i(\phi_i^n \gamma_1)$ for some set of continuous, ascending, isolated ϕ_i,
and so γ_2 is in $\downarrow(\sqcup_i(\eta n \pi_i \gamma_1))$. Such γ_2 generate $\downarrow(\eta n \pi \gamma_1)$ by definition;
so the closure of the set of all such γ_2 equals $\downarrow(\eta n \pi \gamma_1)$, and therefore, $\eta n \pi \sqsubseteq_{Sud} \sqcup_i \eta n \pi_i$.

Step 2: For all $\gamma_1 \in G$, $(\sqcup_i(\eta n \pi_i))\gamma_1 = \sqcup_i(\eta n \pi_i \gamma_1)$.
Let $\gamma_2 \in (\sqcup_i(\eta n \pi_i))\gamma_1$; then there is a sequence $\{\phi_i\}_{i\in\omega}$ such that
$\forall i \in \omega(\phi_i \ll \pi_i \;\&\; \phi_i^n \gamma_1 \sqsubseteq \phi_{i+1}^n \gamma_1)$,
and $\gamma_2 = \sqcup_i(\phi_i^n \gamma_1)$.
Since $\pi_i \sqsubseteq \pi_{i+1}$, without loss of generality we can require $\phi_i \sqsubseteq \phi_{i+1}$.
Let $\phi = \sqcup_i \phi_i \ll \sqcup_i \pi_i = \pi$.
Then $\gamma_2 = \sqcup_i(\phi_i^n \gamma_1) = (\sqcup_i \phi_i^n)\gamma_1 = \phi^n \gamma_1$.
Therefore, $\gamma_2 \in \eta n(\sqcup_i \pi_i)\Gamma 1 = \eta n \pi \Gamma 1$.
Since, for all γ_1, $(\sqcup_i \eta n \pi_i)\gamma_1$ is composed of such γ_2, we have shown that
$\eta n \pi \sqsubseteq_{Sm} \sqcup_i(\eta n \pi_i)$.

Therefore the desired continuity is established:
$\eta n \pi \sqsubseteq_{EM} \sqcup_i(\eta n \pi_i)$. □

Suppose we wrote $\eta n \pi$ using the notation π^n. This notation suggests that if one wanted an analog to the fixed-point of such a π, then the result would appear as

$\sqcup_n(\pi^n \perp_D) = \sqcup_n(\eta n \pi \perp_D)$ which is the join of an increasing sequence
($\eta n \pi \perp \sqsubseteq \eta(n+1)\pi \perp$ because $\phi \ll \pi$ are monotone)
and, thus, reaches a limit in $\mathcal{P}_{EM}(G)$;
$= \sqcup_n\{\phi^n \perp \mid \phi \ll \pi\}$;
$= \{\sqcup_n \phi^n \perp \mid \phi \ll \pi\}$ since G is continuous;
$= \{\mathbf{fix}\ \phi \mid \phi \ll \pi\}$.

So, indeed, shall we define the function ψ.

Definition: $\psi : [P \to \boldsymbol{P}_{EM}(G)]$;
$\psi = \lambda\pi.\{\mathbf{fix}\ \phi \mid \phi \lessdot \pi\}$.

Lemma 4: $\psi\pi = \sqcup_n((\eta n \pi)\perp_G)$.

Proof: Above.

Theorem 3: ψ is monotone and continuous.

Proof: Let $\pi_1 \sqsubseteq \pi_2$ be both in $[G \to \boldsymbol{P}_{EM}(G)]$. Then
by Theorem 1, $\forall n(\ \eta n \pi_1 \perp \sqsubseteq \eta n \pi_2 \perp\ )$ and, taking the join over all n,
$\psi\pi_1 = \sqcup_n(\eta n \pi_1 \perp) \sqsubseteq \sqcup_n(\eta n \pi_2 \perp) = \psi\pi_2$ by Lemma 4.

Let $\pi \in [G \to \boldsymbol{P}_{EM}(G)]$ be approximated by an increasing sequence $\{\pi_i\}_{i \in \omega}$.
$\psi\pi = \psi(\sqcup_i \pi_i) = \sqcup_n(\eta n(\sqcup_i \pi_i)\perp) = \sqcup_n \sqcup_i(\eta n \pi_i \perp) = \sqcup_i(\psi\pi_i)$ by Theorem 2. □

Theorem 4: $\boldsymbol{E}$ is monotone and continuous.

Proof by structural induction on $E \in$ **Exp**: If we set up an ordinary structural induction, we find that all operators used in the definition of $\boldsymbol{E}$, except for that at the second line of the **letrec** axiom, are well known to be monotone and continuous. Consider just that **letrec** line, where

$\sigma\perp_D \in \boldsymbol{P}_{EM}(G)$.

Let

$\pi_\sigma = \lambda\gamma.(\sigma(\gamma\ \mathbf{in}\ D)|\boldsymbol{P}_{EM}(G))$

It is here shown that $\psi\pi_\sigma$ **in E** is the result specified at that line,
and interpreting it in this way we can show that $\boldsymbol{E}$ is monotone and continuous.

π_σ is monotone and continuous in σ and Theorem 3 establishes that
ψ is monotone and continuous in π, and hence

$$\lambda\sigma.(\psi\pi_\sigma) = \lambda\sigma.(\psi\ (\lambda\gamma.(\sigma(\gamma\ \mathbf{in}\ D)|\boldsymbol{P}_{EM}(G)))\ )$$
$$= \lambda\sigma.\{\mathbf{fix}\ \phi \mid \phi \lessdot ((\lambda\gamma.(\sigma(\gamma\ \mathbf{in}\ D)|\boldsymbol{P}_{EM}(G))\}$$
$$= \lambda\sigma.\{\mathbf{fix}\ \phi \mid \phi \lessdot \sigma\}$$

is monotone and continuous when $\sigma\perp_D \in \boldsymbol{P}_{EM}(G)$.
Thus, $\boldsymbol{E}$ is defined using only monotone and continuous operations,
and ordinary structural induction on E succeeds for proving the theorem.
□

4. Operational Semantics

In this section we shall prove the correctness of procedures to generate elements of $\psi\pi$. The presentation is a purely mathematical justification of Procedure 3, below, which prompted this work.

Whereas the preceding theory was developed in the Egli-Milner domain, most of the following theory is stated in the Smyth-upside-down power domain because the proofs are constructive there. I believe that the rift is minor, as reflected in Conjecture 3.

4.1. Constructing approximations to $\psi\pi$

Lemma 5: If $\gamma = \mathbf{fix}\ \phi$ for $\phi \lessdot \pi$, then there exist two ascending sequences $\{\gamma_i, \pi_i\}_{i\in\omega}$ such that $\gamma_0 = \perp_G$, $\sqcup_i \gamma_i = \gamma$, $\sqcup_i \pi_i = \pi$, and for all i:

$\gamma_i \sqsubseteq \gamma_{i+1}$, $\pi_i \sqsubseteq \pi_{i+1}$, γ_i and π_i are isolated,

and $\{\gamma_{i+1}\} \sqsubseteq \pi_i\gamma_i \sqsubseteq \eta(i+1)\pi\perp$.

Proof: Let π be approximated by an increasing sequence of isolated points, $\{\hat{\pi}_i\}_{i\in\omega}$ where $\pi_o = \perp_P$. Lemma 3 shows that there is a sequence $\{\phi_i\}_{i\in\omega}$ such that

$\phi_i \sqsubseteq \phi_{i+1}$, ϕ_i isolated, $\phi_i \lessdot \hat{\pi}_i$, and $\sqcup_i \phi_i = \phi$.

The idea of the following construction is to define the sequence $\{\gamma_i\}_{i\in\omega}$ so that

$\gamma_i \in \eta i \pi \perp$,

or, using Knuth's termial notation [11],

$$i? = \sum_{j=0}^{i} j = \frac{i(i+1)}{2};$$

$\gamma_{i?} \in \eta i \hat{\pi}_i \gamma_{(i-1)?}$.

Since it is not termial—but its inverse—that is required, we define Q to be an inverse of termial [5]:

$Q : \omega \to \omega$;

$Qk =$ the largest integer j such that $j? \le k$;

$$Q = \lambda k. \left\lfloor \frac{\sqrt{8k+1} - 1}{2} \right\rfloor .$$

Define $\pi_i = \hat{\pi}_{Qi}$

$\gamma_0 = \perp_G$; $\gamma_{i+1} = \phi_{Qi}\gamma_i$.

For example,

$\gamma_1 = \phi_0^1\perp$; . . .

$\gamma_3 = \phi_1^2\phi_0^1\perp$; $\gamma_{i?} = \phi_{i-1}^{i}\phi_{i-2}^{i-1}\ldots\phi_0^1\perp$;

$\gamma_6 = \phi_2^3\phi_1^2\phi_0^1\perp$; . . .

The point of introducing Q here is to establish the last point below; otherwise i would suffice wherever Qi is used in the following.

It is claimed that

- $\sqcup_i \pi_i = \sqcup_i \hat{\pi}_{Qi} = \sqcup_i \hat{\pi}_i = \pi$.
- $\pi_i \sqsubseteq \pi_{i+1}$ because Q(i+1) equals either Qi or 1+Qi;
  in the latter case $\hat{\pi}_i \sqsubseteq \hat{\pi}_{i+1}$.
- $\gamma_i \sqsubseteq \gamma_{i+1}$ by simple induction. $\gamma_0 \sqsubseteq \gamma_1$ trivially.
  Assuming $\gamma_i \sqsubseteq \gamma_{i+1}$, we use Q(i+1) equals either Qi or 1+Qi
  to establish $\phi_{Qi} \sqsubseteq \phi_{Q(i+1)}$ and hence
  $\gamma_{i+1} = \phi_{Qi}\gamma_i \sqsubseteq \phi_{Qi}\gamma_{i+1} \sqsubseteq \phi_{Q(i+1)}\gamma_{i+1} = \gamma_{i+2}$
  establishing the hypothesis for i+1 .
- π_i is isolated because there exists j such that $\pi_i = \hat{\pi}_j$ and all $\hat{\pi}_j$ are isolated.
- γ_i is isolated. By induction, γ_0 trivially is, and the rest are the result of applying
  an isolated function ϕ_{Qi} to a value that is (by inductive hypothesis) isolated.
- $\gamma_{i+1} = \phi_{Qi}\gamma_i \in \hat{\pi}_{Qi}\gamma_i = \pi_i\gamma_i$ since $\forall\, i(\phi_i \lessdot \hat{\pi}_i)$.
- A simple induction on $\pi_i \sqsubseteq \pi$ shows that $\pi_i\gamma_i \sqsubseteq \pi^{(i+1)}\perp = \eta(i+1)\pi\perp$.
- $\sqcup_i \gamma_i = \mathbf{fix}\ \phi = \gamma$ from an inductive proof of the fact that
  $$\phi_i^{\,i+1}\perp \sqsubseteq \gamma_{(i+1)?} \sqsubseteq \phi_i^{\,(i+1)?}\perp .$$
  For i=0 we establish $\phi_0^{\,1}\perp \sqsubseteq \gamma_1 \sqsubseteq \phi_0^{\,1}\perp$ trivially.
  Assume this fact for i and consider i+1 :
  $$\begin{aligned}
  \phi_{i+1}^{\,i+2}\perp &\sqsubseteq \phi_{i+1}^{\,i+2}\,\gamma_{(i+1)?} \text{ since } \phi_{i+1} \text{ is monotone;}\\
  &= \gamma_{(i+2)?} \text{ by definition;}\\
  &\sqsubseteq \phi_{i+1}^{\,i+2}\,\phi_i^{\,(i+1)?}\perp \text{ by induction;}\\
  &\sqsubseteq \phi_{i+1}^{\,(i+2)}\,\phi_{i+1}^{\,(i+1)?}\perp = \phi_{i+1}^{\,(i+2)?}\perp \text{ because } \phi_i \sqsubseteq \phi_{i+1} .
  \end{aligned}$$
  Now that we have proved this inequality we shall extend it,
  observing that each ϕ_i is monotone and $\forall\, i(\phi_i \sqsubseteq \phi_{(i+1)?})$:
  $$\phi_i^{\,i}\perp \sqsubseteq \phi_i^{\,i+1}\perp \sqsubseteq \gamma_{(i+1)?} \sqsubseteq \phi_i^{\,(i+1)?}\perp \sqsubseteq \phi_{(i+1)?}^{\,(i+1)?}\perp .$$
  Because $\forall\, i(\gamma_i \sqsubseteq \gamma_{i+1})$, we then take the join over i to find
  $$\sqcup_i \gamma_i = \sqcup_i \gamma_{i?} = \sqcup_i \phi_i^{\,i}\perp = \sqcup_i \sqcup_j \phi_i^{\,j}\perp = \sqcup_i(\mathbf{fix}\ \phi_i) = \mathbf{fix}(\sqcup_i \phi_i) = \mathbf{fix}\ \phi = \gamma.$$

□

The following few results are stated in the Smyth-uypside-down domain. Therefore, the reader will note many down-arrows creating lower sets from singletons, the $\sqsubseteq_{Sud}$ relation so subscripted and restricted to half the strength of $\sqsubseteq_{EM}$, and a similar subscript on $\lessdot$:

Definition: $\phi \lessdot_{Sud} \pi$ if for all $\gamma \in G$ ($\downarrow\{\phi\gamma\} \sqsubseteq_{Sud} \pi\gamma$).

In the Smyth-upside-down domain these definitions are implicit; the notation is used here for emphasis. Under this domain the standing definition of ψ is interpreted using $\lessdot_{Sud}$ in place of $\lessdot$ and is, therefore written as $\tilde{\psi}$.

Lemma 6: In the Smyth-upside-down domain, let $\{\pi_i\}_{i\in\omega}$ satisfy $\forall i(\pi_i \sqsubseteq_{Sud} \pi_{i+1})$.
If $\gamma = \mathbf{fix}\ \phi$ for $\phi \leq_{Sud} \sqcup_i \pi_i$ then there is a sequence $\{\gamma_i\}_{i\in\omega}$ such that $\gamma_0 = \perp_G$, $\gamma = \sqcup_i \gamma_i$, and for all i: $\gamma_i \sqsubseteq \gamma_{i+1}$, γ_i is isolated, and $\downarrow\{\gamma_{i+1}\} \sqsubseteq_{Sud} \pi_i\gamma_i$.

Proof: Let $\pi = \sqcup_i \pi_i$; thus $\phi \leq_{Sud} \pi$.
Because $\gamma = \mathbf{fix}\ \phi$, Lemma 5 shows that there exists $\{\hat{\gamma}_j, \hat{\pi}_j\}_{j\in\omega}$ satisfying
$\hat{\gamma}_0 = \perp_G$, $\sqcup_j \hat{\gamma}_j = \gamma$, $\sqcup_j \hat{\pi}_j = \pi$, and for all j:
$\hat{\gamma}_j \sqsubseteq \hat{\gamma}_{j+1}$, $\hat{\pi}_j \sqsubseteq_{Sud} \hat{\pi}_{j+1}$, $\hat{\gamma}_j$ and $\hat{\pi}_j$ are isolated, and $\downarrow\{\hat{\gamma}_{j+1}\} \sqsubseteq_{Sud} \hat{\pi}_j\hat{\gamma}_j$.

Because each $\hat{\pi}_j$ is isolated, for every j there exists a k_j such that $\hat{\pi}_j \sqsubseteq_{Sud} \pi_{k_j}$.
Define $\gamma_i = \hat{\gamma}_{n(i)}$ for all i, where n is a function on the integers defined by:

$n(0) = 0$;
$n(i+1) = n(i)$ when $k_{n(i)} > i$;
$n(i+1) = n(i)+1$ when $k_{n(i)} \leq i$.

The lemma is proved point-by-point from this definition.

