SOME ORDERED SETS IN COMPUTER SCIENCE

Dana S. Scott
Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

ABSTRACT

Strictly speaking the structures to be used are not lattices since as posets they will lack the top (or unit) element, but the adjunction of a top will make them complete lattices. The closure properties as posets, then, are closure under *inf* of any non-empty subset and *sup* of directed subsets. A family of subsets of a set closed under *intersections* of non-empty subfamilies and *unions* of directed subfamilies is a special type of poset with the closure properties where additionally every element is the directed sup (union) of the finite ("compact") elements it contains. We call such posets *finitary domains*. (With a top they are just the well known algebraic lattices.) The *continuous domains* can be defined as the continuous *retracts* of finitary domains. A mapping between domains is *continuous* if it preserves direct sups. A map of a domain into itself is a *retraction* if it is idempotent. Starting with a finitary domain, the range (= fixed-point set) of a continuous retraction -- as a poset -- is a continuous domain. Numberless characterizations of continuous domains, both topological and order-theoretic, can be found in [2]. For the most part in the lectures we shall concentrate on the finitary domains, but the continuous domains find an interest as a generalization of *interval analysis* and by the connection with spaces of *upper-semicontinuous* functions.

In [10] the presentation of the theory of finitary domains is made especially elementary by considering them as filter completions of *neighborhood systems*. Up to isomorphism every finitary domain can be represented this way, and the use of neighborhoods makes the definitions of notions and the constructions of other domains very explicit. This is helpful in discussing questions of *computability* and of *effective presentation* of domains.

I. Rival (ed.), Ordered Sets, 677–718.

The primary mathematical feature of the theory of domains is the fact that the category of finitary domains and continuous mappings is *cartesian closed*; that is, the category has products and function spaces and has a natural isomorphism

$$X^{(Y \times Z)} \cong (X^Y)^Z .$$

There are several important sub-cartesian-closed categories: the *countably based domains* (= completion of a countable neighborhood system) with continuous maps, and the effectively presented domains with *computable* maps. These categories are not like the category of sets, since every continuous self-map on a finitary domain has a fixed point; indeed, on any finitary domain X there is a continuous fixed-point operator

$$\text{fix} : X^X \to X .$$

A notation for maps in a cartesian closed category with fixed-point operators is a rudimentary programming language: it is a *typed λ-calculus with recursion* (expressed by fixed points). Connections with recursion theory are discussed in [6].

More elaborate languages can be given a *denotational semantics* by construction of special effectively presented domains. For example a domain D where

$$D \cong D \times D \cong D^D$$

can be used to give meanings to the terms of an *untyped λ-calculus*, see [1] and [11] for details and [7] for an introduction. Additional expository discussion is found in [8] and [9]. Other examples of semantical definitions, more appropriate to computer science, and of solutions to domain equations will be given. Proof systems as in [3] will also be discussed.

There are many difficult combinatorial problems about (finite) ordered sets that raise serious problems for computing both of a theoretical and a practical nature. We shall not consider such problems here; the reader can find ample examples and explanations elsewhere in this volume. Rather, we shall discuss how partially ordered sets -- even ones of the cardinality of the continuum -- can be used for the conceptual organization of a plan of how to give mathematical meaning to many constructs in higher-level computer languages. The story has been told before (cf. [11], [3], [4], and [7]), but recently I have found a way of making the details quite a bit more elementary. A fuller presentation is contained in [10], which I hope can be expanded soon into a book; the lecture notes can also be consulted for more examples and many exercises.

Even though we propose working with non-denumerable ordered sets, the constructive and computable nature of the study is kept firmly in hand by restricting attention to those posets that can be obtained as "completions" of certain countable partial semi-lattices that can be computably described (*via* recursive presentations, for example). This is true of the real numbers, but they live in a category of *totally* ordered sets; it was found that for many reasons it is more fruitful to work in a category of posets. Instead of giving the abstract definitions all at once, we begin with some examples that are meant to convince the reader of the reasonableness of the final choice of a category. The choice to be described here is not the only one, but it is a very simple one with many familiar properties; it might even be argued that it is a minimal choice. But we leave such discussions for another time.

It is often suggested that I discovered continuous lattices as a consequence of my search for models of the Church-Curry λ-calculus. But, as was explained in [2], it was exactly the other way 'round: *after* I discovered the proper category, I realized that there was enough "elbow-room" to have models for λ-calculus. I had indeed many times denied there was good mathematical content to λ-calculus, and I was searching for an *alternative*. As I argue in [8], however, the search was very 'round about and could have taken care of λ-calculus much sooner. Be that as it may, the topological and categorical import of the initial idea is now well understood (cf. [2]), and the book [1] of Barendregt ties up the models well with the formal theories (though, perhaps, the last word has not quite been said). In [9] I indicate that the so-called *categorical logic* might also throw

some light on the matter, since any category whatsoever can be used as a *site*, or a basis, for the construction of more grandiose categories (called *topoi*) which have more satisfactory logical properties (more closure conditions, for instance). Whether this is a good idea remains to be seen, and we have no space to discuss it further now.

1. MOTIVATING EXAMPLES

The place where I actually started my investigations was from ideas in recursion theory with a very common-place structure: the partial functions from numbers to numbers. Let $\mathbb{N}$ be the set of natural numbers, and contrast

$$f : \mathbb{N} \to \mathbb{N} \quad \text{with} \quad g : \mathbb{N} \rightharpoonup \mathbb{N} .$$

The first property means that f is a *total* (everywhere defined) function; while the second means that g is only a *partial* (partially defined) function from $\mathbb{N}$ into $\mathbb{N}$ (possibly even nowhere defined). Turning now to posets we have:

EXAMPLE 1. The set $(\mathbb{N} \rightharpoonup \mathbb{N})$ of all partial number-theoretic functions is partially ordered by the relation:

$$f \subseteq g \quad \text{iff whenever } f(n) \text{ is defined, then so is } g(n) \text{ defined and } f(n) = g(n).$$

It is permissible to use the symbol "$\subseteq$" here, since -- as everyone can see -- the relation defined is the same as that of *inclusion between the graphs* of the functions regarded as sets of ordered pairs of numbers in the standard way. (Obviously in this example we could have used other sets aside from $\mathbb{N}$.) Note that the total functions are just the *maximal* elements of the poset $(\mathbb{N} \rightharpoonup \mathbb{N})$. □

Functions in $(\mathbb{N} \rightharpoonup \mathbb{N})$ are mappings between numbers, but there are as well many important *mappings between functions*. Let us call these *functionals*, or operators. For instance, if "$\circ$" denotes *composition* of functions (and it works in a natural way for partial functions), then the correspondence, say,

$$f \mapsto a \circ f \circ b \circ f ,$$

where a and b are fixed functions, is a good example of a functional. Let us write

$$F(f) = a \circ f \circ b \circ f$$

for all $f \in (\mathbb{N} \rightharpoonup \mathbb{N})$. Now $(\mathbb{N} \rightharpoonup \mathbb{N})$ is a set, and F is well

defined on *all* arguments in this set; we can thus write in the set-theoretical sense that

$$F : (\mathbb{N} \rightharpoonup \mathbb{N}) \to (\mathbb{N} \rightharpoonup \mathbb{N}) .$$

But F is hardly an arbitrary set-theoretical mapping; in fact, if a,b,f are computable (say, partial recursive), then so is $F(f)$. The question is whether F has some essential order-theoretic property that might characterize "good" functionals.

The hint about computability is not misleading, because it is rather straightforward to argue that computable functionals are "continuous", and that continuity is an easily expressible order-theoretic property. This was known for a very long time in recursion theory. We simply remark that to compute $F(f)(n)$ at an argument n we need only compute f at a *finite number* of well-determined arguments (namely, $f(n)$ and $f(b(f(n)))$, and then the answer comes by feeding into a this last number). Now $F(f)(n)$ may not be defined -- it all depends upon a chain of events involving places where a, b, and f are defined -- but when it is we can indeed find the answer. In the choice of the definition of this F we generally had to compute *two* values of f for each value of $F(f)$, but the reader can find for himself other functionals that demand the computation of several values (with the exact number even depending upon n).

Setting aside worries about computability and focusing instead just on questions of dependency, we can say that a functional F is *continuous* iff for $f \in (\mathbb{N} \rightharpoonup \mathbb{N})$ and $n \in \mathbb{N}$, the value of $F(f)(n)$ only depends on some finite number $f(k_0),\ldots,f(k_{m-1})$ of values of f. (That is, if g agreed with f at these arguments, then $F(f)(n) = F(g)(n)$, provided that $F(f)(n)$ is defined.) This does not *look* like an order-theoretic property, but it is. We recall that in the poset $(\mathbb{N} \rightharpoonup \mathbb{N})$ every function f is the *union* of the finite functions (functions with a finite graph) it contains. We can even write

$$f = \bigcup_{n=0}^{\infty} f_n ,$$

where f_n is the restriction of f to $\{0,1,\ldots,n-1\}$. In this representation we find that

$$f_0 \subseteq f_1 \subseteq f_2 \subseteq \cdots \subseteq f_n \subseteq f_{n+1} \subseteq \cdots ,$$

so that the "approximations" f_n form a tower. There are many possible towers; there are lots of towers where the f_n are not

even finite. Continuous functionals F, however, always "respect" towers in the sense that

$$(*) \qquad F\left(\bigcup_{n=0}^{\infty} f_n\right) = \bigcup_{n=0}^{\infty} F(f_n) \ .$$

A small amount of thought will convince the reader that conversely, every such functional is continuous. Thus, equation (*), for all towers, is an order-theoretic definition of continuity for functionals on $(\mathbb{N} \to \mathbb{N})$. (This notion is in fact topological continuity in a suitable topology, as will be explained below.)