- $\gamma_0 = \hat{\gamma}_0 = \perp_G$ by definition.
- $\sqcup_i \gamma_i = \sqcup_j \hat{\gamma}_j = \gamma$ because $\forall i(\exists j(\gamma_i = \hat{\gamma}_j))$ and $\forall j(\exists i(\gamma_i = \hat{\gamma}_j))$.
  The first claim follows from the definition of $\{\gamma_i\}$;
  the second is a consequence of the existence of k_j .
- $\gamma_i \sqsubseteq \gamma_{i+1}$ since either $\gamma_i = \gamma_{i+1}$ (when $n(i+1)=n(i)$) or
  $\gamma_i = \hat{\gamma}_{n(i)} \sqsubseteq \hat{\gamma}_{n(i)+1} = \gamma_{i+1}$ (when $n(i+1) = n(i)+1$) .
- Every γ_i is isolated because it coincides with some $\hat{\gamma}_j$, each of which is isolated.
- We must argue inductively that $\downarrow\{\gamma_{i+1}\} \sqsubseteq_{Sud} \pi_i\gamma_i$.
  If $n(1)=0$ the basis is trivial. Otherwise, if $n(1)=1$, then $k(0)=0$ and so $\hat{\pi}_0 \sqsubseteq_{Sud} \pi_0$.
  Then $\downarrow\{\gamma_1\} = \downarrow\{\hat{\gamma}_1\} \sqsubseteq_{Sud} \hat{\pi}_0\hat{\gamma}_0 = \hat{\pi}_0\gamma_0 \sqsubseteq_{Sud} \pi_0\gamma_0$.
  Assume this point for i and consider the successor case:
  Either $n(i+2)=n(i+1)$ and $\downarrow\{\gamma_{i+2}\} = \downarrow\{\gamma_{i+1}\} \sqsubseteq_{Sud} \pi_i\gamma_i \sqsubseteq_{Sud} \pi_{i+1}\gamma_{i+1}$
  (by inductive hypothesis and monotonicity),
  or $n(i+2) = n(i+1)+1 =$ and $k_j \leq i+1$, so
  $\downarrow\{\gamma_{i+2}\} = \downarrow\{\hat{\gamma}_{j+1}\} \sqsubseteq_{Sud} \hat{\pi}_j\hat{\gamma}_j = \hat{\pi}_j\gamma_{i+1} \sqsubseteq_{Sud} \pi_{k_j}\gamma_{i+1} \sqsubseteq_{Sud} \pi_{i+1}\gamma_{i+1}$.

□

Lemma 7: In the Smyth-upside-down domain, if $\{\gamma_i,\pi_i\}_{i\in\omega}$ satisfies $\gamma_0=\bot_G$, and for all i:
$\gamma_i \sqsubseteq \gamma_{i+1}$, γ_i is isolated, $\pi_i \sqsubseteq_{Sud} \pi_{i+1}$, and $\downarrow\{\gamma_{i+1}\} \sqsubseteq_{Sud} \pi_i\gamma_i$,
then $\sqcup_i\gamma_i = \mathbf{fix}\ \phi$ for $\phi \leqslant_{Sud} \sqcup_i\pi_i$.

Proof: Let $\pi=\sqcup_i\pi_i$, and let $\{\phi_i\}_{i\in\omega}$ be defined from $\{\gamma_i\}_{i\in\omega}$ as follows:
$\phi_0 = \bot_F = \lambda\gamma.\bot_G$;
$\phi_{i+1} = \lambda\gamma.(\gamma_i \sqsubseteq \gamma \rightarrow \gamma_{i+1}, \phi_i\gamma)$.
We shall show that, for all i,
$\phi_i \sqsubseteq \phi_{i+1}$, $\phi_{i+1} \leqslant_{Sud} \pi_i$, ϕ_i is monotone and continuous, and $\mathbf{fix}\ \phi_i = \gamma_i$.
It then follows, taking the join, that $\phi = \sqcup_i\phi_i \leqslant_{Sud} \sqcup_i\pi_i = \pi$ such that
$\mathbf{fix}\ \phi = \mathbf{fix}(\sqcup_i\phi_i) = \sqcup_i(\mathbf{fix}\ \phi_i) = \sqcup_i\gamma_i$.

The properties of $\{\phi_i\}_{i\in\omega}$ follow by induction. ϕ_0 trivially satisfies all requirements.
$\phi_1 \leqslant_{Sud} \pi_0$, because for all γ, $\downarrow\{\phi_1\gamma\} = \downarrow\{\gamma_1\} \sqsubseteq_{Sud} \pi_0\gamma_0 = \pi_0\bot_G \sqsubseteq_{Sud} \pi_0\gamma$.
Assume that ϕ_i satisfies the other requirements, as well, and inductively consider ϕ_{i+1} .

- $\phi_i \sqsubseteq \phi_{i+1} \leqslant_{Sud} \pi_i$. For every γ, either $\gamma_i \sqsubseteq \gamma$ or not.
  In the case that $\gamma_i \sqsubseteq \gamma$ we find that $\phi_i\gamma = \gamma_i \sqsubseteq \gamma_{i+1} = \phi_{i+1}\gamma$.
  Moreover, $\downarrow\{\phi_{i+1}\gamma\} = \downarrow\{\gamma_{i+1}\} \sqsubseteq_{Sud} \pi_i\gamma_i \sqsubseteq_{Sud} \pi_i\gamma$ since π_i is monotone.
  In the other case that $\gamma_i \not\sqsubseteq \gamma$, we find $\phi_i\gamma = \phi_{i+1}\gamma$ by definition,
  and $\downarrow\{\phi_{i+1}\gamma\} = \downarrow\{\phi_i\gamma\} \sqsubseteq_{Sud} \pi_{i-1}\gamma \sqsubseteq_{Sud} \pi_i\gamma$ by induction :
  $\phi_i \leqslant_{Sud} \pi_{i-1} \sqsubseteq_{Sud} \pi_i$.
  So in either case $\phi_i\gamma \sqsubseteq \phi_{i+1}\gamma$ and $\downarrow\{\phi_{i+1}\gamma\} \sqsubseteq_{Sud} \pi_i\gamma$, establishing this point.
- ϕ_{i+1} is monotone, because if $\hat\gamma_1 \sqsubseteq \hat\gamma_2$ then there are two cases.
  If $\gamma_i \sqsubseteq \hat\gamma_1$ then $\gamma_i \sqsubseteq \hat\gamma_2$ as well, and $\phi_{i+1}\hat\gamma_1 = \gamma_{i+1} = \phi_{i+1}\hat\gamma_2$.
  If $\gamma_i \not\sqsubseteq \hat\gamma_1$ then $\phi_{i+1}\hat\gamma_1 = \phi_i\hat\gamma_1 \sqsubseteq \phi_i\hat\gamma_2 \sqsubseteq \phi_{i+1}\hat\gamma_2$ by induction and because $\phi_i \sqsubseteq \phi_{i+1}$.
- ϕ_{i+1} is continuous.
  Let $\hat\gamma = \sqcup_j\hat\gamma_j$. Monotonicity of ϕ_{i+1} shows that $\sqcup_j(\phi_{i+1}\hat\gamma_j) \sqsubseteq \phi_{i+1}(\sqcup_j\hat\gamma_j)$.
  We need to show that $\phi_{i+1}(\sqcup_j\hat\gamma_j) \sqsubseteq \sqcup_j(\phi_{i+1}\hat\gamma_j)$; again there are two cases: $\gamma_i \sqsubseteq \hat\gamma$ and $\gamma_i \not\sqsubseteq \hat\gamma$.
  If $\gamma_i \sqsubseteq \hat\gamma$ then, because γ_i is isolated, there is some k such that $\gamma_i \sqsubseteq \hat\gamma_k$.
  Thus, $\phi_{i+1}(\sqcup_j\hat\gamma_j) = \gamma_{i+1} = \phi_{i+1}\hat\gamma_k \sqsubseteq \sqcup_j(\phi_{i+1}\hat\gamma_j)$.
  If $\gamma_i \not\sqsubseteq \hat\gamma$ then, because $\forall\, j(\gamma_i \not\sqsubseteq \hat\gamma_j)$ and because ϕ_i is continuous by induction,
  $\phi_{i+1}(\sqcup_j\hat\gamma_j) = \phi_i(\sqcup_j\hat\gamma_j) = \sqcup_j(\phi_i\hat\gamma_j) = \sqcup_j(\phi_{i+1}\hat\gamma_j)$.
- $\mathbf{fix}\ \phi_{i+1} = \gamma_{i+1}$.
  Since ϕ_{i+1} is monotone and continuous, let $\gamma = \mathbf{fix}\ \phi_{i+1}$.
  In the case that $\gamma_i \sqsubseteq \gamma$ we find that $\gamma_i \sqsubseteq \gamma = \phi_{i+1}\gamma = \gamma_{i+1}$.
  If, on the other hand, $\gamma_i \not\sqsubseteq \gamma$ then by induction, $\mathbf{fix}\ \phi_i = \gamma_i \not\sqsubseteq \gamma = \mathbf{fix}\ \phi_{i+1}$.
  But the latter conclusion contradicts $\phi_i \sqsubseteq \phi_{i+1}$ (implying $\mathbf{fix}\ \phi_i \sqsubseteq \mathbf{fix}\ \phi_{i+1}$),
  and so we reject the latter and accept the former case, that $\gamma = \gamma_{i+1}$.

□

Theorem 5: Let $\pi = \sqcup_i \pi_i$ for $\{\pi_i\}_{i\in\omega}$ ascending: $\forall i(\pi_i \sqsubseteq_{Sud} \pi_{i+1})$.
Then $\gamma = \mathbf{fix}\ \phi$ for $\phi \lessdot_{Sud} \pi$, if and only if there exists a sequence $\{\gamma_i\}_{i\in\omega}$ such that $\gamma_0 = \perp_G$, $\sqcup_i \gamma_i = \gamma$, and for all $i \in \omega$, $\gamma_i \sqsubseteq \gamma_{i+1}$, γ_i is isolated, and $\downarrow\{\gamma_{i+1}\} \sqsubseteq_{Sud} \pi_i \gamma_i$.

Proof: Lemma 6 shows sufficiency; Lemma 7 shows necessity. □

4.2. A counterexample

It is not necessarily the case that all γ such that $\gamma \in \downarrow\Psi\pi$ (equivalently $\{\gamma\} \sqsubseteq_{Sud} \Psi\pi$) can be described as in Theorem 5. That is, some γ are introduced from fixed-points of $\phi \lessdot_{Sud} \pi$ because of closure requirements in the domain $\mathbf{P}_{Sud}(G)$; an example follows. After it we consider the procedure for enumerating sets of sequences approaching a desired set in $\mathbf{P}_{Sud}(G)$.

Example 1 (Clinger):
Let G be the conventional Boolean lattice. Define $\pi \in [G \to \mathbf{P}_{Sud}(G)]$ by the following:

π: $\perp \vdash \{\text{TRUE}\}$;
TRUE $\vdash \{\top\}$;
FALSE $\vdash \{\top\}$;
$\top \vdash \{\top\}$;

Let us consider the skeletons of candidate $\phi \lessdot_{Sud} \pi$:

ϕ_0: $\perp \vdash \perp$.

ϕ_1: $\perp \vdash$ TRUE ;
TRUE $\vdash$ TRUE ;

ϕ_2: $\perp \vdash$ TRUE ;
TRUE $\vdash \top$;
$\top \vdash \top$.

These skeletons are sufficient to determine that
$\mathbf{fix}\ \phi_0 = \perp$; $\mathbf{fix}\ \phi_1 = \text{TRUE}$; $\mathbf{fix}\ \phi_2 = \top$.
Moreover, they exhaust all the possible fixed points for $\phi \lessdot_{Sud} \pi$. Thus,
$\Psi\pi = \{\perp, \text{TRUE}, \top\} = \{\perp, \text{TRUE}, \text{FALSE}, \top\} =_{Sud} \{\top\}$
because of necessary closures in $\mathbf{P}_{Sud}(G)$.

The point of this example is that FALSE is included in $\Psi\pi$ because of downward closures in the Smyth-upside-down domain, even though it is the fixed point of no $\phi \lessdot_{Sud} \pi$!

Theorem 6: $\downarrow\{\gamma\} \mathbin{\mathbb{\sqsubseteq}}_{Sud} \Psi\pi$ if and only if there is some $\phi \leq_{Sud} \pi$ such that $\gamma \mathbin{\mathbb{\sqsubseteq}} \mathbf{fix}\ \phi = \sqcup_i(\phi^i\gamma)$.

Proof: Necessity: If $\exists \phi \leq \pi (\gamma \mathbin{\mathbb{\sqsubseteq}} (\mathbf{fix}\ \phi))$ then
$\{\gamma\} \mathbin{\mathbb{\sqsubseteq}} \{\mathbf{fix}\ \phi\} \mathbin{\mathbb{\sqsubseteq}} \Psi\pi$ since $\phi \leq \pi$. Hence $\{\gamma\} \mathbin{\mathbb{\sqsubseteq}} \Psi\pi$.

Sufficiency: If $\downarrow\{\gamma\} \mathbin{\mathbb{\sqsubseteq}}_{Sud} \Psi\pi$, then $\exists \hat{\gamma} \in \Psi\pi (\gamma \mathbin{\mathbb{\sqsubseteq}} \hat{\gamma})$.
Then let $\{\hat{\gamma}_j\}_{j\in\omega}$ be an ascending chain of isolated points approximating $\hat{\gamma}$: $\sqcup_j \hat{\gamma}_j = \hat{\gamma}$ and
$\forall j \in \omega (\hat{\gamma}_j \sqsubseteq \hat{\gamma}_{j+1}$ & $\hat{\gamma}_j$ is isolated & $\exists \hat{\phi}_j \leq_{Sud} \pi (\hat{\gamma}_j = \mathbf{fix}\ \hat{\phi}_j))$.
Because γ is way below $\sqcup_j \hat{\gamma}_j$, there is some j such that $\gamma \sqsubseteq \hat{\gamma}_j$;
because that $\hat{\gamma}_j$ is isolated, $\gamma \sqsubseteq \hat{\gamma}_j \mathbin{\mathbb{\sqsubseteq}} \hat{\gamma}_j$ and:

$$\gamma \mathbin{\mathbb{\sqsubseteq}} \mathbf{fix}\hat{\phi}_j = \sqcup_i \hat{\phi}_j^i \perp_G \sqsubseteq \sqcup_i \hat{\phi}_j^i \gamma \sqsubseteq \sqcup_i \hat{\phi}_j^i \hat{\gamma}_j = \sqcup_i (\mathbf{fix}\ \hat{\phi}_j) = \mathbf{fix}\ \hat{\phi}_j .$$

□

In cases like Example 1, where γ might be in $\Psi\pi$ because of completion rather than because γ is a specific fixed-point of some $\phi \leq_{Sud} \pi$, Theorem 6 shows that such a legitimate fixed-point above γ may be generated from γ by the same means that it is generated from $\perp$. All that is necessary is that the γ in question be "well removed" from those boundary points specifically introduced by completion. This is satisfied when an isolated point separates γ from that boundary, and is trivially satisfied when γ, itself, is isolated. This latter observation includes all the cases in which we are immediately interested, because the 'computable' ground values are, themselves, isolated. Those are the finite (prefices to infinite) computations that can only output isolated elements.

4.3. Proof Rule

Corollary 1: Let $\pi = \sqcup_i \pi_i$ for $\{\pi_i\}_{i\in\omega}$ ascending: $\forall i (\pi_i \sqsubseteq_{Sud} \pi_{i+1})$.
Then $\downarrow\{\gamma\} \mathbin{\mathbb{\sqsubseteq}}_{Sud} \Psi\pi$ if and only if there exists a sequence
$\{\gamma_i\}_{i\in\omega}$ such that $\gamma_0 = \perp_G$, $\gamma \sqsubseteq \sqcup_i \gamma_i$, $\sqcup_i \pi_i = \pi$, and for all i:
$\gamma_i \sqsubseteq \gamma_{i+1}$, γ_i is isolated, and $\downarrow\{\gamma_{i+1}\} \sqsubseteq_{Sud} \pi_i \gamma_i$.