It is easy to find other examples of continuous functionals and to find examples of other similar domains for such functions. It is also easy to extend the definition of continuity to functionals of several variables. A nice example, which is also well known in semigroup theory is:

EXAMPLE 2. The set $\mathbb{P}$ of all partial permutations of $\mathbb{N}$ (that is, one-one functions on subsets of $\mathbb{N}$ into $\mathbb{N}$) is a subposet of $(\mathbb{N} \to \mathbb{N})$ on which composition and inverse are continuous operations. □

Regarding a set, better a subset of $\mathbb{N}$, as a partial function from $\mathbb{N}$ to a one-element set, we have similarly:

EXAMPLE 3. The set $\mathbb{S}$ of subsets of $\mathbb{N}$ is a poset under ordinary inclusion where the operations of union, $\bigcup$, and intersection, $\bigcap$, are continuous. □

Note that complementation is *not* continuous, because it follows from (*) that all continuous functionals are monotone:

$$(**) \qquad f \subseteq g \text{ always implies } F(f) \subseteq F(g) \ .$$

Clearly set complementation is not even monotone. (Property (**), by the way, does not imply (*).) If a continuous complementation operation is desired, then a different poset should be investigated: the partial functions from $\mathbb{N}$ into a *two-element* set. Just as $\mathbb{P}$ is not quite a group, this new example is not quite a Boolean algebra -- but it is none the worse for that.

Another example of a different kind which was very suggestive to me was:

EXAMPLE 4. The set $\mathbb{F}$ of all continuous functionals from $(\mathbb{N} \to \mathbb{N})$ into $(\mathbb{N} \to \mathbb{N})$ is a poset under the relation:

$$F \subseteq G \quad \text{iff} \quad F(f) \subseteq G(f) \ , \quad \text{for all } f \in (\mathbb{N} \to \mathbb{N}) \ .$$

Under this definition least upper bounds of towers exist in $\mathbb{F}$, and an operation like the passage from $a \in (\mathbb{N} \to \mathbb{N})$ to the functional

$$f \mapsto a \circ f$$

is a continuous operator from $(\mathbb{N} \to \mathbb{N})$ into $\mathbb{F}$. An example of a continuous operator going in the other direction is the *least fixed-point* operator defined for $F \in \mathbb{F}$ by

$$\mathrm{fix}(F) = \bigcup_{n=0}^{\infty} F^n(\perp) \ ,$$

where $\perp$ is the empty function (or least element of $(\mathbb{N} \to \mathbb{N})$, and where F^n is the n-fold composition of F with itself. Using the continuity of F, it can be shown by a standard argument that $f = \mathrm{fix}(F)$ is the least function in the poset $(\mathbb{N} \to \mathbb{N})$ such that $f = F(f)$. What is being stressed now, however, is not so much that this least f exists, but that the whole operator fix is itself continuous. □

We are not stopping to verify all the (easy) assertions in these examples. They should be immediately plausible, however, and they should indicate that there is a starting place for a more general theory here. The question to be answered is: what is common among these examples? and what kind of poset structure should be abstracted from them?

2. CLOSURE SYSTEMS AND DOMAINS

Taking, as we have already done, a number-theoretic function as a set of ordered pairs, then the poset $(\mathbb{N} \to \mathbb{N})$ *is* a family of sets partially ordered by inclusion between sets. Of course, any family of sets is a poset in this way, but $(\mathbb{N} \to \mathbb{N})$ has further closure properties. For one, it is closed under taking *subsets*; BUT this will not be a property to be abstracted from examples, because this is not an order-theoretic property. What do prove to be useful properties are singled out in this definition:

DEFINITION. A *closure system* is a non-empty family of sets closed under intersections of arbitrary non-empty subfamilies and unions of towers. A poset isomorphic to a closure system I call a *domain*. □

The definition requires a few comments. In general towers have to be taken as transfinite towers of any ordinal length, but we will work with families where all the sets are contained in one countable set. In this case simple infinite chains are sufficient, and the worries about full generality are unnecessary. (Actually, even in general it is better to take the closure

condition as employing unions of *directed* subfamilies and to forget about linearly ordered towers.) A closure system by my definition has a *least element* $\perp$, which is just the intersection of the whole family of sets. Often $\perp$ is the empty set $\emptyset$, but this is not necessarily so, and it is not being assumed. It is usual also to have a *greatest element* (which therefore contains all other sets as subsets), but this is definitely not being assumed. A closure system by this definition is not a directed family of sets; two sets in the system need not be included in a common set and are thus "inconsistent" with each other. We will discuss consistency more below.

It should be quite clear that $(\mathcal{N} \to \mathcal{N})$ is a closure system as is $\mathcal{P}$ and, of course $\mathcal{S}$. The question about $\mathcal{F}$ is not quite so obvious and is one of the main points of the theory. If a family of sets is defined by "closure conditions", it is a closure system. What is meant by closure conditions? There are several types. First we have:

(a) $$k \in x \ ,$$

where k is a given constant. Then we have:

(b) $$i,j,\ldots \in x \text{ always implies } \sigma(i,j,\ldots) \in x \ ,$$

where σ is a given operation under which the set x is to be closed. Next we have:

(c) $$i,j,\ldots \in x \text{ always implies } \Pi(i,j,\ldots) \ ,$$

where Π is a property of the $i,j,\ldots$ not mentioning the set x. These are the usual types of closure properties (strictly speaking (c) is more of a consistency condition), but we could also combine (b) and (c) into:

(d) $$i,j,\ldots \in x \text{ always implies either } \Pi(i,j,\ldots) \text{ or } \sigma(i,j,\ldots) \in x \ ,$$

which means x is closed under σ for sequences $i,j,\ldots$ *outside* Π. More generally we can take as a closure condition any universally quantified property of the set x made by a finite *disjunction* of clauses:

$$\Pi(i,j,\ldots),\ \tau(i,j,\ldots) \notin x,\ \sigma(i,j,\ldots) \in x \ ,$$

for various properties Π, functions τ and σ, PROVIDED there is *at most one* of the third kind (that is, the positive atomic clause involving membership in x). Any (infinite) conjunction of closure conditions defines a closure system -- provided there is

at least one set satisfying all of them. Many, many examples of closure systems can now be produced even with simple choices of clauses.

Let us return for a moment to the question of consistency. Let $\mathcal{C}$ be a closure system. We can assume for the sake of argument that $\mathcal{N}$ is the underlying set, so that $\mathcal{C}$ is a family of subsets of $\mathcal{N}$. Not every set $x \subseteq \mathcal{N}$ is consistent relative to $\mathcal{C}$, however. *$\mathcal{C}$-consistency* means simply that $x \subseteq y$ for some $y \in \mathcal{C}$; that is, x can be extended to a closed set. Every $\mathcal{C}$-consistent set has a *closure* defined in the obvious way as the least closed extension:

$$\bar{x} = \bigcap \{y \in \mathcal{C} \mid x \subseteq y\} \ .$$

This kind of closure operation has nice properties, and we can use them to characterize closure systems in the well-known way. All that need be kept in mind over the usual simple examples is that we are not assuming that all subsets of the underlying set are consistent.

DEFINITION. A *consistency system* is a non-empty family of sets closed under taking subsets and unions of towers (or: directed unions). □

The $\mathcal{C}$-consistent sets, call this the family $\mathcal{C}_0$ for short, form a consistency system. But $\mathcal{C}_0$ is far too weak a notion to determine $\mathcal{C}$. We need the closure operation $\bar{}: \mathcal{C}_0 \to \mathcal{C}_0$ so as to define $\mathcal{C}$ as the "closed" elements:

$$\mathcal{C} = \{x \in \mathcal{C}_0 \mid x = \bar{x}\} \ .$$

For instance, $\bar{}$ might be something clever and $\mathcal{C}_0$ might be something boring like the power set of $\mathcal{N}$. Indeed many different closure systems can have the same consistency system. We definitely have to know the closure operator to recapture $\mathcal{C}$ from $\mathcal{C}_0$. What are the properties? Just the familiar ones:

DEFINITION. A *closure operator* is a function $\bar{}$ defined on a consistency system $\mathcal{C}_0$ so that:

(i) $x \subseteq \bar{x} = \bar{\bar{x}}$ for all $x \in \mathcal{C}_0$;

(ii) $\bar{}$ commutes with unions of towers contained in $\mathcal{C}_0$.

As a consequence, every closure operator is monotone:

(iii) $x \subseteq y$ implies $\bar{x} \subseteq \bar{y}$ for $x,y \in \mathcal{C}_0$.

Or in words a closure operator is an inflationary, idempotent, continuous function on a consistency system. □

Condition (ii) is usually described as requiring $^-$ to be an *algebraic* closure operator rather than a *topological* closure, which commutes with finite unions and does not satisfy (ii).

FOLK THEOREM. *There is an (obvious) one-to-one correspondence between closure systems and closure operators.* □

The only point that might be a trifle nonstandard in our discussion is the emphasis on consistency systems, since it is more common to work in the power set of a set as the consistency system. The generalization can be seen as a natural one, however, especially when it is considered how consistency systems relate to finite sets.

PROPOSITION. *Let $\mathcal{C}_0$ be a consistency system and let $\mathcal{C}_{00}$ be the family of finite sets in $\mathcal{C}_0$. Then $\mathcal{C}_{00}$ determines $\mathcal{C}_0$, since a set x is in $\mathcal{C}_0$ just in case the family of finite subsets of x is included in $\mathcal{C}_{00}$. Moreover, every non-empty family of finite sets closed under taking subsets is a $\mathcal{C}_0$ for some consistency system $\mathcal{C}_0$.* □

The proof of the proposition is clear. As a direct consequence we can also see that $\mathcal{C}_0$ is a consistency system if and only if it has the property that a set belong to $\mathcal{C}_0$ just in case all its finite subsets belong to $\mathcal{C}_0$. (This assumes that $\mathcal{C}_0$ is non-empty or, equivalently, that $\emptyset \in C_0$.) In other words consistency systems are families of sets of "finite character".

The finite (consistent) sets are also interesting for the closure operator: every closed set is the union of the closures of its finite subsets. We call a set $x \in \mathcal{C}$ which is the closure of a finite subset a *finitely generated* element, or just a finite element of $\mathcal{C}$ for short. It is not difficult to characterize the finite elements of $\mathcal{C}$ by a purely order-theoretic property. Neither is it difficult to characterize exactly the type of poset the set of finite elements of a closure system is, again in purely order-theoretic terms. We allude to such a characterization in the next section. We do not stress here these order-theoretic questions, since the main purpose of the presentation is to show that there is a multitude of easily obtained closure systems (domains). What we wish to do next is to make a category out of the class of domains and to discuss construction principles in this category.

Before turning to more general matters, however, note that $\mathcal{N}$ could be regarded as (almost) a closure system -- in a straightforward and trivial sense. Instead of $\mathcal{N}$ use

$$\check{\mathcal{N}} = \{\{n\} \mid n \in \mathcal{N}\} \cup \{\emptyset\} .$$

This is a very boring closure system from the order-theoretic point of view. Note that $(\check{\mathcal{N}} \to \mathcal{N})$ can be construed (up to order-theoretic isomorphism) as the domain of continuous functions $f : \check{\mathcal{N}} \to \mathcal{N}$ where $f(\perp) = \perp$. (These are often called *strict* functions.) In this way we can keep all our entities in the same category. A less trivial question concerns the totality of continuous functions on an arbitrary domain. We will give a complete analysis.

3. NEIGHBORHOOD SYSTEMS AND THE REPRESENTATION OF DOMAINS

Closure systems are actually so familiar that it would be possible to work entirely with them and to form the desired category that way. But it recently occurred to me that a suitable form of neighborhood systems is rather more elementary and suggestive, making pictures easier to draw for illustrating many notions and constructs. All I shall actually do is give another set-theoretical *representation* of closure systems, but it is one that puts the very useful finite elements more on the surface.

DEFINITION. Let Δ be a given non-empty set. A *neighborhood system* over Δ is a family $\mathcal{D}$ of subsets of Δ where

(i) $\Delta \in \mathcal{D}$; and

(ii) whenever $X, Y, Z \in \mathcal{D}$ and $Z \subseteq X \cap Y$, then $X \cap Y \in \mathcal{D}$.

A *finite* collection of neighborhoods is *consistent* (in $\mathcal{D}$) if the intersection contains another neighborhood. An *arbitrary* collection is *consistent* just in case every finite subcollection is. □

The purpose of the theory of domains is to have a suitable notion of partial element generalizing the way partial functions behave in $(\mathcal{N} \to \mathcal{N})$. A number of reasons why such a notion is to be regarded as "necessary" will be discussed in due course. The intuition behind neighborhood systems is this: think of the elements of Δ as "tokens" or rough examples of (some of) the desired partial elements. The job of a neighborhood is to collect together all of the tokens sharing the same (finite) amount of information. A token $t \in \Delta$ may not be "consistent", however, because we do not assume that

$$\{X \in \mathcal{D} \mid t \in X\}$$

is a consistent set of neighborhoods. The "message" that t carries could refer to several quite different elements. For the structure of the domain the exact form of the tokens is not really important; we shall see in several ways that one domain can have many representations. The poset structure of $\mathcal{D}$ is what we regard as important. We could, of course, axiomatize the kind of semilattice structure we need; but it is so easy to represent such (partial) semilattices as neighborhood systems, the axioms are not especially needed. Surely the reader has to admit that he knows dozens of neighborhood systems made from familiar mathematical ingredients.

We have spoken about "elements" without defining precisely what we want. We remarked that the elements of the domain to be defined by $\mathcal{D}$ are not necessarily the members of the set Δ. A consistent set of neighborhoods ought to have something to do with an element, and it is a (mathematical) temptation to take a *maximal* consistent set as representing an element. There is nothing wrong with that, except such elements are obviously the *total* elements of the domain defined by $\mathcal{D}$, since relative to $\mathcal{D}$ we have said as much as is consistently possible about each one. These are interesting elements, but they need not be constructively attainable, whereas partial elements are.

A straightforward way of determining an element is to give a sequence of neighborhoods that "converges" to the element. If the $X_n \in \mathcal{D}$ are chosen so that

$$X_0 \supseteq X_1 \supseteq X_2 \supseteq \cdots \supseteq X_n \supseteq X_{n+1} \supseteq \cdots ,$$

then a sense of convergence is fairly clear. But the question is: convergence to what? Perhaps that is not the right question. Suppose we have another sequence:

$$Y_0 \supseteq Y_1 \supseteq Y_2 \supseteq \cdots \supseteq Y_n \supseteq Y_{n+1} \supseteq \cdots ,$$

can we say that the two sequences are shrinking down on the "same element"? If we can, then we have an equivalence relation between neighborhoods and can identify elements simply as *being* such equivalence classes. In other words, elements are introduced as ideal elements corresponding to the notion of limit provided by the sequences of neighborhoods. Now since all we really have is the order structure on $\mathcal{D}$ (that is, the $\subseteq$-relation), we can not expect the definition to be very subtle. In fact, we shall say that the two sequences of neighborhoods are equivalent (converge to the same element(s), are equally convergent) if they go "equally far" or formally:

$$\forall\, n\ \exists\, m\ .\ X_n \supseteq Y_m \text{ and } \forall\, m\ \exists\, n\ .\ Y_m \supseteq X_n .$$

This is clearly an equivalence relation, but it is clumsy to work with equivalence classes if a canonical choice from each class can be made. The official definition not only does this but avoids having to restrict attention to countable systems $\mathcal{D}$, where simple *sequences* of neighborhoods are sufficient.

DEFINITION. The (ideal) *elements* of a neighborhood system $\mathcal{D}$ are those subfamilies $x \subseteq \mathcal{D}$ where:

(i) $\Delta \in x$;

(ii) $X, Y \in x$ always implies $X \cap Y \in x$;

(iii) whenever $X \in x$ and $X \subseteq Y \in \mathcal{D}$, then $Y \in x$.

The totality of all such elements is denoted by $|\mathcal{D}|$. □

Either as a subfamily of $\mathcal{D}$ or as a subfamily of the power set of Δ, it is easy to see that the property $x \in |\mathcal{D}|$ is defined by closure conditions. We can then establish the:

PROPOSITION. *For any neighborhood system $\mathcal{D}$ the totality $|\mathcal{D}|$ is a closure system -- the domain determined by $\mathcal{D}$.* □

There is no mystery to the definition of an element: such an x is just a $\mathcal{D}$-filter, and $|\mathcal{D}|$ could be called the *filter completion* of $\mathcal{D}$. Every sequence of neighborhoods, $X_n \supseteq X_{n+1}$, determines a *filter base*, and we could say that

$$x = \{Y \in \mathcal{D} \mid \exists\, n\ .\ X_n \subseteq Y\}$$

is the (principal) "limit" of the sequence. Clearly two sequences determine the same filter if and only if they are equivalent in the sense mentioned earlier. Such ideas have been around for a long time; what is to be added to them is the way we relate different domains in the natural category that they form.

Before discussing mappings in general, a remark on the simpler notion of *isomorphism* is appropriate. A neighborhood system $\mathcal{D}$ is a poset under inclusion; so is the corresponding domain $|\mathcal{D}|$. If $\mathcal{D}_0$ and $\mathcal{D}_1$ are two neighborhood systems, then there could be two kinds of isomorphism between them, one at the level of neighborhoods and one at the level of elements. We can prove:

PROPOSITION. *Two neighborhood systems are isomorphic (as posets) iff the corresponding domains are isomorphic. In general an isomorphism between domains makes finite elements correspond to*

finite elements. For a neighborhood system, the finite elements are dually isomorphic to the neighborhood system, because the finite elements are exactly the principal filters. □

We stated earlier that tokens $t \in \Delta$ might not necessarily determine elements. Examples are easy to find showing this. Nevertheless it is always possible to represent a system in a form where the tokens are exactly the elements:

PROPOSITION. *Every neighborhood system $\mathcal{D}$ is isomorphic to the neighborhood system*

$$[\mathcal{D}] = \{[X] \mid X \in \mathcal{D}\} ,$$

where for $X \in \mathcal{D}$ we have:

$$[X] = \{x \in |\mathcal{D}| \mid X \in x\} .$$

The underlying set for $[\mathcal{D}]$ is $|\mathcal{D}|$. □

On the other hand, we could also equally well make the finite elements the tokens. It is more direct, for neighborhood systems, to make the neighborhoods themselves the tokens:

PROPOSITION. *Every neighborhood system $\mathcal{D}$ is isomorphic to the neighborhood system*

$$\Downarrow \mathcal{D} = \{\Downarrow X \mid X \in \mathcal{D}\}$$

where for $X \in \mathcal{D}$ we have:

$$\Downarrow X = \{Y \in \mathcal{D} \mid Y \subseteq X\} .$$

The underlying set for $\Downarrow \mathcal{D}$ is $\mathcal{D}$ itself. □

This last -- very elementary -- theorem suggests that finite elements might be used as tokens more generally, and the basic representation result shows that indeed the abstractness of closure systems is only apparent:

REPRESENTATION THEOREM. *Every closure system (as a poset) is isomorphic to the system of elements of a neighborhood system. Hence, the domains determined by neighborhood systems give us up to isomorphism all domains.* □

The proof, which we do not give in detail, is quite direct. If $\mathcal{C}$ is a closure system, let Δ be the set of consistent *finite* sets for $\mathcal{C}$. ($\Delta = \mathcal{C}_{00}$ in the earlier notation.) For $F \in \Delta$ define with the aid of the closure operator

$$\Downarrow F = \{G \in \Delta \mid F \subseteq \bar{G}\} .$$

Then the system

$$C = \{\Downarrow F \mid F \in \Delta\}$$

is such that $\mathcal{C}$ is isomorphic to $|C|$. It follows, therefore, that neighborhood systems give us the full range of domain structures, in what will be found to be a convenient representation.

4. APPROXIMABLE MAPPINGS AND THE CATEGORY OF DOMAINS

The motivating example of functionals in Section 1 is quite sufficient to suggest the definitions for the general case. Of course $(\mathcal{N} \to \mathcal{N})$ is special in that it has only countably many finite elements, in which case simple infinite towers are all that is needed to define continuity. The general domain requires the use of arbitrary directed sets or elements for that kind of definition. But if we remember that in any domain the elements are completely determined by the finite elements they contain (which always form a directed set), then the definition can be expressed in terms of neighborhoods in a much more elementary way.