Proof: Theorem 5 and Theorem 6. □

This is the corollary that yields the **operational** semantics for Ψ . Given a program π , or perhaps a sequence $\{\pi_i\}_{i\in\omega}$ describing π, we can generate sequences $\{\gamma_i\}_{i\in\omega}$ that reach or exceed all γ such that $\downarrow\{\gamma\} \mathbin{\mathbb{\sqsubseteq}}_{Sud} \Psi\pi$ in their limits. Thus, every value in $\Psi\pi$ well removed from its boundary will be either reached or exceeded by one of the sequences that we generate by the repeated iteration of

$$\downarrow\{\gamma_{i+1}\} \sqsubseteq_{Sud} \pi_i \gamma_i \sqsubseteq_{Sud} \eta(i+1)\pi\perp .$$

A different perspective is available from Corollary 1 if we strengthen the fundamental theory of comains. The following commonly accepted axiom has not yet been used here.

Axiom 1: If γ is isolated in **G**, then the cardinality of $\downarrow\{\gamma\}$ **in** $\boldsymbol{P}$**(G)** is properly less than ω.

Axiom 1 implies that there are no isolated elements in G above any non-isolated element. Similarly, there can be no non-isolated element properly above another non-isolated element in . That is, the non-isolated elements (if there are any) are supreme in .

Then, the discussion following Theorem 6 (justifying the use of $\mathbin{\mathbb{\sqsubseteq}}$ rather than $\sqsubseteq$ there) is mooted, and so Corollary 1 might be stated in a weaker form using $\sqsubseteq$. That is, $\downarrow\{\phi\} \sqsubseteq_{Sud} \Psi\pi$

then implies either that $\gamma \in \Downarrow\pi$ or that γ is isolated and so Theorem 6 applies; in either case the sequence described in Corollary 1 exists.

Corollary 2: Let $\pi = \sqcup_i \pi_i$ for $\{\pi_i\}_{i\in\omega}$ ascending: $\forall i(\pi_i \sqsubseteq_{Sud} \pi_{i+1})$.
Then $\downarrow\{\gamma\} \sqsubseteq_{Sud} \Downarrow\pi$ if and only if there exists a sequence
$\{\gamma_i\}_{i\in\omega}$ such that $\gamma_0 = \perp_G$, $\gamma \sqsubseteq \sqcup_i \gamma_i$, $\sqcup_i \pi_i = \pi$, and for all i:
$\gamma_i \sqsubseteq \gamma_{i+1}$, γ_i is isolated, and $\downarrow\{\gamma_{i+1}\} \sqsubseteq_{Sud} \pi_i\gamma_i$.

Proof: Corollary 1, tempered by Axiom 1. □

The proof of Corollary 2 is wholly constructive and so the more convincing. Is such a result available under the Plotkin power domain? I suggest that the answer is affirmative below, but it seems clear from the proofs above (*e.g.* of Theorem 2) that the proof is not constructive. The requisite sequences $\{\phi_i\}_{i\in\omega}$ would be above those constructed before (*e.g.* in the proof of Lemma 7), and little more than an existence claim for $\phi_i \lessdot \pi_i$, instead of $\phi_i \lessdot_{Sud} \pi_i$, is needed to strengthen those proofs.

Conjecture 3: In the Egli-Milner order, let $\pi = \sqcup_i \pi_i$ for $\{\pi_i\}_{i\in\omega}$ ascending: $\forall i(\pi_i \sqsubseteq_{EM} \pi_{i+1})$.
Then $\gamma \in \psi\pi$ if and only if there exists a sequence
$\{\gamma_i\}_{i\in\omega}$ such that $\gamma_0 = \perp_G$, $\gamma = \sqcup_i \gamma_i$, $\sqcup_i \pi_i = \pi$, and for all i:
$\gamma_i \sqsubseteq \gamma_{i+1}$, γ_i is isolated, and $\gamma_{i+1} \in \pi_i\gamma_i$.

From Corollary 2 (and Conjecture 3) we infer both the operational semantics and the proof rule. *It is most important that no $\phi \in \mathbf{F}$ appear in either.*

Proof rule for $\mathbb{P}_{EM}(\mathbf{G})$: $\gamma \in \psi\pi$ if and only if $\gamma \in \pi\gamma$
and anything in $\pi\gamma$ above γ is also in $\psi\pi$.

For $\mathbb{P}_{EM}(\mathbf{G})$ the proof rule need not apply to *every* γ in $\psi\pi$, just so every γ is bracketed (above and beneath) by others satisfying the rule. For $\mathbb{P}_{Sud}(\mathbf{G})$ the rule need only apply to an upper bound, as indicated by $\gamma \sqsubseteq \sqcup_i \gamma_i$ in Corallaries 1 and 2. $\psi\pi$ is the least set (with respect to $\sqsubseteq_{\mathbb{P}(\mathbf{G})}$) that satisfies the proof rule.

There is a temptation to refine this result so that only the upper frontier, the supremum of $\psi\pi$ (as defined), becomes the semantics of $\psi\pi$. Unfortunately, there seems to be no operational semantics that nicely excludes inferior non-isolated points, and so the present definition stands. When the inferior points turn out to be isolated, a decent implementation might opt for one of those above; an instance of this appears in Section 7.2[5].

4.4. Procedures to compute $\psi\pi$

In the following, we shall present examples run through procedures also derived from these corollaries. The proofs of their correctness thereby becomes trivial, since the proof rule derives from the same source. It appears that the procedures are tailored after the proof rule or vice versa.

The procedures are presented generically, for either $\mathbb{P}_{EM}(\mathbf{G})$ or $\mathbb{P}_{Sud}(\mathbf{G})$. The first procedure is designed to find just one $\gamma \in \psi\pi$. Ordinarily σ, and hence π, is defined by some recursive program, that specifies π by some approximating sequence $\{\pi_i\}_{i\in\omega}$. So long as only maximal γ_{i+1} are chosen at each step, it is easy to see that the computation of a single sequence $\{\gamma_i\}$ generates a tree of finite degree at every node. The procedure follows:

---

[5]Excluding detectably inferior γ in $\psi\gamma$ is harmless, unless a strong definition of "fairness" is breached thereby. Otherwise, it just provides "more" meaning from evaluation.

Procedure 1: Compute $\gamma = \sqcup_i \gamma_i \in \psi(\sqcup_i \pi_i)$ where each π_i is isolated.

1. Set $\gamma_0 := \perp$. Set i:=0 .
2. Compute $\pi_i \gamma_i$, representable as a finite set of isolated representative elements because both π_i and γ_i are isolated.
3. Study each element $\gamma \in \pi_i \gamma_i$ and cull it unless $\gamma_i \sqsubset \gamma$.
   (While $\gamma_i \in \pi_i \gamma_i$, it is still culled because the relation is proper!)
   If there is nothing left then stop; γ_i is the isolated limit.
4. Choose the new, isolated $\gamma_{i+1} = \gamma$ surviving Step 3.
5. Increment i, and go to Step 2.

□

In this way, the corollaries guarantee that we will build a sequence that approaches or reaches a least fixed point that is a supremum of $\psi\pi$ in $\boldsymbol{P}_{EM}(G)$, or in $\boldsymbol{P}_{Sud}(G)$ a representative element in the sense of Clinger [3]. (Were fairness at issue, the exclusion of values $\gamma_{i+1} \sqsubset \gamma$ and the strategy for choice among candidate values at Step 4 would be important.)

Suppose that we want to compute, or to compute a sequence approaching, that entire element in the power domain. Such a computation is similar to a breadth-first expansion of the computation tree suggested by the formula describing a path of length i:

$$\eta i \pi \perp_G .$$

Two procedures are presented: Procedure 2 is "brute force", but Procedure 3 is more sparing of computational effort and will be used in the examples below.

Procedure 2: Compute $\epsilon = \psi(\sqcup_i \pi_i)$.

1. Set $\epsilon = \perp_{\boldsymbol{P}_{EM}(G)}$ **in E** .
2. For i := 0 to ω compute $\eta(i+1)\pi_i \perp_G$, while
3. joining ({γ} **in E**) into ϵ for any γ satisfying
   $\gamma \sqsubseteq \phi^i \perp_G$ and $\gamma \sqsubseteq \phi\gamma$ for $\phi \lessdot \pi_i$.

□

Again the corollaries guarantee that Procedure 2 will generate in its limit any $\gamma \in \psi\pi$. Procedure 3 uses an indexing scheme to avoid repeatedly rebuilding the same parts of the computation tree.

Procedure 3: Compute $\epsilon = \psi(\sqcup_i \pi_i)$ where each π_i is isolated.

1. Set j:=0; define $\gamma_\Lambda = \perp_G$.
2. For $i \in \omega$ (according to some effective enumeration strategy†) compute
   $\pi_i \gamma_x$ where $x \in \omega^*$ and γ_x is already defined.
   This set is finitely representable since γ_x and π_i are isolated.
3. and for every† isolated γ such that $\{\gamma\} \sqsubseteq \pi_i \gamma_x$,
   if $\gamma_x \sqsubset \gamma$ then define $\gamma_{x,j} = \gamma$ and increment j .
   If no such γ exist then γ_x *is* a representative limit point of the tree.

□

In the examples below, γ satisfying the test at Step 3 will be displayed plainly, but those not satisfying it will be struck through and referred to as "culls". Thus, a representative point γ_x is found when everything is culled at Step 3.

A sequence of the sort described in Corollary 2 may be extracted from successive finite prefixes of the subscript, x, of a defined γ_x . Thus, once defined, the value of γ_x will be in $\psi\pi$, although perhaps not representative of it. Those generating all culls at Step 3 are representative elements, as are the limits of infinite ascending sequences whose subscripts are prefixes of one another.

5. Semantics for sets of triples

In this section the ground domain, G, is fleshed out to include triples and six atoms. This is done surgically in order to consider the particular problem introduced in Section 1.3. In addition it presents a polished **letrec** semantics and hints at the way that general data structures are to be introduced in the following section. Indeterminism is introduced with McCarthy's *amb* operator.

In this section the down arrow, $\downarrow$, is used as infix notation to indicate projection from product domains in the standard manner. The example will be treated in $\mathcal{P}_{Sud}(G)$, based on the results of Corollary 2.

MacQueen poses the problem of deriving "suitable" semantics for evaluation of an expression of the form

(letrec I {1 2 (if (first I)=1 then 3 else 4)})

where the braces here indicate all permutations of the enclosed items. Section 1.3 states the problem and should be reviewed here under the interpretation of $\mathcal{P}_{Sud}(G)$.

Once the reader is reminded that $\perp$ is a legitimate value, she will, no doubt, suggest other possible values besides those in Section 1.3, like $(\perp\ 2\ \perp)$. While these are certainly correct, each is beneath ($\sqsubseteq$) one of the acceptable permutations already mentioned above. The definition of the Smyth-upside-down power domain that is being used, indicates that the *intended* value of evaluating the above expression is the element

{ (1 2 3), (1 3 2), (2 1 4), (2 4 1), $(\perp$ 1 2), $(\perp$ 2 1) },

a maximal representative of the partition block (mod $\sqsubseteq_{\mathcal{P}_{Sud}(G)}$) in the power domain that we want. It includes all values beneath the six permutations listed within the braces.

5.1. Semantics

Much of the following appears in a previous section. The **letrec** line of the semantics, however, now directs evaluation to a constant fixed-point function. Now that ψ has been shown to be as "safe" as **fix**, its effect need no longer be so explicit in the language definition.

The richer part of this language occurs in the constant definitions, which are at last presented completely.

5.1.1. Syntax

K :: fix | amb | if | first | second | third | equal? | true | false | 1 | 2 | 3 | 4;
E :: I | K | $<E_1\ E_2\ E_3>$ | $(E_0\ E_1)$ | **(lambda** I E) | **(letrec** I E)

5.1.2.
Value Domains

$$\rho \in U = [Ide \to D] \qquad \text{(environments)}$$

$$\delta \in D = G \boxplus S \qquad \text{(denoted values)}$$
$$\epsilon \in E = \mathcal{P}_{Sud}(G) \boxplus S \qquad \text{(expressed values)}$$
$$\sigma \in S = [D \to E] \qquad \text{(systems)}$$

$$\pi \in P = [G \to \mathcal{P}_{Sud}(G)] \qquad \text{(programs)}$$
$$\phi \in F = [G \to G] \qquad \text{(functions)}$$

$$\gamma \in G = A + T \qquad \text{(ground values)}$$
$$\tau \in T = G \times G \times G \qquad \text{(triples)}$$
$$\alpha \in A = \{\textbf{true}, \textbf{false}, 1, 2, 3, 4\} \qquad \text{(atoms)}$$

5.1.3. Semantic Functions

$$\mathcal{E} : [Exp \to [U \to E]]$$

$$\mathcal{E}[\![I]\!]\rho \equiv \nabla(\rho[\![I]\!]) \qquad \text{(parameter)}$$
$$\mathcal{E}[\![K]\!]\rho \equiv \nabla(\mathcal{K}[\![K]\!]) \qquad \text{(constant)}$$

$$\mathcal{E}[\![(\textbf{lambda } I\ E)]\!]\rho \equiv (\lambda\delta.\mathcal{E}[\![E]\!]\rho[\delta/I]) \textbf{ in } E \qquad \text{(function)}$$

$$\mathcal{E}[\![<E_1\ E_2\ E_3>]\!]\rho \equiv \left\{(\gamma_1, \gamma_2, \gamma_3) \textbf{ in } G \;\middle|\; \gamma_i \in ((\mathcal{E}[\![E_i]\!]\rho)\,|\mathcal{P}_{Sud}(G))\right\} \textbf{ in } E \qquad \text{(triple)}$$

$$\mathcal{E}[\![(E_0\ E_1)]\!]\rho \equiv (\text{Let } \sigma = ((\mathcal{E}[\![E_0]\!]\rho)|S) \text{ and } \epsilon = (\mathcal{E}[\![E_1]\!]\rho) \text{ in}$$
$$(\epsilon \in \mathcal{P}_{Sud}(G) \to \bigsqcup\{\sigma(\gamma \textbf{ in } D) \mid \gamma \in (\epsilon|\mathcal{P}_{Sud}(G))\},$$
$$(\epsilon \in S \to \sigma(\epsilon|S \textbf{ in } D),$$
$$(\epsilon = \top_E \to \sigma\top_D,$$
$$\sigma\bot_D))) \qquad \text{(application)}$$

$$\mathcal{E}[\![(\textbf{letrec } I\ E)]\!]\rho \equiv \mathcal{E}[\![(\textbf{fix } (\textbf{lambda } I\ E))]\!]\rho \qquad \text{(recursion†)}$$

Proposition†: The last of the above semantic equations is superfluous,

because of $\mathcal{K}[\![\text{fix}]\!]$.