DEFINITION. Let $\mathcal{D}_0$ and $\mathcal{D}_1$ be two neighborhood systems. An *approximable mapping* $f : \mathcal{D}_0 \to \mathcal{D}_1$ is a binary relation between the sets of neighborhoods such that

(i) $\Delta_0 f \Delta_1$;

(ii) XfY and XfY' always imply $Xf(Y \cap Y')$; and

(iii) XfY, $X' \subseteq X$, and $Y \subseteq Y'$ always imply $X'fY'$. □

The idea of this definition is that $f : \mathcal{D}_0 \to \mathcal{D}_1$ should be regarded as an input-output relation. When we say "XfY" we mean that if we are willing to say at least X about the input, then f will tell us at least Y about the output -- perhaps even more. So condition (i) is the trivial requirement that *no* information about the input assures *no* information about the output. Condition (ii) is a consistency condition: if f is willing, say at different times, to give two output statements for the same input information, then it must be willing to allow the *conjunction* of those output statements. Finally, condition (iii) is a monotonicity condition: given an input-output pair of neighborhoods, if either the input is strengthened or the output is weakened, then f must still be willing to go along with the resulting pair. Intuitively, this should all make good sense; formally, the conditions of the definition are just those needed to prove the next result:

PROPOSITION. *Approximable mappings* $f : \mathcal{D}_0 \to \mathcal{D}_1$ *always determine continuous functions between the domains* $|\mathcal{D}_0|$ *and* $|\mathcal{D}_1|$ *by virtue of the formula:*

$$f(x) = \{Y \in \mathcal{D}_1 \mid \exists\, X \in x \,.\, XfY\} \,.$$

Moreover, every continuous function can be obtained this way. □

The verification of the statement is simple, for we put into (i)-(iii) the conditions required to show that the image under the relation f of a filter $x \in |\mathcal{D}_0|$ is indeed a filter in $|\mathcal{D}_1|$. In the other direction, if $f : |\mathcal{D}_0| \to |\mathcal{D}_1|$ is continuous, then the neighborhood relation that determines it is just:

$$Y \in f(\uparrow X) \,,$$

where we define

$$\uparrow X = \{X' \in \mathcal{D}_0 \mid X \subseteq X'\}$$

as the principal filter corresponding to X. We use the same symbol for an approximable mapping $f : \mathcal{D}_0 \to \mathcal{D}_1$ and a continuous $f : |\mathcal{D}_0| \to |\mathcal{D}_1|$, because by what we have just said we have an order-preserving isomorphism between the two kinds of objects. (Recall, the order relation between continuous mappings is the pointwise order.) This observation together with the obvious fact that conditions (i)-(iii) are closure conditions proves the fundamental:

THEOREM. *The set of continuous functions between two given domains is again (isomorphic to) a domain under the pointwise partial order.* □

The result will become even clearer if we give an explicit notation to the neighborhoods of the function space.

DEFINITION. For two neighborhood systems $\mathcal{D}_0$ and $\mathcal{D}_1$, let $(\mathcal{D}_0 \to \mathcal{D}_1)$ be the neighborhood system over the set

$$\{f \mid f : \mathcal{D}_0 \to \mathcal{D}_1\}$$

consisting of all finite, non-empty intersections of sets of the form

$$[X,Y] = \{f \mid XfY\} \,,$$

where $X \in \mathcal{D}_0$ and $Y \in \mathcal{D}_1$. □

Any family of non-empty sets generates a neighborhood system by closing it under non-empty intersections. What is interesting about the construction just defined is that the neighborhood system gives back just the right set of elements to be a representation of the function space.

THEOREM. *The elements of the neighborhood system* $(\mathcal{D}_0 \to \mathcal{D}_1)$ *form a domain isomorphic to the domain of approximable mappings from* $\mathcal{D}_0$ *into* $\mathcal{D}_1$. □

As with the $[\mathcal{D}]$-construction, we find that tokens equal elements (up to isomorphism). In proving this above result we find that we can identify the finite elements of the function space by a simple formula involving finite sequences of pairs of neighborhoods. Suppose $X_i \in \mathcal{D}_0$ and $Y_i \in \mathcal{D}_1$ are given for each $i < n$. The first question is that of consistency; that is, when do we have

$$\bigcap_{i<n} [X_i,Y_i] \neq \emptyset \;?$$

This means: when does there exist a function f, where for each $i < n$, f maps $[X_i]$ into $[Y_i]$? The answer is: for all subsets $I \subseteq \{i \mid i < n\}$, if the set $\{X_i \mid i \in I\}$ is consistent in $\mathcal{D}_0$, then the set $\{Y_i \mid i \in I\}$ must also be consistent in $\mathcal{D}_1$. That is a very reasonable input-output consistency condition on a sequence of pairs of neighborhoods. It is easily seen to be necessary; the sufficiency is proved by showing that the consistency condition is just what is required to show that there is a minimal element f_0 of the neighborhood $[X_0,Y_0] \cap \ldots \cap [X_{n-1},Y_{n-1}]$ given the formula:

$$Xf_0Y \;\text{ iff }\; \bigsqcup\{Y_i \mid X \subseteq X_i\} \subseteq Y \,,$$

for arbitrary $X \in \mathcal{D}_0$ and $Y \in \mathcal{D}_1$.

We do not have time to elaborate on these observations, but they prove that if domains can be given *countable* neighborhood systems, then so can the function space. Not only this, but the whole definition is very constructive: from knowing about the poset structure of $\mathcal{D}_0$ and $\mathcal{D}_1$, we know fully about the poset structure of $(\mathcal{D}_0 \to \mathcal{D}_1)$.

It may not have been necessary to introduce two names for the same notion: approximable/continuous mappings. But the definitions look quite different, and the second is a slightly

mysterious name -- until one realizes that domains can be made into topological spaces in such a way that continuous maps are exactly the topologically continuous maps. (Hint: use the sets $[X]$ for $X \in \mathcal{D}$ as a basis for a topology on $|\mathcal{D}|$.) The reason for "approximable" is that such maps can be given finite approximations, and are indeed determined by these approximations. In any neighborhood system $\mathcal{D}$, for $X \in \mathcal{D}$ and $x \in |\mathcal{D}|$ the following are equivalent:

$$X \in x \ , \ x \in [X] \ , \ \dagger X \subseteq x \ ,$$

and we can say the finite element $\dagger X$ *approximates* x. (This is the same as saying that the neighborhood X gives a *finite amount of information* about x.) Moreover, we can write:

$$x = \bigcup\{\dagger X \mid X \in x\} \ ,$$

so that every element is the *union* (or *limit*) of its finite approximations; and, hence, it is determined by them. What we have shown is that the same goes for approximable mappings. The statements

$$f \in \bigcap_{i<n} [X_i, Y_i]$$

and

$$X_i f Y_i \ , \ \text{all } i < n$$

are equivalent; and we saw how to define the principal finite element of the indicated neighborhood. The mappings we are talking about are those that can be obtained through finite approximation. (Keep in mind that the precise meaning of the word "finite" is relative to the neighborhood systems under consideration.)

It should be apparent by now that the domains together with the approximable mappings form a *category*, in the technical sense of the word. Just to underline the fact, we say how to form identity and composition maps.

PROPOSITION. *For any neighborhood system $\mathcal{D}$, the identity map $I_{\mathcal{D}} : \mathcal{D} \to \mathcal{D}$ is approximable, and we have the equivalent definitions:*

(i) $I_{\mathcal{D}}(x) = x$, *for all* $x \in |\mathcal{D}|$; *and*

(ii) $X I_{\mathcal{D}} Y$ *iff* $X \subseteq Y$, *for all* $X, Y \in \mathcal{D}$. □

PROPOSITION. *For given neighborhood systems $\mathcal{D}_0$, $\mathcal{D}_1$, and $\mathcal{D}_2$ and approximable maps*

$$f : \mathcal{D}_0 \to \mathcal{D}_1 \quad \text{and} \quad g : \mathcal{D}_1 \to \mathcal{D}_2$$

the composition $g \circ f : \mathcal{D}_0 \to \mathcal{D}_2$ is approximable, and we have the equivalent definitions:

(i) $(g \circ f)(x) = g(f(x))$, *for all* $x \in |\mathcal{D}_0|$; *and*

(ii) $X(g \circ f)Z$ *iff* $\exists\, Y \,.\, XfY$ *and* YgZ, *for all* $X \in \mathcal{D}_0$ *and* $Z \in \mathcal{D}_2$. □

The category of domains has many good properties. We have seen how to construct function spaces; the construction of products is even easier.

PROPOSITION. *For neighborhood systems $\mathcal{D}_0$ and $\mathcal{D}_1$ over Δ_0 and Δ_1 assume*

$$\Delta_0 \cap \Delta_1 = \emptyset \ .$$

Define

$$\mathcal{D}_0 \times \mathcal{D}_1 = \{X \cup Y \mid X \in \mathcal{D}_0, \ Y \in \mathcal{D}_1\} \ .$$

Then $(\mathcal{D}_0 \times \mathcal{D}_1)$ is a neighborhood system and the domain $|\mathcal{D}_0 \times \mathcal{D}_1|$ is order isomorphic to the usual cartesian product of $|\mathcal{D}_0|$ and $|\mathcal{D}_1|$. The pairing function on elements $x \in |\mathcal{D}_0|$ and $y \in |\mathcal{D}_1|$ can be defined by

$$(x, y) = \{X \cup Y \mid X \in x, \ Y \in y\} \ .$$

This is a filter in $|\mathcal{D}_0 \times \mathcal{D}_1|$ and the pointwise pairing of approximable maps is again approximable. The obvious projection maps are also approximable. □

We cannot stop to verify or even mention all the details, but the category of domains not only has categorical products, but the function-space construction is categorically well behaved. There is a technical name for the well-behavedness: the category of domains and approximable maps is *cartesian closed*. This means in particular that the *evaluation map*,

$$\text{eval} : (\mathcal{D}_1 \to \mathcal{D}_2) \times \mathcal{D}_1 \to \mathcal{D}_2 \ ,$$

is approximable (that is, it exists in the category). Furthermore, the systems

$$((\mathcal{D}_0 \times \mathcal{D}_1) \to \mathcal{D}_2) \quad \text{and} \quad (\mathcal{D}_0 \to (\mathcal{D}_1 \to \mathcal{D}_2))$$

are "naturally" isomorphic. This makes it possible to write down an endless number of logically defined maps in the category.

The category is much more than just a cartesian closed category, however. For instance, the *fixed-point operator*,

$$\mathrm{fix} : (\mathcal{D} \to \mathcal{D}) \to \mathcal{D} ,$$

is approximable, where

$$\mathrm{fix}(f) = f(\mathrm{fix}(f))$$

holds for all $f \in |\mathcal{D} \to \mathcal{D}|$. And there are a multitude of other equations satisfied by the fixed-point operator. In another direction, the category supports many other interesting functors besides $\times$ and $\to$. Reference [10] can be consulted for details.

5. DOMAIN EQUATIONS AND THE UNIVERSAL DOMAIN

In Section 1 we began with some simple examples which led us directly to the general definition of closure/neighborhood systems. In Section 4 we saw that the category of domains had some good closure properties, which of course let us multiply our examples. But the product and function-space constructs are pretty bland; what is needed is a way of obtaining more "special-purpose" examples where the domains themselves have interesting or useful structure. One way of constructing good domains is by a recursion which iterates given functors.

A convenient trick for defining quite complex neighborhood systems is to construe the neighborhoods as sets of finite sequences. A two-letter alphabet is sufficient, and we take $\Sigma = \{0,1\}$, the binary alphabet. We then let Σ^* be the free semi-group of all binary words, where we regard Σ itself as equal to the set of words of length 1 in the usual way. The empty word is Λ, and for $\sigma,\tau \in \Sigma^*$ we denote concatenation by $\sigma\tau$. If $X \subseteq \Sigma^*$, then

$$\sigma X = \{\sigma\tau \mid \tau \in X\} .$$

Note that if $X,Y \subseteq \Sigma^*$, then the sets $0X$ and $1Y$ are always disjoint. For forming neighborhood systems in this section we shall always take $\Delta = \Sigma^*$ and assume that the systems do not contain the empty set.

Let us continue our list of examples of special domains.

EXAMPLE 5. Let $\mathcal{B}$ be the unique family of non-empty subsets of Σ^* defined by:

$$\mathcal{B} = \{\Sigma^*\} \cup \{0X \mid X \in \mathcal{B}\} \cup \{1X \mid X \in \mathcal{B}\} .$$

The least element in $|\mathcal{B}|$ is

$$\perp = \{\Sigma^*\} ,$$

and for $x \in |\mathcal{B}|$ and $\sigma \in \Sigma^*$ we define:

$$\sigma x = \{Y \in \mathcal{B} \mid \exists\, X \in x \,.\, \sigma X \subseteq Y\} . \quad \square$$

The family $\mathcal{B}$ is simply the least family of sets generated from Σ^* by the two set-forming operations shown. That $\mathcal{B}$ is a neighborhood system is easily proved by induction using a law like:

$$\sigma X \cap \sigma Y = \sigma(X \cap Y) ;$$

that the operation $x \mapsto \sigma x$ is approximable is also obvious since $\sigma X \subseteq Y$ is the appropriate neighborhood relation. Well, these are details; the real question is: what is the meaning of $\mathcal{B}$? The explanation is facilitated by a general definition.

DEFINITION. For any two neighborhood systems (over Σ^*) $\mathcal{D}_0$ and $\mathcal{D}_1$, let

$$\mathcal{D}_0 + \mathcal{D}_1 = \{\Sigma^*\} \cup \{0X \mid X \in \mathcal{D}_0\} \cup \{1Y \mid Y \in \mathcal{D}_1\} .$$

This construct is called the *sum* of the domains. $\square$

The definition of the product could be made to look similar for systems over Σ^*:

$$\mathcal{D}_0 \times \mathcal{D}_1 = \{\{\Lambda\} \cup 0X \cup 1Y \mid X \in \mathcal{D}_0,\ Y \in \mathcal{D}_1\} .$$

The reason for adding in the empty string Λ is that

$$\Sigma^* = \{\Lambda\} \cup 0\Sigma^* \cup 1\Sigma^* .$$

If we did not put it into the product definition, then the result $\mathcal{D}_0 \times \mathcal{D}_1$ would only be a system over $\Sigma^* - \{\Lambda\}$.

In the category of domains, both the product and the sum can be made into functors in an obvious way. The product *is* the categorical (cartesian) product. The sum, however, *is not* the categorical coproduct. Such does not exist in the category: though there is a terminal object (say, the trivial system $\{\Sigma^*\}$), there is no initial object, and no way to make disjoint sums. The uniform occurrence of $\perp$ in all domains is one of the blocks to forming disjoint sums. The elements of the above defined $\mathcal{D}_0 + \mathcal{D}_1$ could be regarded as a disjoint sum of the two posets $|\mathcal{D}_0|$ and $|\mathcal{D}_1|$ *plus* a new $\perp$-element. Though it is not the categorical coproduct, it still has many good properties and is a useful functor.

We can now detail in greater depth the idea behind the system $\mathcal{B}$. The *finite* elements of $\mathcal{B}$ are all of the form $\sigma\perp$ where σ is a finite word (or sequence) of 0's and 1's. We find that the relationship

$$\sigma\perp \subseteq \tau\perp$$

holds if and only if σ is an initial segment of τ. Of course $\perp$ is included in everything. What do these elements approximate? Answer: *infinite* sequences over the binary alphabet; these are in fact the *total* elements of $|\mathcal{B}|$. If ρ is an infinite sequence, and ρ_n is the initial segment of ρ of length n, then we can unambiguously identify ρ with this element of $|\mathcal{B}|$:

$$\bigcup_{n=0}^{\infty} \rho_n \perp .$$

This is nothing more than the filter

$$\{\rho_n \Sigma^* \mid n \in \mathbb{N}\} ,$$

and it is easily proved to be maximal. So the domain determined by $\mathcal{B}$ is the domain of binary sequences, where the finite sequences are only *partial* and are extendable to the infinite *total* sequences. Moreover, $\mathcal{B}$ satisfies a *domain equation* of the form

$$\mathcal{B} \cong \mathcal{B} + \mathcal{B} .$$

Not only this, but $\mathcal{B}$ is uniquely determined up to isomorphism as the *initial solution* of the equation. (This notion can be made precise categorically by introducing the *algebras* corresponding to a functor from domains to domains; in this case the functor is

$$X \mapsto X + X .$$

An initial algebra is a "free" algebra on zero generators.) This initial algebra characterization of $\mathcal{B}$ is convenient to work with, and it gives a "recursive definition" for the domain.

EXAMPLE 6. Let $\mathcal{C}$ be the neighborhood system

$$\mathcal{C} = \mathcal{B} \cup \{\{\sigma\} \mid \sigma \in \Sigma^*\} .$$

The domain equation satisfied by $\mathcal{C}$ is:

$$\mathcal{C} \cong \{\Sigma^*\} + \mathcal{C} + \mathcal{C} .$$

We can identify the words $\sigma \in \Sigma^*$ with the principal filters $\uparrow\{\sigma\}$, which are also maximal in $|\mathcal{C}|$. □

The difference between $\mathcal{B}$ and $\mathcal{C}$ is that $\mathcal{C}$ allows for two kinds of finite elements: the partial ones of the form $\sigma\perp$, and the total ones of the form σ. Both $\mathcal{B}$ and $\mathcal{C}$ allow the operations $x \mapsto \sigma x$, and both allow for a more general associative concatenation operation $x,y \mapsto xy$. In $\mathcal{B}$, the operation is trivial and satisfies the equation

$$xy = x .$$

In $\mathcal{C}$ the situation is less extreme, but still the equation is satisfied for *infinite* x. It is an interesting question to find less one-sided domains that define a semigroup extending Σ^*, as an approximable multiplication, and allow infinite elements. There are several possibilities, but the author does not understand them very well yet.

EXAMPLE 7. Let E be recursively defined by:

$$E = \{\Sigma^*\} \cup \{\{00\},\{01\}\} \cup \{10X \cup 11Y \mid X,Y \in E\} .$$

The domain equation satisfied by E is:

$$E \cong \{\Sigma^*\} + \{\Sigma^*\} + (E \times E) .$$

There are two (total) distinguished elements of E:

$$0 = \{X \in E \mid 00 \in X\} , \text{ and}$$
$$1 = \{X \in E \mid 01 \in X\} .$$

The approximable operation $x,y \mapsto \langle x,y\rangle$ is defined by:

$$\langle x,y\rangle = \{Z \in E \mid \exists\, X \in x\ \exists\, Y \in y\ .\ 10X \cup 11Y \subseteq Z\} .$$

There are two projection operations $z \mapsto pz$ and $z \mapsto qz$ defined by:

$$pz = \{X \in E \mid \exists\, Z \in z\ .\ 10\Sigma^* \cap Z \subseteq 10X\} , \text{ and}$$
$$qz = \{Y \in E \mid \exists\, Z \in z\ .\ 11\Sigma^* \cap Z \subseteq 11Y\} .$$

We have satisfied in E as equations:

$$p\langle x,y\rangle = x \quad \text{and} \quad q\langle x,y\rangle = y ,$$

but *not* the opposite:

$$\langle pz,qz\rangle = z ,$$

since E is more than just its own cartesian square. □

What is the "meaning" of E? The finite elements are exactly those generated by the binary operation $x,y \mapsto \langle x,y\rangle$ from $\perp$, 0,

and 1. Those not containing $\perp$ are *total*; while any involving $\perp$ are *partial*. This can be seen from the inclusions:

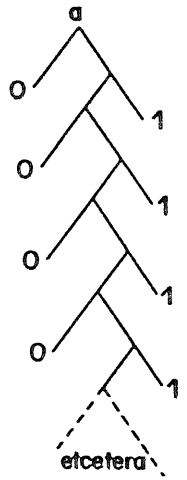
$$\perp \subseteq x \ , \text{ and}$$

$$\langle x,y\rangle \subseteq \langle x',y'\rangle \quad \text{if} \quad x \subseteq x' \text{ and } y \subseteq y' \ .$$

We could diagram such elements as *trees* with binary branching and with the ends of the branches marked with $\perp$ or 0 or 1. Besides these finite trees there are also many, many infinite trees. For example, the equation

$$a = \langle 0,\langle a,1\rangle\rangle$$

has a unique solution in E, and the diagram of the infinite, total tree is clear:



We could also think of E as a domain of fully parsed *expressions* with two *atomic expressions* and one *binary mode of composition*. By leaving room for partial expressions as well as total expressions, we allow for a coherent theory of *infinite expressions*.

There is an unlimited supply of similar domains all characterized by analogous domain equations. Solutions to the domain equations can be proved to exist in many ways. What may be novel in the present exposition is that the neighborhoods

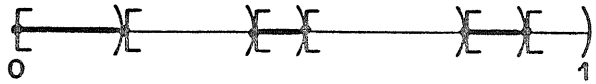
(finite elements) are defined first by a recursion with a structure parallel to the desired domain equation, then the whole domain is found as the completion of the neighborhood system. This approach is particularly simple when the domain equation involves only the functions + and ×, because the required neighborhoods are so easily sorted out. For more complex equations, the method of retracts is more general and easier to apply directly, but the analysis of what domain is obtained may be far from obvious. The scope of the method of retracts is most broad when applied to the *universal domain*. Such a domain is not actually unique, but a particularly nice one can be defined recursively in a way similar to the previous examples.