$$
\begin{aligned}
\boldsymbol{K} &: [\mathbf{Con} \rightarrow \mathbf{D}] \\
\boldsymbol{K}\,[\![\text{fix}]\!] &\equiv (\lambda\delta.\,(\text{Let } \sigma = \delta|\mathbf{S} \text{ in} \\
&\qquad (\sigma\bot_{\mathbf{D}} \sqsubseteq \boldsymbol{P}_{\text{Sud}}(\mathbf{G}) \rightarrow \psi(\lambda\gamma.(\sigma(\gamma \textbf{ in } \mathbf{D}) \mid \boldsymbol{P}_{\text{Sud}}(\mathbf{G}))) \textbf{ in } \mathbf{E}\,, \\
&\qquad (\sigma\bot_{\mathbf{D}} \sqsubseteq \mathbf{S} \rightarrow \textbf{fix } \lambda\epsilon.(\sigma(\epsilon|\mathbf{S} \textbf{ in } \mathbf{D})), \\
&\qquad (\sigma\bot_{\mathbf{D}} = \top_{\mathbf{E}} \rightarrow \top_{\mathbf{E}}\,, \\
&\qquad \bot_{\mathbf{E}}\,)))\,))\textbf{ in } \mathbf{D}\,; && (\dagger) \\
\boldsymbol{K}\,[\![\text{amb}]\!] &\equiv (\lambda\delta.(\text{ Let } (\gamma_1,\gamma_2,\gamma_3) = \delta|\mathbf{G}|\mathbf{T} \text{ in} \\
&\qquad \{\gamma_1, \gamma_2, \gamma_3\} \textbf{ in } \mathbf{E}))\ \textbf{ in } \mathbf{D}\,; && [14] \\
\boldsymbol{K}\,[\![\text{if}]\!] &\equiv (\lambda\delta.\,(\{\text{ Let } \tau=(\delta|\mathbf{G}|\mathbf{T}) \text{ in} \\
&\qquad (\tau\downarrow 1|\mathbf{A} = \textbf{true} \rightarrow \tau\downarrow 2, \\
&\qquad (\tau\downarrow 1|\mathbf{A} = \textbf{false} \rightarrow \tau\downarrow 3, \\
&\qquad \bot_{\mathbf{G}}))\qquad\qquad \}\textbf{ in } \mathbf{E})\,)\textbf{ in } \mathbf{D}\,; \\
\boldsymbol{K}\,[\![\text{first}]\!] &\equiv (\lambda\delta.(\,\{\,((\delta|\mathbf{G}|\mathbf{T}) \downarrow 1)\,\}\textbf{ in } \mathbf{E}\,))\textbf{ in } \mathbf{D}\,; \\
\boldsymbol{K}\,[\![\text{second}]\!] &\equiv (\lambda\delta.(\,\{\,((\delta|\mathbf{G}|\mathbf{T}) \downarrow 2)\,\}\textbf{ in } \mathbf{E}\,))\textbf{ in } \mathbf{D}\,; \\
\boldsymbol{K}\,[\![\text{third}]\!] &\equiv (\lambda\delta.(\,\{\,((\delta|\mathbf{G}|\mathbf{T}) \downarrow 3)\,\}\textbf{ in } \mathbf{E}\,))\textbf{ in } \mathbf{D}\,; \\
\boldsymbol{K}\,[\![\text{equal?}]\!] &\equiv (\lambda\delta.\,(\{\text{ Let } \tau=(\delta|\mathbf{G}|\mathbf{T}) \text{ in} \\
&\qquad ((\tau\downarrow 1|\mathbf{A})=\bot_{\mathbf{A}} \rightarrow \bot_{\mathbf{A}}, \\
&\qquad ((\tau\downarrow 2|\mathbf{A})=\bot_{\mathbf{A}} \rightarrow \bot_{\mathbf{A}}, \\
&\qquad ((\tau\downarrow 1|\mathbf{A})=(\tau\downarrow 2|\mathbf{A}) \rightarrow \textbf{true}, \\
&\qquad \textbf{false})))\qquad\qquad \textbf{in } \mathbf{G}\,\}\textbf{ in } \mathbf{E}))\textbf{ in } \mathbf{D}\,; \\
\boldsymbol{K}\,[\![\text{true}]\!] &\equiv \textbf{true in G in D}\,; \\
\boldsymbol{K}\,[\![\text{false}]\!] &\equiv \textbf{false in G in D}\,; \\
\boldsymbol{K}\,[\![1]\!] &\equiv \textbf{1 in G in D}\,; \\
\boldsymbol{K}\,[\![2]\!] &\equiv \textbf{2 in G in D}\,; \\
\boldsymbol{K}\,[\![3]\!] &\equiv \textbf{3 in G in D}\,; \\
\boldsymbol{K}\,[\![4]\!] &\equiv \textbf{4 in G in D}\,.
\end{aligned}
$$

5.2. Solution to MacQueen's Problem

Since $S_{<1\ 2\ 3>}$ can be expressed as a set of triples, it can be represented in the domains defined above. Specifically we might represent it as ϵ:

$$
\begin{aligned}
S_{<1\,2\,3>} \simeq \{\ &(((\mathbf{1 \text{ in } G}), (\mathbf{2 \text{ in } G}), (\mathbf{3 \text{ in } G}))\ \mathbf{\text{in } G}), (((\mathbf{1 \text{ in } G}), (\mathbf{3 \text{ in } G}), (\mathbf{2 \text{ in } G}))\ \mathbf{\text{in } G}), \\
&(((\mathbf{2 \text{ in } G}), (\mathbf{1 \text{ in } G}), (\mathbf{3 \text{ in } G}))\ \mathbf{\text{in } G}), (((\mathbf{2 \text{ in } G}), (\mathbf{3 \text{ in } G}), (\mathbf{1 \text{ in } G}))\ \mathbf{\text{in } G}), \\
&(((\mathbf{3 \text{ in } G}), (\mathbf{1 \text{ in } G}), (\mathbf{2 \text{ in } G}))\ \mathbf{\text{in } G}), (((\mathbf{3 \text{ in } G}), (\mathbf{2 \text{ in } G}), (\mathbf{1 \text{ in } G}))\ \mathbf{\text{in } G})\,\} \\
&\mathbf{\text{in } E}\,; \\
= \epsilon = \{\ &(\mathbf{1\ 2\ 3}), (\mathbf{1\ 3\ 2}), (\mathbf{2\ 1\ 3}), (\mathbf{2\ 3\ 1}), (\mathbf{3\ 1\ 2}), (\mathbf{3\ 2\ 1})\,\}\ \mathbf{\text{in } E}
\end{aligned}
$$

where we have omitted internal commas and the tiresome occurrences of "**in G**" in the last line. Having understood that "**in G**" gets in the way, we shall elide its use later.

Now let us write a program in this language to generate $\epsilon \simeq S_{<1\,2\,3>}$ from $<1\ 2\ 3>$ using the constant function, *amb*.

```
(E0 <1 2 3>) where
E0 = (lambda T (amb <                                  <(first T) (second T) (third T)>
              (amb <<(first T) (second T) (third T)> <(first T) (third T) (second T)>
                                                      <(second T) (first T) (third T)>>)
              (amb <<(second T) (third T) (first T)> <(third T) (first T) (second T)>
                                                      <(third T) (second T) (first T)>>)>).
```

Extend this to a function akin to $\lambda x.S_{<1\,2\,x>}$; given an argument, **x**, it returns the set of all the permutations of **1**, **2**, and **x**:

E_1 = (**lambda** X (E_0 <1 2 X>)) where E_0 is as defined above.

$\mathcal{E}\ [\![E_1]\!]\rho = \lambda\delta.\mathcal{E}\ [\![(E_0 <1\ 2\ X>]\!]\rho[\delta/X]$;

$= \lambda\delta.(\{(1\ 2\ \delta|G)\ (1\ \delta|G\ 2)\ (2\ 1\ \delta|G)\ (2\ \delta|G\ 1)\ (\delta|G\ 1\ 2)\ (\delta|G\ 2\ 1)\}$ **in** E) .

Extension to $\lambda x S_{<1\ 2\ ((x=1)\to 3,4)>}$ is also easy; one would ordinarily expect the result of this function to be under $S_{<1\ 2\ 3>}$ or $S_{<1\ 2\ 4>}$, and certainly above $S_{<1\ 2\ \perp_A>}$. (In the code below there appears to be an extra argument for *equal?*; why?)

E_2 = (**lambda** X (E_1 (if <(equal? <X 1 1>) 3 4>))) where E_1 is defined as above.

$\mathcal{E}\ [\![E_2]\!]\rho = \lambda\delta.\mathcal{E}\ [\![(E_1$ (if ...))$]\!]\rho[\delta/X]$;

$= \lambda\delta.(\{(1\ 2\ C)\ (1\ C\ 2)\ (2\ 1\ C)\ (2\ C\ 1)\ (C\ 1\ 2)\ (C\ 2\ 1)\}$ **in** E) ,

where C is syntactically the conditional ($(\delta|G|A)=1\to 3,4$)**in** G .

Now suppose that we were passing a triple (of, say, integers) to this function instead of x, and retrieving the first item in the triple in place of x:

E_3 = (**lambda** T (E_2 (first T))) ;

= (**lambda** T (E_0 <1 2 (if<(equal?<(first T)1 1>) 3 4>) >)) ;

where E_0 and E_2 are defined as above.

$\mathcal{E}\ [\![E_3]\!]\perp_U$ = ($\lambda\delta.\mathcal{E}\ [\![(E_2$ (first T))$]\!]\rho[\delta/T]$)**in** E ;

= ($\lambda\delta.(\{(1\ 2\ C)\ (1\ C\ 2)\ (2\ 1\ C)\ (2\ C\ 1)\ (C\ 1\ 2)\ (C\ 2\ 1)\}$ **in** E))**in** E ,

where C is syntactically the conditional ($((\delta|G|T\downarrow 1)|A)=1\to 3,4$)**in** G .

Let us define σ as this last value projected into S . Again we expect the result of this function to be under either $S_{<1\ 2\ 3>}$ or $S_{<1\ 2\ 4>}$ but above $S_{<1\ 2\ \perp_A>}$.

Define σ = ($\mathcal{E}\ [\![E_3]\!]\perp_U$)|S as above and derive π from σ:

$\pi = \lambda\gamma.(\sigma(\gamma$ **in** D) | $\mathcal{P}_{Sud}(G))$;

$= \lambda\gamma.($ (($\mathcal{E}\ [\![E_3]\!]\perp_U$)($\gamma$ **in** D)) | $\mathcal{P}_{Sud}(G)$) ;

$= \lambda\gamma.(\{(1\ 2\ C)\ (1\ C\ 2)\ (2\ 1\ C)\ (2\ C\ 1)\ (C\ 1\ 2)\ (C\ 2\ 1)\}$,

where C is syntactically the conditional ($((\gamma|T\downarrow 1)|A)=1\to 3,4$) **in** G ;

$= \lambda\gamma.(S_{<1\ 2\ (((\text{first }\gamma|T)=1)\to 3,4)>}$ | $\mathcal{P}_{Sud}(G))$.

Enough is in hand to present the first solution to MacQueen's problem: we apply *fix* to E_3.

$\mathcal{E}\ [\![(\text{fix } E_3)]\!]\perp_U = \psi\pi$ **in** E because $\sigma\perp_D \in \mathcal{P}_{Sud}(G)$.

Two ways to figure $\psi\pi$ occur here. The difficult path is to follow the definition of ψ and to find all $\phi \leq \pi$ (or equivalently $\phi \leq \sigma$), and then find their respective least-fixed-points. This is not particularly easy (G is infinite so there are infinitely many candidate ϕ), and awfully inefficient because, after all, the ϕ are not of direct interest.

The tractable path is to use either procedure which finds all the least fixed points, only implicitly dealing with $\phi \in F$. Since π is isolated, choose Procedure 3 where all $\pi_i = \pi$. This particular example has only finitely-many and isolated least-fixed-points among the infinity of dominated functions, moreover, so the tree exploration will terminate cleanly when all branches of the tree cull out to leaves at Step 3.

ϵ_0 = $\perp_{\mathcal{P}_{Sud}(G)}$ **in** E = $\{\perp_G\}$ **in** E $\sqsubseteq \sigma\perp_D$.

Let γ_0 be $\perp_G$, as forced by Procedure 3 and as suggested by ϵ_0 .

ϵ_1 = $\pi\perp_G$ **in** E ;

= { $(1\ 2\ \perp_G)$, $(1\ \perp_G\ 2)$, $(2\ 1\ \perp_G)$, $(2\ \perp_G\ 1)$, $(\perp_G\ 1\ 2)$, $(\perp_G\ 2\ 1)$} **in** E .

Procedure 3 proceeds on the illustrated six choices for $\gamma_1 \in \epsilon_0|\mathcal{P}_{Sud}(G)$, and indexed $\epsilon_{1,i}$ as $1 \leq i \leq 6$.

$\epsilon_{1,1}$ $= \pi\ (1\ 2\ \perp_G)$ **in E** ;
$=\{\ (1\ 2\ 3),\ (\sout{1\ 3\ 2}),\ (\sout{2\ 1\ 3}),\ (\sout{2\ 3\ 1}),\ (\sout{3\ 1\ 2}),\ (\sout{3\ 2\ 1})\}$ **in E** .

$\epsilon_{1,2}$ $= \pi\ (1\ \perp_G\ 2)$ **in E** ;
$=\{\ (\sout{1\ 2\ 3}),\ (1\ 3\ 2),\ (\sout{2\ 1\ 3}),\ (\sout{2\ 3\ 1}),\ (\sout{3\ 1\ 2}),\ (\sout{3\ 2\ 1})\}$ **in E** .

$\epsilon_{1,3}$ $= \pi\ (2\ 1\ \perp_G)$ **in E** ;
$=\{\ (\sout{1\ 2\ 4}),\ (\sout{1\ 4\ 2}),\ (2\ 1\ 4),\ (\sout{2\ 4\ 1}),\ (\sout{4\ 1\ 2}),\ (\sout{4\ 2\ 1})\}$ **in E** .

$\epsilon_{1,4}$ $= \pi\ (2\ \perp_G\ 1)$ **in E** ;
$=\{\ (\sout{1\ 2\ 4}),\ (\sout{1\ 4\ 2}),\ (\sout{2\ 1\ 4}),\ (2\ 4\ 1),\ (\sout{4\ 1\ 2}),\ (\sout{4\ 2\ 1})\}$ **in E** .

$\epsilon_{1,5}$ $= \pi\ (\perp_G\ 1\ 2)$
$=\{\ (\sout{1\ 2\ \perp_G}),\ (\sout{1\ \perp_G\ 2}),\ (\sout{2\ 1\ \perp_G}),\ (\sout{2\ \perp_G\ 1}),\ (\perp_G\ 1\ 2),\ (\sout{\perp_G\ 2\ 1})\}$ **in E** .

$\epsilon_{1,6}$ $= \pi\ (\perp_G\ 2\ 1)$ **in E** ;
$=\{\ (\sout{1\ 2\ \perp_G}),\ (\sout{1\ \perp_G\ 2}),\ (\sout{2\ 1\ \perp_G}),\ (\sout{2\ \perp_G\ 1}),\ (\sout{\perp_G\ 1\ 2}),\ (\perp_G\ 2\ 1)\ \}$ **in E** .

where the crossed out values in **G** indicate values rejected at Step 3 of Procedure 3. Identify those residue values as (the first) candidates for γ_2.

There are still more candidates for γ_1, because we must also consider every γ beneath ($\sqsubseteq_{\mathbb{P}_{Sud}(G)}$) the γ_1 already processed above. These will be indexed $\epsilon_{1,j}$ for j a digitwise composite of the indexing above, depending on which elements of ϵ_1 introduce it. In this case, however, we thereby generate nothing not already beneath γ_2 values already discovered. (For the remainder of this example, we shall not distinguish among $\perp_G$, $\perp_T$ and $\perp_A$; all are written $\perp$.)

$\epsilon_{1,123456}$ $= \pi\ (\perp\ \perp\ \perp)$ **in E** $= \epsilon_1$;
$\epsilon_{1,25}$ $= \pi\ (\perp\ 1\ \perp)$ **in E** $= \epsilon_1$;
$\epsilon_{1,46}$ $= \pi\ (\perp\ \perp\ 1)$ **in E** $= \epsilon_1$;
$\epsilon_{1,16}$ $= \pi\ (\perp\ 2\ \perp)$ **in E** $= \epsilon_1$;
$\epsilon_{1,25}$ $= \pi\ (\perp\ \perp\ 2)$ **in E** $= \epsilon_1$;

$\epsilon_{1,12}$ $= \pi\ (1\ \perp\ \perp)$ **in E** ;
$=\{\ (1\ 2\ 3),\ (1\ 3\ 2),\ (\sout{2\ 1\ 3}),\ (\sout{2\ 3\ 1}),\ (\sout{3\ 1\ 2}),\ (\sout{3\ 2\ 1})\}$ **in E** .