EXAMPLE 8. Let U be recursively defined by

$$\begin{aligned} U = {} & \{\Sigma^*\} \cup \{\{\Lambda\} \cup 0X \mid X \in U\} \\ & \cup \{\{\Lambda\} \cup 1Y \mid Y \in U\} \\ & \cup \{\{\Lambda\} \cup 0X \cup 1Y \mid X,Y \in U\} \ . \end{aligned}$$

It can be proved by induction that $\emptyset \notin U$, and indeed that all sets in U are infinite. It must also be proved by induction that U is closed under (consistent) intersections; thus, U is a neighborhood system. The same kind of proof shows that it is also closed under union of two sets. □

The system U is similar to those of other examples, but it is not the same as any of the ones we have yet seen. How is it different? What is its special nature? Perhaps another representation will help. Consider the *dyadic* rational numbers in the interval $[0,1)$. Take this half-open set as Δ. By a neighborhood we will mean any non-empty subset of Δ that is a finite union of half-open intervals. A picture of one would look like:



Here the dark intervals are the ones making up the desired union, and we might as well take them as separated as shown. These sets clearly form a neighborhood system U'. But is it isomorphic to U? To see that it is, consider the two shrinking transformations

$$t \mapsto t/2 \quad \text{and} \quad t \mapsto (t+1)/2 \ ,$$

which we can call

$$t \mapsto 0t \quad \text{and} \quad t \mapsto 1t$$

for short. For $X,Y \subseteq \Delta$, there are three images we can form by means of these transformations:

$$0X \ , \ 1Y \ , \text{ and } 0X \cup 1Y \ .$$

The lemma to prove is: any non-empty union of dyadic intervals contained in Δ can be obtained from Δ by applying these three operations over and over. (The proof is by induction on the largest power of 2 needed in a denominator of a number at an endpoint of one of the intervals in the union.) With this way of thinking of the generation of U' it is obvious that it is isomorphic to U.

The geometric representation of U', however, has advantages in visualization: If we adjoin $\emptyset$ to U', it becomes a *Boolean algebra*. (Note, as is clear from the picture, that the complement of a union of dyadic intervals is again such -- unless it is empty.) By taking half-open intervals, points are *excluded*: U' as a Boolean algebra is *atomless*. But U' is countable, and -- up to isomorphism -- there is only one countable atomless Boolean algebra, namely, the free one on $\aleph_0$-generators. This Boolean algebra is quite remarkable, for *every* countable Boolean algebra is isomorphic to a *subalgebra*! What will this mean for neighborhood systems?

Let $\mathcal{D}$ be any countable neighborhood system. Pass to the other representation of $\mathcal{D}$, say $[\mathcal{D}]$. This system $[\mathcal{D}]$ has the advantage that consistency means having a non-empty intersection. Generate a Boolean algebra of subsets of the set $|\mathcal{D}|$ from the family $[\mathcal{D}]$. This will of course be a countable Boolean algebra, because $\mathcal{D}$ was countable. So this algebra is isomorphic to a subalgebra of $U' \cup \{\emptyset\}$. But then by excluding the empty set from consideration we see that $[\mathcal{D}]$ is isomorphic to a subsystem of U'. All that is needed is a clear definition of *subsystem* of a neighborhood system.

DEFINITION. For two neighborhood systems $\mathcal{D}_0$ and $\mathcal{D}_1$ we write

$$\mathcal{D}_0 \lhd \mathcal{D}_1$$

to mean that $\mathcal{D}_0$ and $\mathcal{D}_1$ are systems over the same set Δ and that $\mathcal{D}_0 \subseteq \mathcal{D}_1$. Moreover, we require that whenever $X, Y \in \mathcal{D}_0$ and $X \cap Y \in \mathcal{D}_1$, then $X \cap Y \in \mathcal{D}_0$. In other words, consistency in $\mathcal{D}_0$ is just the *restriction* of the notion to $\mathcal{D}_1$. In case $\mathcal{D}_0$ is isomorphic to a system $\mathcal{D}_0' \lhd \mathcal{D}_1$, we write this as

$$\mathcal{D}_0 \underset{\sim}{\lhd} \mathcal{D}_1$$

for short. □

THE EMBEDDING THEOREM. *We have $\mathcal{D} \underset{\sim}{\lhd} U$ for every countable neighborhood system $\mathcal{D}$.* □

This is the sense in which U is "universal", but there are many such systems. For example $U + U$ is universal but *not* isomorphic to U. But it is interesting enough to have one universal system.

Perhaps the reader will have noticed that the relation $\lhd$ is defined by a *closure condition*? Since we have a sufficiently large, but countable U already fixed, we conclude:

THE UNIVERSE THEOREM. *The system*

$$\{\mathcal{D} \mid \mathcal{D} \lhd U\}$$

is a closure system with a countable number of finite elements: the finite subsystems of U. □

The finite elements are easy to pick out, because if $X \subseteq U$ is a finite set, all we have to do is to close it under finite intersections (non-empty!) to obtain $\bar{X} \lhd U$. But $\bar{X}$ is a finite family of sets itself. Since U is countable, there are only countably many finitely generated subsystems. This gives us -- as a domain -- a "universe" of all countable domains (up to isomorphism). We are used to thinking of a set of sets, but not of a ring of rings or lattice of lattices. A domain of domains is quite possible, and -- provided we can find suitable approximable operations on the "universe" -- it can be quite useful for proving the existence of a remarkable number of domains. The way to achieve this is by the *calculus of retractions*. The existence of the necessary retraction mappings is guaranteed by the next result.

PROPOSITION. (i) *If $\mathcal{D}_0 \lhd \mathcal{D}_1$, then there exist approximable maps*

$$i : \mathcal{D}_0 \to \mathcal{D}_1 \quad \text{and} \quad j : \mathcal{D}_1 \to \mathcal{D}_0$$

forming a projection pair in the sense that

$$j \circ i = I_{\mathcal{D}_0} \quad \text{and} \quad i \circ j \subseteq I_{\mathcal{D}_1} \ .$$

Indeed, for $x \in |\mathcal{D}_0|$ and $y \in |\mathcal{D}_1|$, we have

$$i(x) = \{Y \in \mathcal{D}_1 \mid \exists\, X \in x \ .\ X \subseteq Y\} \text{ and}$$

$$j(y) = y \cap \mathcal{D}_0 \ .$$

(ii) *Conversely, if there is a projection pair between* $\mathcal{D}_0$ *and* $\mathcal{D}_1$, *then* $\mathcal{D}_0 \trianglelefteq \mathcal{D}_1$, *because* $\mathcal{D}_0$ *is isomorphic to*

$$\{Y \in \mathcal{D}_1 \mid Y(i \cdot j)Y\} \trianglelefteq \mathcal{D}_1 .$$

Consequently, projection pairs determine exactly the relation $\trianglelefteq$ *between domains.* □

Part (ii) above suggests that subsystems are also determined by mappings, and in fact we have:

THEOREM. *The following conditions are equivalent for an approximable map* $a : \mathcal{D} \to \mathcal{D}$:

(i) *There is a projection pair with* $a = i \cdot j$;

(ii) $a \cdot a = a \subseteq I_{\mathcal{D}}$ *and the range of* a *is isomorphic to a domain;*

(iii) $a \cdot a = a \subseteq I_{\mathcal{D}}$, *and whenever* XaZ, *then* $X \subseteq YaY \subseteq Z$, *for some* $Y \in \mathcal{D}$.

Consequently, the maps satisfying these conditions form a domain, because there is a mapping

$$\mathrm{sub} : (\mathcal{D} \to \mathcal{D}) \to (\mathcal{D} \to \mathcal{D})$$

defined by

$$X \ \mathrm{sub}(a) Z \quad \textit{iff} \quad X \subseteq YaY \subseteq Z, \ \textit{some} \ Y \in \mathcal{D}$$

which itself satisfies these conditions on $(\mathcal{D} \to \mathcal{D})$ *and has as its range exactly these maps on* $\mathcal{D}$. □

A mapping $a : \mathcal{D} \to \mathcal{D}$ where $a \cdot a = a$ is called a *retraction*. Its range of values is the same as its fixed-point set,

$$\{x \in |\mathcal{D}| \mid x = a(x)\} .$$

As a poset, such a fixed-point set shares many of the properties of domains, but it need not be a domain in the sense we have been using the word. If $I_{\mathcal{D}} \subseteq a$, then it is a closure system (and, hence, a domain). But the case we are interested in is $a \subseteq I_{\mathcal{D}}$, and it need not be a closure system. Even when a satisfies the conditions of the theorem, the fixed-point set need not be a closure system -- but it is isomorphic to one. The trick is that for *any* approximable a, we have

$$\mathcal{D}_a = \{Y \in \mathcal{D} \mid YaY\} \trianglelefteq \mathcal{D} .$$

But when a satisfies the conditions, this set determines a, and the poset of fixed points of a is isomorphic to $|\mathcal{D}_a|$ by restriction, as in the definition of j above.

More than all this, however, is the additional information that, owing to the uniform way the construction was done, we find

$$\{X \mid X \trianglelefteq \mathcal{D}\} \trianglelefteq (\mathcal{D} \to \mathcal{D}) ,$$

because sub is the required retraction. Applying all this to U, we see

$$\{\mathcal{D} \mid \mathcal{D} \trianglelefteq U\} \trianglelefteq (U \to U) \trianglelefteq U ,$$

since $(U \to U)$ is a countable system. This may not seem very striking until the reader realizes that the embeddability of all these spaces in U gives us a *notation* for functions and spaces all as *elements* of the *same* space U.

6. WHAT DOES THIS ALL HAVE TO DO WITH COMPUTER SCIENCE?

The question will have crossed the reader's mind long before this. For the answer, we have to go back to the examples of Section 1 that motivated the study in the first place. Everyone knows about partial functions $f : \mathbb{N} \rightharpoonup \mathbb{N}$. We could even use other sets A and B and consider partial functions $g : A \rightharpoonup B$. There are sets that can be constructively given (in the most down-to-earth manner!) and partial functions that can be effectively (recursively) defined (and computed efficiently!). Many people are content to leave the theory of computability with that, for indeed there is much to say about algorithms for functions defined on explicitly given *discrete* sets. The theory advocated here, however, without denying the importance of the discrete case, suggests that there is another level to which the analysis of computability can be applied. It is also claimed that computer science needs this extended theory.

The first step brought us to the domain (closure system) of *functions* $(\mathbb{N} \rightharpoonup \mathbb{N})$, and we could just as well have defined a function space $(A \rightharpoonup B)$ of partial functions between any two sets. These are posets, and we recognized that there are *functionals* (operators) defined on these function spaces which have simple-to-state order-theoretic properties. Moreover, these properties (continuity, monotonicity) hold of functionals that are *computably defined.*

Examples of programs that operate on functions as well as numbers are immediate: for example, take a function f and iterate it n-times on 0, say, computing $f^n(0)$. We just suggested a functional of type

$$(\mathcal{N} \rightharpoonup \mathcal{N}) \to (\mathcal{N} \rightharpoonup \mathcal{N}) \ .$$

It is obviously a basic idea, and examples of this sort can be multiplied over and over. In other words, programs embody not only specific calculations of special functions but also *methods* for obtaining in a general way new functions from given functions. It should be self-evident that computer science needs the study of such methods, and what the author is trying to collect is the mathematical tools and ideas to pursue this study. This is where the theory of domains comes in.

Going back to the start again, we can regard $\mathcal{N}$ as a domain -- in two ways, in fact

$$\check{\mathcal{N}} = \{\{n\} \mid n \in \mathcal{N}\} \cup \{\emptyset\} \qquad \text{(closure system)}$$

$$\hat{\mathcal{N}} = \mathcal{P}(\mathcal{N}) \cup \{\{n\} \mid n \in \mathcal{N}\} \qquad \text{(neighborhood system)}$$

All we have done is to add $\perp$ as the "undefined" element to $\mathcal{N}$. As we have gradually shifted to neighborhood systems, we take $\hat{\mathcal{N}}$ (and the corresponding $\hat{A}$ for any set A) as the one to use; of course it is trivial to see that $|\hat{\mathcal{N}}|$ and $\check{\mathcal{N}}$ are isomorphic posets.

Now $\mathcal{P}(\mathcal{N} \rightharpoonup \mathcal{N})$ is not the same poset as $|\mathcal{P}(\hat{\mathcal{N}} \to \hat{\mathcal{N}})|$, but it is almost the same. There is an easy-to-define system

$$\mathcal{D} \lhd \mathcal{P}(\hat{\mathcal{N}} \to \hat{\mathcal{N}})$$

where $|\mathcal{D}|$ is isomorphic to $\mathcal{P}(\mathcal{N} \rightharpoonup \mathcal{N})$. We find for a number of reasons (e.g. category-theoretic reasons!) that it is simpler and more regular to take $(\mathcal{D}_0 \to \mathcal{D}_1)$ as the principal function-space construction.

So, starting from $\hat{\mathcal{N}}$, or $\hat{A}$ and the like, we iterate such constructs (functors) as

$$(\mathcal{D}_0 \times \mathcal{D}_1) \quad \text{and} \quad (\mathcal{D}_0 \to \mathcal{D}_1) \ ,$$

obtaining the "higher-type" domains. But more than these there are domains like U that, in a certain sense, are of the highest type -- at least among countable neighborhood systems. There is clearly no reason to exclude U, as can be appreciated from its very simple recursive definition. And, to extract all the interesting subdomains of U is the result of an elementary construction

$$\mathcal{D}_a = \{Y \in U \mid YaY\}$$

for any approximable $a : U \to U$. We thus almost have at hand a notation for definitions. What is lacking is the formal notation for functions.

Before passing to the notational questions, we have to emphasize the justification of the theory of domains from the point of view of the *theory of computability*: Not only do these higher type domains exist, but they can be *effectively given*. ("Effective" is used here in the sense of recursive function theory.) Surely the reader has seen already how elementary our examples are; that is to say, the constructions of the desired neighborhood system -- and of the desired approximable mappings -- requires very little. (Indeed with the aid of Σ^* a large number of examples of neighborhood systems can be found with a minimum of set notation.) We can give a name to this feeling by suggesting a general procedure of "Gödel numbering" of neighborhoods. And there would be other ways of making the degree of constructivity concrete.

DEFINITION. A *computable presentation* of a neighborhood system $\mathcal{D}$ is a way of indexing the sets in $\mathcal{D}$ so we can write:

$$\mathcal{D} = \{X_n \mid n \in \mathcal{N}\} \ ,$$

and where we know the following two relations are recursive:

(i) $X_n \cap X_m = X_k$;

(ii) $\exists\, k \in \mathcal{N} \,.\, X_n \cap X_m = X_k$.

A system with a computable presentation is *effectively given*. □

We must take care with this definition, as Kanda and Park [5] have pointed out, for presentations are not isomorphism invariants. To *give* a domain effectively, the indexing must be given together with (the Gödel numbers of) the recursive functions that decide the two relations (i) and (ii). What is being given here is just the poset structure of $\mathcal{D}$, but note we demand in (ii) that the notion of consistency of neighborhoods is effectively decidable. In view of (i), whenever X_n and X_m are consistent, then we can find the k that indexes the intersection.

All the examples we have mentioned are effectively given. It is very easy to check that if $\mathcal{D}_0$ and $\mathcal{D}_1$ are effectively given, then so are

$$\mathcal{D}_0 \times \mathcal{D}_1 \quad \text{and} \quad \mathcal{D}_0 + \mathcal{D}_1 \ .$$

Indeed, from the explicit enumerations for the $\mathcal{D}_i$ we can construct what is needed for the compound domain. The case of

$$\mathcal{D}_0 \to \mathcal{D}_1$$

may not be so obvious, but the result is true here also. The proof rests on the facts that (1) we can test a consistency question in $(\mathcal{D}_0 \to \mathcal{D}_1)$ of the form

$$\bigcap_{i<n} [X_i, Y_i] \neq \emptyset$$

as a sequence of consistency checks in $\mathcal{D}_0$ and $\mathcal{D}_1$ (as was indicated in Section 4), and that (2) we can also test inclusions between function-space neighborhoods by means of the basic equivalence:

$$\bigcap_{i<n} [X_i, Y_i] \subseteq [X, Y] \quad \text{iff} \quad \bigcap \{Y_i \mid X \subseteq X_i\} \subseteq Y \ ,$$

which holds provided the first-mentioned neighborhood is non-empty. Again tests involving neighborhoods of $(\mathcal{D}_0 \to \mathcal{D}_1)$ are thrown back on (several) tests of relationships among neighborhoods of $\mathcal{D}_0$ and of $\mathcal{D}_1$. We can see in this way $(\mathcal{D}_0 \to \mathcal{D}_1)$, too, is effectively given. Therefore we have a very rich collection of such domains. What requires further comment is the way in which different effectively given domains are to be related.

Before speaking of relationships, there is a prior question about *elements*. A quite simple domain like $(\hat{\mathcal{N}} \to \hat{\mathcal{N}})$ or U, though effectively given, has an *uncountable* number of elements. (For countable neighborhood systems $\mathcal{D}$, the cardinality of $|\mathcal{D}|$ never goes beyond that of the continuum.) Obviously not all of these elements can be regarded as having a constructive existence. We should try to pick out those that do.

DEFINITION. Let $\mathcal{D}$ be effectively given with

$$\mathcal{D} = \{X_n \mid n \in \mathcal{N}\} \ .$$

An element $x \in |\mathcal{D}|$ is said to be *computable* if the set

$$\{n \in \mathcal{N} \mid X_n \in x\}$$

is recursively enumerable. □

Some justification of this definition is required, but the primary example of $(\mathcal{N} \to \mathcal{N})$ is sufficient for seeing why it is correct. Let $(\mathcal{N} \to \mathcal{N})_0$ be the set of *finite* partial functions. Then as a neighborhood system we use as neighborhoods of $(\mathcal{N} \to \mathcal{N})$ the sets

$$\{f \in (\mathcal{N} \to \mathcal{N}) \mid e \subseteq f\}$$

where $e \in (\mathcal{N} \to \mathcal{N})_0$. Then the elements of this system correspond exactly to the elements of the set $(\mathcal{N} \to \mathcal{N})$. When is $f \in (\mathcal{N} \to \mathcal{N})$ computable? By the definition above, we first look to a standard enumeration

$$(\mathcal{N} \to \mathcal{N})_0 = \{e_n \mid n \in \mathcal{N}\}$$

to show that the domain is effectively given. Then, the meaning of the definition in this case is that the set

$$\{n \in \mathcal{N} \mid e_n \subseteq f\}$$

should be recursively enumerable. But this is just a slightly more complicated way of saying that the graph of f, as a subset of $(\mathcal{N} \times \mathcal{N})$, is recursively enumerable. But, as is well known, this in turn is just a way of saying that f is *partial recursive*. This is the correct notion of computable element of $(\mathcal{N} \to \mathcal{N})$. Using only the *total recursive* functions in $(\mathcal{N} \to \mathcal{N})$ would be wrong, because from a definition of an enumeration there is no way of telling whether the function is total or partial. The partial recursive functions can be effectively enumerated but not the total recursive functions. The "partial" notion is the fundamental one, as experience in recursive function theory has shown.

Another way of thinking about the reason for formulating the definition in the way we have is this: the elements of a domain are in general *infinite*. Neighborhoods represent finite approximations; recall that for $X \in \mathcal{D}$ and $x \in |\mathcal{D}|$ we have:

$$X \in x \quad \text{iff} \quad \uparrow X \subseteq x \ ,$$

and "in the limit"

$$x = \bigcup \{\uparrow X \mid X \in x\} \ .$$

So when we enumerate $\{n \in \mathcal{N} \mid X_n \in x\}$ we are enumerating the finite approximations to x. We are expecting for x to be computable that these approximations can *eventually* be listed in an effective way. We *do not* demand that we can uniformly decide whether

$$X_n \in x \ .$$

The point is that x is infinite and it has to be built up "slowly", step by step. At one moment we may know that

$$X_{m_0}, X_{m_1}, \ldots, X_{m_{k-1}} \in x \ ,$$

but confronted with an arbitrary X_n we cannot say -- at that moment -- that it will not turn up later. Of course if it is

$$\mathsf{rep} : U \to (U \to U)$$

(standing for "abstract" and "represent", respectively), which form a projection pair to prove

$$(U \to U) \mathrel{\underset{\sim}{\triangleleft}} U \ .$$

Via rep the computable elements of U therefore parameterize the computable elements of $(U \to U)$ in a many-one manner. So all the fuss is being reduced to the study of elements of U alone.