$\epsilon_{1,34}$ $= \pi\ (2\ \perp\ \perp)$ **in E** ;
$=\{\ (\sout{1\ 2\ 4}),\ (\sout{1\ 4\ 2}),(2\ 1\ 4),\ (2\ 4\ 1),\ (\sout{4\ 1\ 2}),\ (\sout{4\ 2\ 1})\}$ **in E** .

In order to recover the full result in **E**, we continue to apply Procedure 3 to these γ_2 candidates, but doing so causes everything to be culled at Step 3, and the breadth-first evaluation stops.

$\epsilon_{1,1,1} = \pi(1\ 2\ 3)$ **in E** $= \epsilon_{1,1}$;
$\epsilon_{1,2,2} = \pi(1\ 3\ 2)$ **in E** $= \epsilon_{1,2}$;
$\epsilon_{1,3,3} = \pi(2\ 1\ 4)$ **in E** $= \epsilon_{1,3}$;
$\epsilon_{1,4,4} = \pi(2\ 4\ 1)$ **in E** $= \epsilon_{1,4}$;
$\epsilon_{1,5,5} = \pi(\perp_G\ 1\ 2)$ **in E** $= \epsilon_{1,5}$;
$\epsilon_{1,6,6} = \pi(\perp_G\ 2\ 1)$ **in E** $= \epsilon_{1,6}$;

That is, we have *reached* the fixed-points of all $\phi \leq \pi$ (which fortunately are isolated in this example) and Theorem 6 gives us the desired result. The result of evaluation is the join of all these $\{\gamma_2\}$ **in E**:

$$\epsilon = \{\ (1\ 2\ 3),\ (1\ 3\ 2),\ (2\ 1\ 4),\ (2\ 4\ 1),\ (\perp_G\ 1\ 2),\ (\perp_G\ 2\ 1)\ \}\ \textbf{in E}\ .$$

Note that neither **(4 1 2) in G** nor **(4 2 1) in G** occurs inside $\epsilon|\mathbb{P}_{Sud}(G)$. While either of these two values, γ', is a fixed point for some $\phi \leq \pi$ and so an element of a 'fixed point' with respect to π:

$$\{\gamma'\}\sqcup\ (\epsilon|\mathcal{P}_{Sud}(G)) \sqsubseteq\ \bigsqcup\{\pi\gamma\ |\ \gamma\in\ (\{\gamma'\}\sqcup\ (\epsilon|\mathcal{P}_{Sud}(G))\ )\}\ ;$$

nevertheless, each is **fix** ϕ for no $\phi<\pi$, and so fails to be in our "least" solution required by $\mathcal{K}$ [[fix]].

The denotational semantics developed here, therefore, meets our intuition on how

(letrec I {1 2 (if (first I)=1 then 3 else 4)})

should be computed effectively and our original intent for its final value (Section 1.3.)

6. Denotational semantics for ferns and multisets

This section presents the formal semantics that is the motivation of this paper. Parts of the definition, notably the definition of $\mathcal{E}$, and of the domains **D**, **E**, **S**, have already been seen above. The new content is the domain of structures, **M**, here called multistructures for historical reasons that are suggestive not of the ground domain (here extended), but of the power domain $\mathcal{P}$(**G**) which includes sets of structures. New here also is the rich variety of constants reflected in the extended definition of $\mathcal{K}$. Ferns [6,7] are a generalization of both deterministic lists and of multisets; all examples here use pure multisets.

Notable in $\mathcal{K}$ are the definitions of *frons* and *arbiter*; these are *not* isolated functions, here described in terms of fixed points of functionals. They are not the only primitive functions that return non-singleton values in $\mathcal{P}_{EM}$(**G**); both *amb* and *arbit* return such indeterminate values. They illustrate the need for approximating π by $\{\pi_i\}_{i\in\omega}$ in Theorem 6.

6.1. Syntactic Categories

I $\in$ **Ide**;	(the usual identifiers)
K $\in$ **Con**;	(constants)
E $\in$ **Exp**;	(expressions)
E* $\in$ **Exp***.	(expression strings)

6.2. Syntax

K :: fix | if | amb | cons | first | rest | null? | atom? | equal? | nil | zero? | succ | strictify | frons | arbit | arbiter | true | false | 0 | 1 | 2,...

E :: **I** | **K** | <**E***> | (**E**$_0$ **E**$_1$) | **(lambda I E)** | **(letrec I E)**

E* :: Λ | **E** **E***

6.3. Value Domains

$\rho \in \mathbf{U}$;
$\delta \in \mathbf{D}$; $\epsilon \in \mathbf{E}$; $\sigma \in \mathbf{S}$;
$\pi \in \mathbf{P}$; $\phi \in \mathbf{F}$; (as before)

$\gamma \in \mathbf{G} = \mathbf{A} + \mathbf{M}$; (ground values)
$\mu \in \mathbf{M} = \{\mathbf{nil}\} + \mathbf{T}$; (multistructures)
$\tau \in \mathbf{T} = \mathbf{G}\times\mathbf{M}$; (tuples or twins)
$\alpha \in \mathbf{A}$; (some primitive values, including **true,false,0,1,2,...**)

6.4. Notational Convention

The following notation is used to construct twins (non-null structures) that are strict in their *first* field. Compare to Landin's *prefix** [13].

Notation: $\ll\gamma,\mu\gg$ means $(\gamma=\bot_G \rightarrow \bot_T , <\gamma,\mu>)$.

6.5. Semantic Functions

$$\mathcal{E} : [\mathbf{Exp} \rightarrow [\mathbf{U} \rightarrow \mathbf{E}]] ;$$

$$\mathcal{E}\,[\![\mathrm{I}]\!]\rho \equiv \nabla\,(\rho\,[\![\mathrm{I}]\!]) ; \qquad \text{(parameter)}$$

$$\mathcal{E}\,[\![\mathrm{K}]\!]\rho \equiv \nabla\,(\mathcal{K}\,[\![\mathrm{K}]\!]) ; \qquad \text{(constant)}$$

$$\mathcal{E}\,[\![<>]\!]\rho \equiv \{nil\ \mathbf{in}\ \mathrm{G}\}\mathbf{in}\ \mathrm{E} ; \qquad \text{(empty multilist)}$$

$$\mathcal{E}\,[\![<\mathrm{E}\ \mathrm{E}^*>]\!]\rho \equiv \bigsqcup\{\, \{<\gamma_1, \gamma_2|\mathrm{M}>\ \mathbf{in}\ \mathrm{M}\ \mathbf{in}\ \mathrm{G}\}\mathbf{in}\ \mathrm{E} \mid \{\gamma_1\}\sqsubseteq(\mathcal{E}\,[\![\mathrm{E}]\!]\rho)|\mathcal{P}_{EM}(\mathrm{G}) \text{ and } \{\gamma_2\}\sqsubseteq(\mathcal{E}\,[\![<\mathrm{E}^*>]\!]\rho)|\mathcal{P}_{EM}(\mathrm{G})\,\} ; \qquad \text{(non-empty list)}$$

$$\mathcal{E}\,[\![(\mathbf{lambda}\ \mathrm{I}\ \mathrm{E})]\!]\rho \equiv (\lambda\delta\,\mathcal{E}\,[\![\mathrm{E}]\!]\rho[\delta/\mathrm{I}])\ \mathbf{in}\ \mathrm{E} ; \qquad \text{(function)}$$

$$\begin{aligned}\mathcal{E}\,[\![(\mathrm{E}_0\ \mathrm{E}_1)]\!]\rho \equiv\ & (\mathbf{Let}\ \sigma=((\mathcal{E}\,[\![\mathrm{E}_0]\!]\rho)|\mathrm{S}))\ \mathbf{and}\ \epsilon=(\mathcal{E}\,[\![\mathrm{E}_1]\!]\rho)\ \mathbf{in}\\ & (\epsilon \in \mathcal{P}_{EM}(\mathrm{G}) \rightarrow \bigsqcup\{\, \sigma(\gamma\ \mathbf{in}\ \mathrm{D}) \mid \gamma\in(\epsilon|\mathcal{P}_{EM}(\mathrm{G}))\,\},\\ & (\epsilon \in \mathrm{S} \rightarrow \sigma(\epsilon|\mathrm{S}\ \mathbf{in}\ \mathrm{D}),\\ & (\epsilon=\top_E \rightarrow \sigma\top_D ,\\ & \sigma\bot_D)))) ;\end{aligned} \qquad \text{(application)}$$

$$\mathcal{E}\,[\![(\mathbf{letrec}\ \mathrm{I}\ \mathrm{E})]\!]\rho \equiv \mathcal{E}\,[\![(\mathrm{fix}\ (\mathbf{lambda}\ \mathrm{I}\ \mathrm{E}))]\!]\rho\ . \qquad \text{(recursion*)}$$

The following constant primitives take arguments in G which sometimes are structured in tuples. The exception is $\mathcal{K}\,[\![\mathrm{if}]\!]$ which is "curried" to allow systems to be the result of conditionals.

$$\mathcal{K} : [\mathbf{Con} \rightarrow \mathbf{D}] ;$$

$$\begin{aligned}\mathcal{K}\,[\![\mathrm{fix}]\!] \equiv\ & (\lambda\delta.(\mathbf{Let}\ \sigma=\delta|\mathrm{S}\ \mathbf{in}\ (\sigma\bot_D\in\mathcal{P}_{EM}(\mathrm{G}) \rightarrow \psi(\lambda\gamma.((\sigma(\gamma\ \mathbf{in}\ \mathrm{D}))\mid\mathcal{P}_{EM}(\mathrm{G}))) ,\\ & (\sigma\bot_D\in\mathrm{S} \rightarrow \mathbf{fix}\ (\lambda\epsilon.(\sigma(\epsilon|\mathrm{S}\ \mathbf{in}\ \mathrm{D}))),\\ & (\sigma\bot_D=\top_E \rightarrow \top_E ,\\ & \bot_E))))\ ))\ \mathbf{in}\ \mathrm{D} ;\end{aligned} \qquad (*)$$

$$\begin{aligned}\mathcal{K}\,[\![\mathrm{if}]\!] \equiv\ & (\lambda\delta.\ (\mathbf{Let}\ \alpha=\delta|\mathrm{G}|\mathrm{A}\ \mathbf{in}\ (\ \alpha = \mathbf{true} \rightarrow (\lambda\delta_1.((\lambda\delta_2.\nabla\ \delta_1)\mathbf{in}\ \mathrm{E})),\\ & (\ \alpha = \mathbf{false} \rightarrow (\lambda\delta_1.((\lambda\delta_2.\nabla\ \delta_2)\mathbf{in}\ \mathrm{E})),\\ & \bot_S\ ))\mathbf{in}\ \mathrm{E}))\ \mathbf{in}\ \mathrm{D} ;\end{aligned}$$

$$\begin{aligned}\mathcal{K}\,[\![\mathrm{amb}]\!] \equiv\ & (\lambda\delta.(\mathbf{Let}\ \tau=\delta|\mathrm{G}|\mathrm{M}|\mathrm{T}\ \mathbf{in}\\ & \{\ (\tau\downarrow 1),\ (\tau\downarrow 2|\mathrm{T}\downarrow 1)\ \}\mathbf{in}\ \mathrm{E}))\ \mathbf{in}\ \mathrm{D} ;\end{aligned} \qquad [14]$$

$\mathcal{K}$ [[cons]] ≡ (λδ.(Let τ=δ|G|M|T in
{ <τ↓1, (τ↓2|T↓1|M)> in M in G}in E)) in D ; [14]

$\mathcal{K}$ [[first]] ≡ (λδ. ({δ|G|M|T↓1}in E)) in D ; — [14: *car*]

$\mathcal{K}$ [[rest]] ≡ (λδ. ({ (δ|G|M|T↓2)in G }in E)) in D ; [14: *cdr*]

$\mathcal{K}$ [[null?]] ≡ (λδ.(Let μ=δ|G|M in [14: *null*]
{ (μ = nil → true,
(μ = $\perp_M$ → $\perp_A$,
false))in G }in E)) in D ;

$\mathcal{K}$ [[atom?]] ≡ (λδ. ({ (δ|G ∈ A → true, [14: *atom*]
(δ|G ∈ M → false,
$\perp_A$))in G }in E)) in D ;

$\mathcal{K}$ [[equal?]] ≡ (λδ. (Let τ=δ|G|M|T, α_1=τ↓1|A, α_2=τ↓2|T↓1|A in
{ (α_1=$\perp_A$ → $\perp_A$,
(α_2=$\perp_A$ → $\perp_A$,
(α_1=α_2 → true,
false)))in G }in E)) in D ; [14: *eq*]

$\mathcal{K}$ [[nil]] ≡ nil in G in D ; [14]

$\mathcal{K}$ [[zero?]] ≡ (λδ.(Let α=δ|G|A in
{ (α=0 → true,
(α=$\perp_A$ → $\perp_A$,
false)) in G }in E)) in D ;

$\mathcal{K}$ [[succ]] ≡ (λδ. ({ (δ|G|A + 1)in G }in E)) in D ;

$\mathcal{K}$ [[α]] ≡ α in G in D ; for all α ∈ A

$$\mathcal{K}\ [\![\text{strictify}]\!] \equiv (\lambda\delta.(\text{Let } \tau=\delta|\mathbf{G}|\mathbf{M}|\mathbf{T} \text{ in } \{(\tau\downarrow 1=\perp_{\mathbf{G}} \rightarrow \perp_{\mathbf{G}}, \tau\downarrow 2|\mathbf{T}\downarrow 1)\} \text{in } \mathbf{E})) \text{ in } \mathbf{D}\ ; \qquad [7]$$

$$\mathcal{K}\ [\![\text{frons}]\!] \equiv (\lambda\delta.(\text{Let } \tau=\delta|\mathbf{G}|\mathbf{M}|\mathbf{T} \text{ in } (\mathit{fons}\ (\tau\downarrow 1))\ ((\tau\downarrow 2|\mathbf{T}\downarrow 1)|\mathbf{M})\ ) \text{ in } \mathcal{P}_{\mathrm{EM}}(\mathbf{M}) \text{ in } \mathcal{P}_{\mathrm{EM}}(\mathbf{G}) \text{ in } \mathbf{E}))\text{in } \mathbf{D} \qquad [7]$$

where $\mathit{fons} : \mathbf{G} \rightarrow \mathbf{M} \rightarrow \mathcal{P}_{\mathrm{EM}}(\mathbf{T})$;

$$\mathit{fons} = \mathbf{fix}\ (\lambda f\lambda\gamma\lambda\mu.\ (\{\ll\gamma,\mu\gg\} \sqcup\ (\mu=\mathbf{nil} \rightarrow \{<\gamma,\mu>\}, \{\ll(\mu|\mathbf{T}\downarrow 1), \tau \text{ in } \mathbf{M}\gg \mid \tau \in ((f\gamma)(\mu|\mathbf{T}\downarrow 2))\}\ )\ ))\ ;$$

$$\mathcal{K}\ [\![\text{arbit}]\!] \equiv (\lambda\delta.(\text{Let } \tau = \delta|\mathbf{G}|\mathbf{M}|\mathbf{T} \text{ in } \{(\tau\downarrow 1 = \perp_{\mathbf{G}} \rightarrow \perp_{\mathbf{G}}, \mathbf{true} \text{ in } \mathbf{G}), (\tau\downarrow 2|\mathbf{T}\downarrow 1 = \perp_{\mathbf{G}} \rightarrow \perp_{\mathbf{G}}, \mathbf{false} \text{ in } \mathbf{G})\} \text{ in } \mathbf{E})) \text{ in } \mathbf{D}; \qquad [8]$$

$$\mathcal{K}\ [\![\text{arbiter}]\!] \equiv (\lambda\delta.((\text{Let } \tau=\delta|\mathbf{G}|\mathbf{M}|\mathbf{T} \text{ in } ((\mathit{demult}(\tau\downarrow 1|\mathbf{M}|\mathbf{T}))(\tau\downarrow 2|\mathbf{T}\downarrow 1|\mathbf{M}|\mathbf{T}))\text{in } \mathcal{P}_{\mathrm{EM}}(\mathbf{M}) \text{ in } \mathcal{P}_{\mathrm{EM}}(\mathbf{G}) \text{ in } \mathbf{E})) \text{ in } \mathbf{D} \qquad [12]$$

where $\mathit{demult} : \mathbf{T} \rightarrow \mathbf{T} \rightarrow \mathcal{P}_{\mathrm{EM}}(\mathbf{T})$;

$$\mathit{demult} = \mathbf{fix}(\lambda d\lambda\tau_1\lambda\tau_2.\ (\{\ll\ll\tau_1\downarrow 1, <\mathbf{true} \text{ in } \mathbf{G}, \mathbf{nil}> \gg \text{in } \mathbf{M} \text{ in } \mathbf{G}, \tau \text{ in } \mathbf{M}\gg \mid \tau\in((d(\tau_1\downarrow 2|\mathbf{T}))\tau_2)\} \sqcup \{\ll\ll\tau_2\downarrow 1, <\mathbf{false} \text{ in } \mathbf{G}, \mathbf{nil}> \gg \text{in } \mathbf{M} \text{ in } \mathbf{G}, \tau \text{ in } \mathbf{M}\gg \mid \tau\in((d\tau_1)(\tau_2\downarrow 2|\mathbf{T}))\}\ )).$$

---

7. MacQueen's Example with Multistructures

Reconsider MacQueen's problem, now recast in the language of the previous section. We seek the value of

(**letrec** I (frons< 1 (frons< 2 (frons <(((if (equal?< 1(first I)>)) 3)4) nil>) >) >))

according to this language, with this example interpreted in $\mathcal{P}_{\mathrm{EM}}(\mathbf{G})$.