The success of the reduction can be made much more visible if we turn U into an "algebra" -- and it is a very rich one. Keep in mind that

$$(U \times U) \mathrel{\underset{\sim}{\triangleleft}} U \ ,$$

so there are (computable!) maps

$$\mathsf{pair} : U \times U \to U \ , \text{ and}$$

$$\mathsf{fst}, \mathsf{snd} : U \to U \ ,$$

where we have:

$$\mathsf{fst}(\mathsf{pair}(x,y)) = x \ ;$$

$$\mathsf{snd}(\mathsf{pair}(x,y)) = y \ ; \text{ and}$$

$$\mathsf{pair}(\mathsf{fst}(z), \mathsf{snd}(z)) \subseteq z \ .$$

This already gives U an algebraic flavor.

Next, by way of abbreviation write:

$$u(x) = \mathsf{rep}(u)(x)$$

for any two elements $u, x \in |U|$. Since every element of U represents a function, we might as well use the function-value notation to make this more transparent. But this process can be reversed (using abs). Suppose we have an expression $\sim x \sim$. where

$$(x \mapsto \sim x \sim)$$

is an approximable mapping from U into U. Again, by way of abbreviation write:

$$\lambda x \ . \sim x \sim = \mathsf{abs}(x \mapsto \sim x \sim) \ .$$

The meaning of this λ-notation is that the λ-abstract on the left is the element of U (chosen by abs) to represent the function

defined by the expression. By way of algebra we have:

$$(\lambda x \ . \sim x \sim)(y) = \sim y \sim \ ; \text{ and}$$

$$\lambda x \ . \ u(x) \subseteq u \ ,$$

which are analogous to our equations and inclusions for the cartesian product $(U \times U)$.

To make this notation useful for the theory of domains, we should also recall that

$$(U + U) \mathrel{\underset{\sim}{\triangleleft}} U \ .$$

We can thus introduce operations on U expressing this structure. To make the equations easier to state, it is useful to remember that also:

$$T \mathrel{\underset{\sim}{\triangleleft}} U \ ,$$

where T is the two-element domain,

$$T = \{\{0\}, \{1\}, \Sigma^*\} \ .$$

We call the corresponding elements of U

$$\mathsf{true} \quad \text{and} \quad \mathsf{false} \ .$$

They are distinguishable, and we can give them helpful rôles by using

$$\mathsf{cond} : U \times U \times U \to U \ ,$$

where cond conditionally chooses between the two elements as follows:

$$\mathsf{cond}(\bot, x, y) = \bot \ ;$$

$$\mathsf{cond}(\mathsf{true}, x, y) = x \ ;$$

$$\mathsf{cond}(\mathsf{false}, x, y) = y \ ; \text{ and}$$

$$\mathsf{cond}(z, \mathsf{true}, \mathsf{false}) \subseteq z \ .$$

The mapping on z indicated in the last line is the projection of U onto (the isomorphic copy of) T. We must, however, assume that the projection actually takes place by saying that the only values of cond(z,true,false) are true, false, and $\bot$.

Returning to $(U + U)$, we can say in an explicit way that there is a copy of this domain inside U by introducing a number of mappings

which, inl, inr, outl, outr

which satisfy a number of equations. For example:

$$\text{outl}(\text{inl}(x)) = x \ ;$$

$$\text{outr}(\text{inr}(y)) = y \ ;$$

$$\text{outl}(\text{inr}(y)) = \perp \ ;$$

$$\text{outr}(\text{inl}(x)) = \perp \ ;$$

$$\text{which}(\text{inl}(x)) = \text{true} \ ;$$

$$\text{which}(\text{inr}(y)) = \text{false} \ ;$$

$$\text{which}(z) = \text{cond}(\text{which}(z),\text{true},\text{false}) \ ;$$

$$\text{cond}(\text{which}(z),\text{inl}(\text{outl}(z)),\text{inr}(\text{outr}(z))) \subseteq z \ .$$

In other words (the image of) $U + U$ has two parts which the function which has tagged with true and false. (The true side is the left side and the false side is the right side.) But $U \lhd U + U$ in two ways: we can inject an element on the left by inl and on the right by inr -- and if we take them out again, nothing is lost.

We can make an essential reduction in notation if we recall the isomorphism

$$(U \to (U \to U)) \cong ((U \times U) \to U) \ .$$

Using this together with $(U \to U) \lhd U$ as embodied in the λ-notation, we can regard all the functions just introduced as *elements* of U. To put it another way, instead of writing:

$$\text{pair}(x,y) \ ,$$

we write:

$$\text{pair}(x)(y) \ .$$

Or more generally we can define:

$$u(x,y,\ldots,z) = u(x)(y) \ \ldots \ (z) \ .$$

We must stress that in this reduction of all the constructs to elements of U we make full use of $u(x)$ as a binary operation on U -- but it is the *only* binary operation we need. The structure on U, therefore, that we are emphasizing consists of

$u(x)$ and $\lambda x \ . \sim x \sim$

together with these *constants*:

fix, sub ;

cond, true, false, $\perp$;

pair, fst, snd ; and

which, inl, inr, outl, outr .

Reductionism could actually be pushed much further if we so wanted. For example, we could just as well *define* pair by;

$$\text{pair} = \lambda x \lambda y \lambda t \ . \ \text{cond}(t,x,y) \ ,$$

but this is a bit artificial. Though we could eliminate all the last eight constants in our list above, we would just have to introduce them again at once by definition, because they are so basic.

So, however we choose to set it up, U becomes an algebra in which all kinds of useful constructs can be defined with the aid of certain constants and the λ-notation. U is an effectively given domain, and every operation defined in this algebra is not only *approximable* but also *computable*. (With a couple more constants, by the way, we could assure that *all* computable elements and operations on U are λ-definable. But this completeness is not especially relevant to the present discussion.) The λ-notation with these constants becomes a mathematically clean *programming language* for defining a very rich collection of computable notions.

Because U is a mathematically defined domain, what we have been giving is the mathematical semantics (meaning) for the definition language. It is quite another question as to how you *find* the value of some long expression written in this notation -- this is where computer science comes in properly, since it is reasonable to ask how we can use the language, how we can manipulate the expressions. Computers are needed because the calculations with the expressions are too long (*far* too long) for pencil and paper. What mathematics has done (within the theory of certain posets) is to show that the expressions *have* a value, that it makes sense to ask whether two expressions *have the same* value. The answers to these mathematical questions are neither obvious from the notation itself nor from a few rules that have been suggested for general manipulations. The theory of domains (or something similar) is needed to make the questions precise and interesting. And there can be more than one answer,

since, as we have seen, there is much freedom in setting up a universal domain like U.

To close the circle of discussion started with our introduction of effectively given domains presented in this section, we still have to show how reduction to (elements of) U is an aid to solving domain equations. The answer all has to do with the use of projections. We recall that projections $a = i \circ j$ coming from projection pairs were the right kind of retracts for thinking of the $\mathcal{D} \triangleleft U$. For short let us call them *finitary* projections. What is nice about U and its algebra is:

THEOREM. *There are computable maps $x, +, \to$ defined on $U \times U$ such that whenever a and b are finitary projections, then so are $(a \times b)$, $(a + b)$ and $(a \to b)$ and we have*

$$\mathcal{D}_{a \times b} \cong \mathcal{D}_a \times \mathcal{D}_b \ ;$$

$$\mathcal{D}_{a+b} \cong \mathcal{D}_a + \mathcal{D}_b \ ; \text{ and}$$

$$\mathcal{D}_{a \to b} \cong \mathcal{D}_a \to \mathcal{D}_b \ .$$

Indeed we can define:

$$a \times b = \lambda t \ . \ \mathsf{pair}(a(\mathsf{fst}(t)), b(\mathsf{snd}(t))) \ ;$$

$$a + b = \lambda t \ . \ \mathsf{cond}(\mathsf{which}(t), \mathsf{inl}(a(\mathsf{outl}(t))), \mathsf{inr}(b(\mathsf{outr}(t)))) \ ;$$

$$a \to b = \lambda t \ . \ b \circ t \circ a \ ;$$

where in general

$$f \circ g = \lambda x \ . \ f(g(x)) \ .$$

The collection of finitary projections becomes a category when we define

$$f : a \to b \ \textit{iff} \ f = b \circ f \circ a \ .$$

It is equivalent to the cartesian closed category of countable neighborhood systems and approximable mappings. We also find that

$$\mathcal{D}_a \textit{ is effectively given } \textit{iff } a \textit{ is computable} \ . \ \square$$

The consequence of this theorem is that the relevant parts of the whole theory of domains are "internalized" into U and into its algebra of elements. The programming language that the algebra gives us is not only a definition language for computable functions

at all types, but it is also a language for *defining the types themselves*. For example, the least fixed point of the equation

$$b = b + b \ ,$$

does in fact give a finitary projection b where

$$\mathcal{D}_b \cong B \ .$$

Other domain equations can be solved in a similar way using the ordinary fixed-point operator

$$\mathsf{fix} : (U \to U) \to U \ .$$

References [7] and [10] contain more details.

REFERENCES

[1] H.P. Barendregt (1981) *The Lambda Calculus: Its Syntax and Semantics*, North-Holland Publishing Co., xiv + 615 pp.

[2] G. Gierz, K.H. Hofmann, K. Keimel, J.D. Lawson, M. Mislove, and D.S. Scott (1980) *A Compendium of Continuous Lattices*, Springer-Verlag, xx + 371 pp.

[3] M.J. Gordon (1979) *The Denotational Description of Programming Languages*, Springer-Verlag, 160 pp.

[4] M.J. Gordon, A.J.R. Milner, and C.P. Wadsworth (1979) *Edinburgh LCF*, Springer-Verlag Lecture Notes in Computer Science, vol. 78, 159 pp.

[5] A. Kanda and D. Park (1979) When are two effectively given domains identical?, *Proc. of the 4th GI Theoretical Computer Science Symposium*, Springer-Verlag Lecture Notes in Computer Science, vol. 67, 170-181.

[6] D.S. Scott (1976) Data types as lattices, *SIAM J. Comput.* 5, 522-587.

[7] D.S. Scott (1977) Logic and programming languages, *Comm. ACM* 20, 634-641.

[8] D.S. Scott (1980) Lambda calculus: Some models, some philosophy, in: *The Kleene Symposium* (J. Barwise, et al., eds.) North Holland Publishing Co., 223-265.