A major difference in the set that is the answers arises because *frons* is strict in the first position of its result *unless* it happens to be a "tuple" of one ground value—with *rest* yielding **nil**.

No explanations or intuition here! We just want the value that $\mathcal{E}$ gives for this expression in environment $\perp_{\mathbf{U}}$.

7.1. Approximations to *frons*

As indicated earlier, *frons* has not been defined as simply as, say, *amb*. According to its definition it only exists as a limit point, which will here be approximated to a sufficient degree only to answer the problem at hand. Really it is not *frons* that has been specified directly as a fixed point of a functional, but rather its 'helper,' *fons*, which is now explored.

$$\mathit{fons} : \mathbf{G} \rightarrow \mathbf{M} \rightarrow \mathcal{P}_{\mathrm{EM}}(\mathbf{T})\ ;$$

$$\mathit{fons} = \mathbf{fix}\ (\lambda f\lambda\gamma\lambda\mu.\ (\{\ll\gamma,\mu\gg\} \sqcup\ (\mu=\mathbf{nil} \rightarrow \{<\gamma,\mu>\}, \{\ll(\mu|\mathbf{T}\downarrow 1), \tau \text{ in } \mathbf{M}\gg \mid \tau \in ((f\gamma)(\mu|\mathbf{T}\downarrow 2))\}\ )\ ))\ ;$$

The value of *fons* will be approximated by an ascending sequence $\{\mathit{fons}_i\}_{i\in\omega}$. Each element is the result of $i \geq 0$ applications of the lambda expression, in the fixed-point expression above, to $\perp_{\mathbf{G}\rightarrow\mathbf{M}\rightarrow\mathcal{P}_{\mathrm{EM}}(\mathbf{T})}$.

$$
\begin{aligned}
fons_0 &= \bot_{G\to M\to \mathcal{P}_{EM}(T)};\\
fons_{i+1} &= (\text{Let } f=fons_i \text{ in}\\
&\qquad \lambda\gamma\lambda\mu.\ (\{\lessdot\gamma,\mu\gtrdot\} \sqcup\ (\mu=\mathbf{nil} \to \{<\gamma,\mu>\},\\
&\qquad\qquad \{\lessdot(\mu|T\downarrow 1),\tau \ \mathbf{in}\ M\gtrdot \mid \tau \in ((f\gamma)(\mu|T\downarrow 2))\}\ ))\ );
\end{aligned}
$$

Then,

$$
\begin{aligned}
fons_1 &= \lambda\gamma\lambda\mu.\ (\{\lessdot\gamma,\bot_M\gtrdot\} \sqcup\ (\mu=\mathbf{nil} \to \{<\gamma,\mu>\},\{\bot_T\}\ ))\ ;\\
&= \lambda\gamma\lambda\mu.\ (\mu=\mathbf{nil} \to \{<\gamma,\mathbf{nil}>\},\{\lessdot\gamma,\mu\gtrdot,\bot_T\}\ )\ .
\end{aligned}
$$

$$
\begin{aligned}
fons_2 &= \lambda\gamma\lambda\mu.\ (\{\lessdot\gamma,\bot_M\gtrdot\} \sqcup\\
&\qquad (\mu=\mathbf{nil} \to \{<\gamma,\mu>\},\\
&\qquad\quad \{\lessdot(\mu|T\downarrow 1),\tau \ \mathbf{in}\ M\gtrdot \mid \tau \in (\mu|T\downarrow 2=\mathbf{nil} \to \{<\gamma,\mathbf{nil}>\},\\
&\qquad\qquad\qquad \{\lessdot\gamma,\mu|T\downarrow 2\gtrdot,\bot_T\}\}\ ))\ ;\\
&= \lambda\gamma\lambda\mu.\ (\mu=\mathbf{nil} \to \{<\gamma,\mathbf{nil}>\},\\
&\qquad (\mu|T\downarrow 2)=\mathbf{nil} \to \{\lessdot\gamma,\mu\gtrdot\}\sqcup\ \{\lessdot\mu|T\downarrow 1,<\gamma,\mathbf{nil}>\gtrdot\},\\
&\qquad \{\lessdot\mu|T\downarrow 1,\lessdot\gamma,\mu|T\downarrow 2\gtrdot \mathbf{in}\ M\gtrdot,\lessdot\gamma,\mu\gtrdot,\bot_T\}\ ))\ .
\end{aligned}
$$

According to a similar derivation:

$$
\begin{aligned}
fons_3 &= \lambda\gamma\lambda\mu.\ (\mu=\mathbf{nil} \to \{<\gamma,\mathbf{nil}>\},\\
&\qquad (\mu|T\downarrow 2)=\mathbf{nil} \to \{\lessdot\gamma,\mu\gtrdot\}\sqcup\ \{\lessdot\mu|T\downarrow 1,<\gamma,\mathbf{nil}>\ \mathbf{in}\ M\gtrdot\},\\
&\qquad (\mu|T\downarrow 2|T\downarrow 2=\mathbf{nil} \to \{\lessdot\gamma,\mu\gtrdot\}\sqcup\ \{\lessdot\mu|T\downarrow 1,\lessdot\gamma,\mu|T\downarrow 2\gtrdot \mathbf{in}\ M\gtrdot\}\ \sqcup\\
&\qquad\qquad \{\lessdot\mu|T\downarrow 1,\lessdot\mu|T|\downarrow 2|T\downarrow 1,<\gamma,\mathbf{nil}>\ \mathbf{in}\ M\gtrdot \mathbf{in}\ M\gtrdot\}\ ,\\
&\qquad \{\lessdot\mu|T\downarrow 1,\lessdot\mu|T|\downarrow 2|T\downarrow 1,\lessdot\gamma,\mu|T\downarrow 2|T\downarrow 2\gtrdot \mathbf{in}\ M\gtrdot \mathbf{in}\ M\gtrdot,\\
&\qquad\qquad \lessdot\mu|T\downarrow 1,\lessdot\gamma,\mu|T\downarrow 2\gtrdot \mathbf{in}\ M\gtrdot,\lessdot\gamma,\mu\gtrdot,\bot_T\}\ )))\ .
\end{aligned}
$$

Now we can proceed with the approximation to $\mathcal{K}$ [[frons]] which we shall call $\{frons_i\}_{i\in\omega}$.

$$
frons_i = (\lambda\delta.(\text{Let } \tau=\delta|G|M|T \text{ in } (\ (fons_i\ (\tau\downarrow 1))\ ((\tau\downarrow 2|T\downarrow 1)|M)\ )\mathbf{in}\ \mathcal{P}_{EM}(M)\ \mathbf{in}\ \mathcal{P}_{EM}(G)\ \mathbf{in}\ E))\mathbf{in}\ D
$$

yielding much the same functions as appear above except for argument selectors:

$$
\begin{aligned}
frons_0 &= (\lambda\delta.(\text{Let } \tau=\delta|G|M|T,\ \gamma=(\tau\downarrow 1),\ \text{and } \mu=(\tau\downarrow 2|T\downarrow 1)|M \text{ in}\\
&\qquad \overset{\bot_{\mathcal{P}_{EM}(G)}}{\mathbf{in}\ \mathcal{P}_{EM}(M)}\ \mathbf{in}\ \mathcal{P}_{EM}(G)\ \mathbf{in}\ E))\ \mathbf{in}\ D\ .
\end{aligned}
$$

$$
\begin{aligned}
frons_1 &= (\lambda\delta.(\text{Let } \tau=\delta|G|M|T,\ \gamma=(\tau\downarrow 1),\ \text{and } \mu=(\tau\downarrow 2|T\downarrow 1)|M \text{ in}\\
&\qquad (\mu=\mathbf{nil} \to \{<\gamma,\mathbf{nil}>\},\{\lessdot\gamma,\mu\gtrdot,\bot_T\}\ )\\
&\qquad \mathbf{in}\ \mathcal{P}_{EM}(M)\ \mathbf{in}\ \mathcal{P}_{EM}(G)\ \mathbf{in}\ E))\ \mathbf{in}\ D\ .
\end{aligned}
$$

$$
\begin{aligned}
frons_2 &= (\lambda\delta.(\text{Let } \tau=\delta|G|M|T,\ \gamma=(\tau\downarrow 1),\ \text{and } \mu=(\tau\downarrow 2|T\downarrow 1)|M \text{ in}\\
&\qquad (\mu=\mathbf{nil} \to \{<\gamma,\mathbf{nil}>\},\\
&\qquad (\mu|T\downarrow 2)=\mathbf{nil} \to \{\lessdot\gamma,\mu\gtrdot\}\sqcup\ \{\lessdot\mu|T\downarrow 1,<\gamma,\mathbf{nil}>\ \mathbf{in}\ M\gtrdot\},\\
&\qquad \{\lessdot\mu|T\downarrow 1,\lessdot\gamma,\mu|T\downarrow 2\gtrdot \mathbf{in}\ M\gtrdot,\lessdot\gamma,\mu\gtrdot,\bot_T\}\ ))\\
&\qquad \mathbf{in}\ \mathcal{P}_{EM}(M)\ \mathbf{in}\ \mathcal{P}_{EM}(G)\ \mathbf{in}\ E))\ \mathbf{in}\ D\ .
\end{aligned}
$$

$$
\begin{aligned}
frons_3 &= (\lambda\delta.(\text{Let } \tau=\delta|G|M|T,\ \gamma=(\tau\downarrow 1),\ \text{and } \mu=(\tau\downarrow 2|T\downarrow 1)|M \text{ in}\\
&\qquad (\mu=\mathbf{nil} \to \{<\gamma,\mathbf{nil}>\},\\
&\qquad (\mu|T\downarrow 2)=\mathbf{nil} \to \{\lessdot\gamma,\mu\gtrdot\}\sqcup\ \{\lessdot\mu|T\downarrow 1,<\gamma,\mathbf{nil}>\ \mathbf{in}\ M\gtrdot\},\\
&\qquad (\mu|T\downarrow 2|T\downarrow 2=\mathbf{nil} \to \{\lessdot\gamma,\mu\gtrdot\}\sqcup\ \{\lessdot\mu|T\downarrow 1,\lessdot\gamma,\mu|T\downarrow 2\gtrdot \mathbf{in}\ M\gtrdot\}\ \sqcup\\
&\qquad\qquad \{\lessdot\mu|T\downarrow 1,\lessdot\mu|T|\downarrow 2|T\downarrow 1,<\gamma,\mathbf{nil}>\ \mathbf{in}\ M\gtrdot \mathbf{in}\ M\gtrdot\}\ ,\\
&\qquad \{\lessdot\mu|T\downarrow 1,\lessdot\mu|T|\downarrow 2|T\downarrow 1,\lessdot\gamma,\mu|T\downarrow 2|T\downarrow 2\gtrdot \mathbf{in}\ M\gtrdot \mathbf{in}\ M\gtrdot,\\
&\qquad\qquad \lessdot\mu|T\downarrow 1,\lessdot\gamma,\mu|T\downarrow 2\gtrdot \mathbf{in}\ M\gtrdot,\lessdot\gamma,\mu\gtrdot,\bot_T\}\ )))\\
&\qquad \mathbf{in}\ \mathcal{P}_{EM}(M)\ \mathbf{in}\ \mathcal{P}_{EM}(G)\ \mathbf{in}\ E))\ \mathbf{in}\ D\ .
\end{aligned}
$$

7.2. Evaluation

Considering

$\mathcal{E}$ ⟦(letrec 1 (frons< 1 (frons< 2 (frons < (((if (equal?< 1(first I)>)) 3)4) nil>) >) >))⟧$\perp_U$

we observe the derivation of σ and π:

$\sigma = \lambda\delta.\mathcal{E}$ ⟦(frons< 1 (frons< 2 (frons < (((if (equal?< 1(first I)>)) 3)4) nil>) >) >)⟧$\perp_U[\delta/I]$;

$\pi = \lambda\gamma.((\sigma(\gamma \text{ in } D)) \mid \mathcal{P}_{EM}(G))$.

In order to use Procedure 3, we need an increasing sequence of isolated points approximating π:

$\sigma_i = \lambda\delta.\mathcal{E}$ ⟦(frons$_i$< 1 (frons$_i$< 2 (frons$_i$ < (((if (equal?< 1(first I)>)) 3)4) nil>) >) >)⟧$\perp_U[\delta/I]$;

$\pi_i = \lambda\gamma.((\sigma_i(\gamma \text{ in } D)) \mid \mathcal{P}_{EM}(G))$.

Then, again omitting the tiresome domain coercions like **in M** and **in** $\mathcal{P}_{EM}(G)$ as before,

$$\pi_0\perp_G = \{\perp_G\} .$$

$$\pi_1\perp_G = \{ <1, \perp_T>, \perp_T \} .$$

$$\pi_2\perp_T = \{ <1, <2, \perp_T>> , <2, <1, \perp_T>>, <1, \perp_T>, <2, \perp_T>, \perp_T \} .$$

$$\pi_2<1, \perp_T> = \{ <1, <2, <3, \mathbf{nil}>>>, <1, <3, <2, \mathbf{nil}>>>, <1, <2, \perp_T>>, <1, <3, \perp_T>>, <1, \perp_T>, \perp_T \} .$$

Note that π_2 cannot yet generate <3, <2, <1, **nil**>>> and <2, <3, <1, **nil**>>> . The restriction arises from the fact that π_2 can only reverse adjacent pairs of elements in the multistructures; it cannot freely permute triples.

$$\pi_3\langle 1, \langle 2, \langle 3, \mathbf{nil}\rangle\rangle\rangle = \{ \langle 1, \langle 2, \langle 3, \mathbf{nil}\rangle\rangle\rangle, \cancel{\langle 1, \langle 3, \langle 2, \mathbf{nil}\rangle\rangle\rangle}, \cancel{\langle 2, \langle 1, \langle 3, \mathbf{nil}\rangle\rangle\rangle}, \langle 2, \langle 3, \langle 1, \mathbf{nil}\rangle\rangle\rangle, \cancel{\langle 3, \langle 1, \langle 2, \mathbf{nil}\rangle\rangle\rangle}, \cancel{\langle 3, \langle 2, \langle 1, \mathbf{nil}\rangle\rangle\rangle} \} = \pi_3\langle 1, \langle 2, \bot_T\rangle\rangle .$$

$$\pi_3\langle 1, \langle 3, \langle 2, \mathbf{nil}\rangle\rangle\rangle = \{ \cancel{\langle 1, \langle 2, \langle 3, \mathbf{nil}\rangle\rangle\rangle}, \langle 1, \langle 3, \langle 2, \mathbf{nil}\rangle\rangle\rangle, \cancel{\langle 2, \langle 1, \langle 3, \mathbf{nil}\rangle\rangle\rangle}, \cancel{\langle 2, \langle 3, \langle 1, \mathbf{nil}\rangle\rangle\rangle}, \cancel{\langle 3, \langle 1, \langle 2, \mathbf{nil}\rangle\rangle\rangle}, \cancel{\langle 3, \langle 2, \langle 1, \mathbf{nil}\rangle\rangle\rangle} \} = \pi_3\langle 1, \langle 3, \bot_T\rangle\rangle .$$

$$\pi_3\langle 2, \langle 1, \bot_T\rangle\rangle = \{ \cancel{\langle 1, \langle 2, \langle 4, \mathbf{nil}\rangle\rangle\rangle}, \cancel{\langle 1, \langle 4, \langle 2, \mathbf{nil}\rangle\rangle\rangle}, \langle 2, \langle 1, \langle 4, \mathbf{nil}\rangle\rangle\rangle, \cancel{\langle 2, \langle 4, \langle 1, \mathbf{nil}\rangle\rangle\rangle}, \cancel{\langle 4, \langle 1, \langle 2, \mathbf{nil}\rangle\rangle\rangle}, \cancel{\langle 4, \langle 2, \langle 1, \mathbf{nil}\rangle\rangle\rangle} \} .$$

$$\pi_3\langle 1, \bot_T\rangle = \{ \langle 1, \langle 2, \langle 3, \mathbf{nil}\rangle\rangle\rangle, \langle 1, \langle 3, \langle 2, \mathbf{nil}\rangle\rangle\rangle, \langle 1, \langle 2, \bot_T\rangle\rangle, \langle 1, \langle 3, \bot_T\rangle\rangle, \langle 1, \bot_T\rangle, \cancel{\langle 2, \langle 1, \langle 3, \mathbf{nil}\rangle\rangle\rangle}, \cancel{\langle 2, \langle 3, \langle 1, \mathbf{nil}\rangle\rangle\rangle}, \cancel{\langle 3, \langle 1, \langle 2, \mathbf{nil}\rangle\rangle\rangle}, \cancel{\langle 3, \langle 2, \langle 1, \mathbf{nil}\rangle\rangle\rangle} \} .$$

$$\pi_3\langle 2, \bot_T\rangle = \{ \cancel{\langle 1, \langle 2, \langle 4, \mathbf{nil}\rangle\rangle\rangle}, \cancel{\langle 1, \langle 4, \langle 2, \mathbf{nil}\rangle\rangle\rangle}, \langle 2, \langle 1, \langle 4, \mathbf{nil}\rangle\rangle\rangle, \langle 2, \langle 4, \langle 1, \mathbf{nil}\rangle\rangle\rangle, \cancel{\langle 4, \langle 1, 2\rangle\rangle\rangle}, \cancel{\langle 4, \langle 2, 1\rangle\rangle\rangle} \} .$$

$$\pi_3\bot_T = \{ \langle 1, \langle 2, \bot_T\rangle\rangle, \langle 2, \langle 1, \bot_T\rangle\rangle, \langle 1, \bot_T\rangle, \langle 2, \bot_T\rangle, \bot_T \} .$$

The next step, using π_4, is not presented here because it does not introduce any new values into $\psi\pi$. The derivation will then stop with the following precisely defined element in the power domain:

$$\{ \bot_T, \langle 1, \langle 2, \langle 3, \mathbf{nil}\rangle\rangle\rangle, \langle 1, \langle 3, \langle 2, \mathbf{nil}\rangle\rangle\rangle, \langle 2, \langle 1, \langle 4, \mathbf{nil}\rangle\rangle\rangle, \langle 2, \langle 4, \langle 1, \mathbf{nil}\rangle\rangle\rangle \} .$$

There are other γ in this powerdomain element, besides those explicitly listed, specifically those included by closing this set to the representative of the congruence class defined from $\sqsubseteq_{EM}$. This set is the set of those γ satisfying the proof rule.

The presence of $\bot_T$ here is annoying, since ithe opportunity for precisely expressing bottom-avoiding set values is one reason for using the Plotkin powerdomain. This is an instance, suggested after the proof rule in Section 4.3, where we would prefer to accept only the supremum set of the four permutations, avoiding the inferior $\bot_T$. Avoidance of isolated inferior values, and certainly avoidance of isolated $\bot$, may be accomplished in implementation using ordinary breadth-first evaluation to implement Procedure 3 (with sinister requirements for global breadth-first evaluation [4] if used throughout the semantics.) If there were a mechanism to implement bottom-avoidance in $\mathcal{P}_{EM}(T)$ then it would generate exactly four ground values in the resulting powerdomain element, because once $\bot_T$ is avoided only the four elements listed explicitly above survive as limit points. Lesser values are here included only by closure of an element that included $\bot$.

Broy [1] describes the gap between $\perp_T$ and the four supremea as "holes" within a powerdomain element; one way to avoid them is to so collapse all inferior values to $\perp$, as was done above, and then to force interpretation of such an element as its non-bottom constituents, Clinger's "representative element" [3] in $\mathcal{P}_{Sud}(\mathbf{G})$. This can be done by using an appropriate bottom-avoiding join in all definitions of *fons* and $fons_i$. Here we would use the join on $\mathcal{P}(\mathbf{T})$, $\dot{\sqcup}$, defined by

$$A \mathbin{\dot{\sqcup}} B = (A = \perp \rightarrow B, A \sqcup B) .$$

This operator coincides with $\sqcup$ in $\mathcal{P}_{Sud}(\mathbf{G})$, but is not even monotone in $\mathcal{P}_{EM}(\mathbf{G})$. Used in the definition of *fons* with the ordinary join used in the function application axiom, however, it does yield the operational behavior originally ascribed to *frons* [6]. We would like a semantics to surgically eliminate bottom like this, allowing bottom-avoidance within definitions like that of *fons*, even if it is not a generally safe powerdomain join. Perhaps, as Broy suggests, there is a way of alternating between two power domains that does it.

The fifth and sixth permutations that arose in the "triples semantics" of Section 5.2 are collapsed to $\perp_T$ by the strictness inherent in *frons*. The first element of the extra two triples in the resulting powerdomain element there was $\perp_G$. Both are indicated by the presence of $\perp_T$ in the result above, and in implementation, too, if bottom-avoidance is ignored.

---

8. Rote: System Structures

Functions are supposed to be first class citizens under denotational semantics. Yet the language of Section 7 provides no way for $\sigma \in \mathbf{S}$ to be preserved within a structure. In this section we present a minor enhancement that provides this ability.

Thus far, only ground values could be in data structures and any structure (multistructure in **M**). This is true because systems may be used to encode a set of values, and certainly we do not want ground values to be nested many layers deep in nondeterminism. (The application rule would then be required to look arbitrarily deep in set structure.)

We escape this problem by specifying an entirely new domain of data structures, the "rote", which is little more than a list of denoted values. It, however, is deterministic!!!. The only indeterminism in $\rho \in \mathbf{R}$ occurs within the function domain $[\mathbf{D} \rightarrow \mathbf{E}]$ — not directly in a rote. Infinite streams, therefore, pose no problem for fairness if restricted to rotes and thereby excluded from indeterminism.

8.1. Syntax

Two additions are made to Section 6.2. A new primitive function is introduced to construct rotes, and a new constant is needed to denote the empty rote.

K :: kons | nul .

8.2. Value Domains

Ide is countable; the map from identifiers used in any finite program fragment to a finite prefix of ω is often called the "symbol table". Because identifiers are countable, any environment can be represented as a rote, where the i^{th} identifier is discharged by accessing the i^{th} position in the rote (treated as a list). Thus, the reuse of the letter ρ is consistent with its former notation; **U** can be subsumed by **R**.

The following is a revision of Section 6.3

$\rho \in U = [Ide \to D]$; (environments)

$\delta \in D = G \boxplus S$; (denoted values)
$\epsilon \in E = \mathcal{P}_{EM}(G) \boxplus S$; (expressed values)
$\sigma \in S = [D \to E] + R$; (systems)

$\pi \in P = [G \to \mathcal{P}_{EM}(G)]$; (programs)
$\phi \in F = [G \to G]$; (functions)

$\gamma \in G = A + M$; (ground values)
$\mu \in M = \{nil\} + T$; (multistructures)
$\tau \in T = G \times M$; (tuples or twins)
$\alpha \in A$; (atoms)

$\rho \in R = \{nul\} + D \times R \sim U$. (rote)

8.3. Semantic Functions

This section presents only the structure building primitives revised from Section 6.5. The code for *kons*, like the code for *if* there, must be curried to allow arguments in S. The structure probing functions have been altered here to handle an argument from either domain of structures: M or R .

$\mathcal{K}[\![cons]\!]$ (as in Section 6.5)
$\mathcal{K}[\![nil]\!]$ (as in Section 6.5)

$$\mathcal{K}[\![kons]\!] \equiv (\lambda\delta_1.(\lambda\delta_2.(<\delta_1, \delta_2|S|R> \text{ in } S \text{ in } E))\text{in } E)\text{in } D .$$

$$\begin{aligned}\mathcal{K}[\![first]\!] \equiv (\lambda\delta.(&\delta \in G \to (\{\delta|G|M|T\downarrow 1\} \text{ in } E),\\ &(\delta \in S \to \nabla(\delta|S|R\downarrow 1),\\ &(\delta = \top_D \to \top_E,\\ &\perp_E)))) \text{ in } D ;\end{aligned}$$

$$\begin{aligned}\mathcal{K}[\![rest]\!] \equiv (\lambda\delta.(&\delta \in G \to (\{(\delta|G|M|T\downarrow 2) \text{in } G\} \text{ in } E),\\ &(\delta \in S \to (\delta|S|R\downarrow 2 \text{ in } S \text{ in } E),\\ &(\delta = \top_D \to \top_E,\\ &\perp_E)))) \text{ in } D ;\end{aligned}$$

$$\begin{aligned}\mathcal{K}[\![null?]\!] \equiv (\lambda\delta.(&\delta = \top_D \to \top_E,\\ &\{(\delta \in G \to (\text{Let } \mu = \delta|G|M \text{ in}\\ &\quad(\mu = \text{nil} \to \text{true},\\ &\quad(\mu = \perp_M \to \perp_A,\\ &\quad\text{false}))),\\ &(\delta \in S \to (\text{Let } \rho = \delta|S|R \text{ in}\\ &\quad(\rho = \text{nul} \to \text{true},\\ &\quad(\rho = \perp_R \to \perp_A,\\ &\quad\text{false}))),\\ &\perp_A)) \text{ in } G\} \text{in } E)) \text{ in } D ;\end{aligned}$$

$\mathcal{K}[\![nul]\!] \equiv$ **nul in S in D** ;

9. Other Examples

This section presents additional examples without complete exposition. They are important, however, because they represent the standard problems addressed by this theory. First a stream-merge is presented in terms of *frons*, and later subsections present *frons, amb, arbit*, and *arbiter*, each expressed in terms of the next.

Code is written in the expanded programming language of Section 6. Evaluation of these fragments is trivial, yielding a lambda expression that is not terribly interesting. Application of this lambda expression to an argument, however, *is* interesting, because that argument should evaluate to an element in $\mathcal{P}_{EM}(G)$ and the naturally extended version of application will be used over the ground elements in that set. Natural extension does all the interesting work!

A proof that $\mathcal{E}$ valuation of each of the codes (with $\perp_U$) below yields the same functions as $\mathcal{K}$ does in Section 6 would establish that each of the four primitives is of equivalent power. However, the semantics of Section 6 includes infinite nesting in M and so a transfinite induction would be necessary for Sections 9.2 and 9.5. Recursion-induction will work, however, for finite cases there. Sections 9.3 and 9.4 are straightforward; only the proof for 9.3 is given.

9.1. Merge and other Indeterminate Operators

Merge is designed to work on streams. It takes as an argument a multistructure (really a list) of streams, and interleaves them as long as non-bottom elements occur as the first element of each stream. Any stream with a bottom as its first element is ignored as the interleaving continues on the others.

```
merge = (lambda listoflists
  ( (letrec mer (lambda streams
                  (cons < (first(first streams))
                          (mer (frons < (rest(first streams)) (rest streams)> )) > ) ))
    ( (letrec streamify (lambda lists
                  (frons < ((letrec stream (lambda list
                              (strictify <     (first list)
                                               (cons< (first list)(streamify(rest list))> ) > ) ))
                            (first lists))
                           (streamify (rest lists)) > ) ))
      listoflists)) )) .
```

In the preceding code, the function identified as *stream* turns a list (or nested tuple) into one whose successive suffices are strict in the associated prefixes; the name is taken from Landin [13]. The function *streamify* takes a list of lists and returns a shuffled list of the streams derived therefrom. Then *merge* does the interleaving based on the convergence of the successive first stream, and indirectly its first element.

For an application of *merge* consider

$$\mathcal{E}\,[\![(\text{merge}< <1\,2\,3> <4\ (\textbf{letrec}\ I\ I)> \ (\textbf{letrec}\ J\ J) >]\!]\perp_U .$$

The result of *streamify* is the set of permutations of the list containing

$$<1,<2,<3,\perp_M>>>>,\ <4,\perp_M>,\ \text{and}\ \perp_G .$$

*Merg*ing then yields the result

$$\{\ <1,<2,<3,<4,\perp_M>>>>,\ <1,<2,<4,<3,\perp_M>>>>,$$
$$<1,<4,<2,<3,\perp_M>>>>,\ <4,<1,<2,<3,\perp_M>>>>\ \}$$

in $\mathcal{P}_{EM}(G)$ as specified.

9.2. *Frons* **in terms of** *Amb*

```
(letrec frons (lambda pair
        (amb<   (((if (null? (first(rest pair)))) pair) (letrec I I) )
                (amb<   (strictify< (first pair) (cons < (first pair)(first(rest pair))> ) > )
                        (strictify< (first(rest pair))
                                (cons <  (first(rest pair))
                                         (frons < (first pair)(rest(rest pair)) > )
                                > ) > ) > ) > ) )) .
```

9.3. *Amb* **in terms of** *Arbit*

```
amb ≈ (lambda pair
        (((if (arbit pair)) (first pair)) (first(rest pair)) )) .
```

Proof:

$$
\begin{aligned}
&\mathcal{E}\,[\![\textbf{(lambda pair (((if (arbit pair)) (first pair)) (first(rest pair))))}]\!]\rho = \\
&\quad = (\lambda\delta.\mathcal{E}\,[\![\ (((\textbf{if (arbit pair)) (first pair))} \\
&\qquad\qquad \textbf{(first(rest pair))))}\ ]\!]\rho[\delta/\textbf{pair}]\ \textbf{in E}\ ; \\
&\quad = (\lambda\delta.(\ (\mathcal{E}\,[\![\ ((\textbf{if (arbit pair)) (first pair))}]\!]\rho[\delta/\textbf{pair}]|\textbf{S}) \\
&\qquad\qquad (\delta|\textbf{G}|\textbf{M}|\textbf{T}{\downarrow}2|\textbf{T}{\downarrow}1\ \textbf{in D})\ ))\ \textbf{in E}\ ; \\
&\quad = (\lambda\delta.((\ (\mathcal{E}\,[\![\textbf{(if (arbit pair))}]\!]\rho[\delta/\textbf{pair}]|\textbf{S}) \\
&\qquad\qquad (\delta|\textbf{G}|\textbf{M}|\textbf{T}{\downarrow}1\ \textbf{in D})\ ) \\
&\qquad (\delta|\textbf{G}|\textbf{M}|\textbf{T}{\downarrow}2|\textbf{T}{\downarrow}1\ \textbf{in D})\ ))\ \textbf{in E}\ ; \\
&\quad = (\lambda\delta.((\ \bigsqcup_{\gamma\,\in\,(\mathcal{E}\,[\![\textbf{(arbit pair)}]\!]\rho[\delta/\textbf{pair}]|\mathcal{F}_{BM}(G)} \\
&\qquad (\ \gamma\,|\textbf{A} = \textbf{true} \rightarrow\ (\lambda\delta_1.((\lambda\delta_2.\nabla\ \delta_1)\textbf{in E})), \\
&\qquad (\ \gamma\,|\textbf{A} = \textbf{false} \rightarrow\ (\lambda\delta_1.((\lambda\delta_2.\nabla\ \delta_2)\textbf{in E})), \\
&\qquad \perp_S\ )) \\
&\qquad\qquad (\delta|\textbf{G}|\textbf{M}|\textbf{T}{\downarrow}1\ \textbf{in D})\ ) \\
&\qquad\qquad (\delta|\textbf{G}|\textbf{M}|\textbf{T}{\downarrow}2|\textbf{T}{\downarrow}1\ \textbf{in D})\ ))\ \textbf{in E}\ ; \\
&\quad = (\lambda\delta.((\ \bigsqcup_{\gamma\,\in\,\{(\delta|\textbf{G}|\textbf{M}|\textbf{T}{\downarrow}1\,=\,\perp_G\,\rightarrow\,\perp_G,\ \textbf{true in G}),\ (\delta|\textbf{G}|\textbf{M}|\textbf{T}{\downarrow}2|\textbf{T}{\downarrow}1\,=\,\perp_G\,\rightarrow\,\perp_G,\ \textbf{false in G})\}} \\
&\qquad (\ \gamma\,|\textbf{A} = \textbf{true} \rightarrow\ (\lambda\delta_1.((\lambda\delta_2.\nabla\ \delta_1)\textbf{in E})), \\
&\qquad (\ \gamma\,|\textbf{A} = \textbf{false} \rightarrow\ (\lambda\delta_1.((\lambda\delta_2.\nabla\ \delta_2)\textbf{in E})), \\
&\qquad \perp_S\ )) \\
&\qquad\qquad (\delta|\textbf{G}|\textbf{M}|\textbf{T}{\downarrow}1\ \textbf{in D})\ ) \\
&\qquad\qquad (\delta|\textbf{G}|\textbf{M}|\textbf{T}{\downarrow}2|\textbf{T}{\downarrow}1\ \textbf{in D})\ ))\ \textbf{in E}\ ; \\
&\quad = (\lambda\delta.((\ \{(\delta|\textbf{G}|\textbf{M}|\textbf{T}{\downarrow}1\ \ = \perp_G \rightarrow \perp_G,\ \delta|\textbf{G}|\textbf{M}|\textbf{T}{\downarrow}1), \\
&\qquad\qquad (\delta|\textbf{G}|\textbf{M}|\textbf{T}{\downarrow}2|\textbf{T}{\downarrow}1 = \perp_G \rightarrow \perp_G,\ \delta|\textbf{G}|\textbf{M}|\textbf{T}{\downarrow}2|\textbf{T}{\downarrow}1),\ \perp_G\}\ \textbf{in E}) \sqcup\ \perp_E))\ \textbf{in E}; \\
&\quad = (\lambda\delta.(\ \{(\delta|\textbf{G}|\textbf{M}|\textbf{T}{\downarrow}1,\ \delta|\textbf{G}|\textbf{M}|\textbf{T}{\downarrow}2|\textbf{T}{\downarrow}1\}\ \textbf{in E}))\ \textbf{in E}; \\
&\quad = \nabla\,\mathcal{K}\,[\![\textbf{amb}]\!]\ ; \\
&\quad = \mathcal{E}\,[\![\textbf{amb}]\!]\rho\ .
\end{aligned}
$$

□

9.4. *Arbit* **in terms of** *Arbiter*

```
arbit ≈ (lambda pair
        (first(rest(first (arbiter << (first pairs)> < (first(rest pairs))> > ))) ) .
```

9.5. *Arbiter* **in terms of** *Frons*

```
(letrec arbiter (lambda pairstreams
        (merge < (tagTRUE (first pairstreams)) (tagFALSE (first(rest pairstreams))) > )) ) ,
```

where each instance of tagBOOL is macro-expanded according to the pattern:

```
tagBOOL ≡ (letrec tagBOOL (lambda stream
              (strictify <(first stream)
                         (cons <<(first stream) bool> (tagBOOL (rest stream)) >) >) ))
```

for the appropriately consistent boolean value of BOOL and bool. The idea is to attach a boolean tag to each element in the stream forming a stream of records that contain the original elements and the tag associated with this stream. Every record in the resulting stream of records is strict in its contained element.

10. Conclusions

Several conclusions arise from this paper that should become principles of indeterminate programming languages. A few hints and observations will steer designers around some nasty traps; at least one common misconception must be set aside.

10.1. Lessons for Indeterminate Language Design

Call-by-name is not a safe evaluation rule. The correct perspective in indeterminate systems is environmental transparency, which maintains that an identifier is bound to a unique denoted value in any environment. In every semantics herein, power domains are directly excluded from environments. (They are indirectly available through application of a bound function, however, under the philosophy that application already consumes resources and may introduce indeterminism, but merely discharging a binding should not and cannot.)

Environmental transparency prompts the distinction between **D** and **E** [4]. It is important not to confuse their roles. **D** contains the chunks that are the objects of bindings, and hence is the domain of system-function parameters. **E** contains the sets that represent uncertainty or indeterminism, and hence is the domain of system-function results. Since one must maintain the integrity of an identifier bound to a δ, one cannot substitute more than once ϵ (and a choice of a second, independent δ') for that identifier. Such independent choice occurs under call-by-name where ϵ is reevaluated (likely to the same value) and a choice of δ is extracted (likely a different one) at every discharge of the bound identifier. The strategy is safe in deterministic programming because there is no choice.

Under indeterminate programming, however, Lambda calculus's straightforward β-*substitution fails*. Syntactic substitution of argument for parameter is trouble. String reduction is unsatisfactory; graph reduction is needed.

The distinction between **D** and **E** in turn prompts all the other results herein. The known technique of natural extension of application effects the binding of indentifiers to simple values in **D**. Finding a reasonable interpretation for extracting fixed points from $[\mathbf{D} \to \mathbf{E}]$ leads to definition of the $\lessdot$ relation.

An indeterminate function $\pi \in [\mathbf{G} \to \boldsymbol{P}_{\mathrm{EM}}(\mathbf{G})]$ *stands for a collection of functions on the underlying ground domain* **G**:

$$\pi \simeq \{\phi \in [\mathbf{D} \to \mathbf{D}] \mid \phi \lessdot \pi\} .$$

This perception holds because of the way that indeterminate functions (called *systems* here) are used. In spite of reflexivity of domains, such a system is never the output of a real operating system. (We would never agree on the way a function should be displayed on output anyway.) The only use for such systems, aside from being passed around as bound values, is for use in application and computing fixed points. For both these uses it is sufficient to perceive π as a composition of ϕ's.

Indeterminate functions should be excluded from indeterminate data structures. To allow systems within the structures of Sections 5 and 6 introduces an unnecessary implementation cost from breadth-first or depth-first choice among function objects, choice which propogates through the application of these objects or corrupts the interpretation of taking fixed points. (This is required by the definition of join on the function domain $[\mathbf{D} \to \mathbf{E}]$; it distributes through lambda-abstraction.) Structures are supposed to be cheap! The additional cost is unnecessary because the indeterminacy of the entire data structure can be subsumed into a single function, with a

pleasing reduction in the conceptual baggage that a programmer must carry; programmers already deal with programs as simple values. They would understand indeterminate choice from a set of ground values much more easily than they would accept indeterminate choice of programs. ("Choose any of these square-root routines and....")

Section 8 presents rotes, a second kind of data structure that allows systems-functions in but keeps indeterminism out. This type distinction is easier to implement; it has been used in deterministic programming practice for some time (*e.g.* in LISP). The new development is the earlier multiset structure that allows indeterminate content, $\mathbf{P}(G)$, but prohibits programs.

10.2. Open Issues

The techniques used here are quite general. Within the original constraints they provide a facile tool for designing semantics of powerful programming languages for synchronous and asynchronous multiprocessors. With streams representating communication pipes, they extend nicely to communicating processes.

There are, nevertheless, two constraints and a loose end. First, and of lesser practical significance, is the fact that the languages here are not strongly typed. It is possible to compose a function in S that returns a system, σ, for some arguments and returns an element from the power domain, $\mathbf{P}_{EM}(G)$, for others. The application axiom can call for the join of such incompatible results, and this has not been hendled well here. The introduction of $\top_E$ through use of the $\boxplus$ domain sum is less than satisfactory. A typing system is needed for this kind of domain structure that can be implemented as part of the language.

Second, and very important, is reconciliation of fairness with the semantics here. A direction is indicated by these results. Section 8 shows how to specify determinate data structures of a very general sort, called rotes; these data structures may contain indeterminate functions (systems), and may themselves be of infinite length. The original data structure, a multiset, is allowed to participate in indeterminate "contention" within a power domain. As defined here it, too, may be infinite.

Infinite length of multisets (as of streams [13]) is a luxury that will never be implemented in practice, because fairness within a powerdomain element becomes unwieldy over infinite ground elements (and even impossible with bounded fairness). It will turn out, however, that only finite data structures are needed to model contending processes, that multisets will be perfectly useful even if they must be finite. With that restriction, every ground element becomes finite and the impressed power domain is much simplified. With ground elements finite, the problems of specifying and implementing fairness suddenly become tractable.

Bottom-avoidance, as desired in Section 7.2, also might be impossible unless the ground domain is restricted to finite values. The example of that section shows that bottom-avoidance (of $\perp_T$ there) is nearly possible after using $frons_3$, which is an approximation to *frons* sufficiently large to detect that each permutation is finite (that μ=**nil**). That is, the fact that the recursive expression generates a set of finite ground values is there necessary merely to argue bottom-avoidance.

Finally, one should note the inadequacies of the two power domains used here, and recognize that yet another formulation for the power domain, itself, might improve these results. The Plotkin power domain, $\mathbf{P}_{EM}(G)$, seems inadequate because Conjecture 3 depends on an existential argument. The Smyth-upside-down domain provides clear, intuitive, and constructive proofs, but the downward closure of all its elements distorts the results that we desire. In both cases we can wish for more explicit "bottom avoidance" in denotational semantics than is now provided operationally [6].

Acknowledgements:

Will Clinger suffered through many wishful (and incorrect) attempts at this result. He directed me toward viable strategies, and endured my cherished hopes well. Aside from mathematics credited to him herein, the domain constructions, separating **D** from **E**, reflect [4] and joint research. This work could certainly not exist without him; it is a privilege to have him as a neighbor. Extension of the original formulation in the Smyth-upside-down domain to the Plotkin domain would not have happened without remarks and encouragement from Glynn Winskel and Stephen Brookes; I thank them.

Over the months others have listened and offered suggestions: David MacQueen posed the motivating problem years ago. Steve Johnson, Mitchell Wand, and Dan Friedman can also recognize their influence herein. Research reported herein was supported by the National Science Foundation under grant number MCS82-03978. Tektronix, Inc., graciously provided facilities for production of this revision.

References

1. Manfred Broy. A theory for nondeterminism, parallelism, communication, and concurrency. Institut fur Informatik, Technische Universitat Munchen (1983).

2. Robert Cartwright and James Donahue. The semantics of lazy (and industrious) evaluations, *Conf. Rec. ACM Symp. on LISP and Functional Programming*, ACM Order No. 552820 (1982), 253-264.

3. William Clinger. *Foundations of Actor Semantics*, Ph.D. dissertation, Massachusetts Institute of Technology (1981). Also MIT Artificial Intelligence Technical Report 633.

4. William Clinger. Nondeterministic Call by Need is Neither Lazy Nor by Name. *Conf. Rec. ACM Symp. on LISP and Functional Programming*, ACM Order No. 552820, 226-234.

5. Martin Davis. *Computability and Unsolvability*, McGraw-Hill, New York (1958), 44.

6. Daniel P. Friedman and David S. Wise. An approach to fair applicative multiprogramming. In G. Kahn (ed.), *Semantics of Concurrent Computation*, Springer, Berlin (1979), 203-225.

7. Daniel P. Friedman and David S. Wise. An indeterminate constructor for applicative programming. *Conf. Record 7th ACM Symp. on Principles of Programming Languages* (Jan., 1980), 245-250.

8. G. Gierz, K.H. Hofmann, K. Keimel, J.D. Lawson, M. Mislove, and D.S. Scott. *A Compendium of Continous Lattices*, Springer, Berlin (1980), 2.

9. M.C.B. Hennessy and Gordon D. Plotkin. Full Abstraction for a simple parallel programming language. In J. Bečcàř (ed.), *Mathematical Foundations of Computer Science 1979*, Springer, Berlin (1979), 108-120.

10. Robert Keller. Denotational models for parallel programs with indeterminate operators. In E.J. Neuhold (ed.), *Formal Description of Programming Language Concepts*, North-Holland, Amsterdam (1978), 337-366.

11. Donald E. Knuth. *The Art of Computer Programming* **I**, Fundamental Algorithms, Addison Wesley, Reading, MA (1973), Section 1.2.5, Eqn. 9.

12. Paul Kosinski. A data flow language for operating systems programming. *Proc. ACM*

SIGPLAN-SIGOPS Interface Meeting, SIGPLAN Notices **8**, 9 (September, 1973), 89-94.

13. Peter Landin. A correspondence between ALGOL 60 and Church's lambda notation. *Comm. ACM* **8**, 2 (August, 1965), 89-101.

14. John McCarthy, A basis for a mathematical theory of computation. In P. Braffort and D. Hirschberg (eds.), *Computer Programming and Formal Systems*, North-Holland, Amsterdam (1963), 33-70.

15. Robin Milner. A theory of type polymorphism in programming. *J. Comp. Sys. Sci.* **17** (1978), 348-375.

16. Gordon Plotkin. A powerdomain construction. *SIAM J. Comput.* **5** (1976), 452-487.

17. Willard V.O. Quine. *Word and Object*, MIT Press, Cambridge, MA (1960), Section 30.

18. Dana Scott. Domains for denotational semantics, *Proc. 9th Intl. Colloq. on Automata, Languages, and Programming*, Aarhus (revised version dated 1 July 1982).

19. Guy L. Steele, Jr. *Common Lisp Reference Manual*, Excelsior Edition, Spice Project, Computer Science Department, Carnegie-Mellon University, Pittsburgh (August, 1983). To be published in 1984 by Digital Press, Maynard, MA.

20. Michael Smyth. Power domains. *J. Comput. System Sci.* **5** (1977), 257-274.

21. Joseph E. Stoy. *Denotational Semantics: the Scott-Strachey Approach to Programming Language Theory*, The MIT Press, Cambridge, MA (1977).

22. Alfred N. Whitehead and Bertrand Russell. *Principia Mathematica* **I**, *2nd edition*, Cambridge University Press (1927), 665.