DOMAINS FOR DENOTATIONAL SEMANTICS

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

Abstract. The purpose of the theory of domains is to give models for spaces on which to define computable functions. The kinds of spaces needed for denotational sematics involve not only spaces of higher type (*e.g.* function spaces) but also spaces defined recursively (*e.g.* reflexive domains). Also required are many special domain constructs (or functors) in order to create the desired structures. There are several choices of a suitable category of domains, but the basic one which has the simplest properties is the one sometimes called *consistently complete algebraic cpo's.* This category of domains is studied in this paper from a new, and it is to be hoped, simpler point of view incorporating the approaches of many authors into a unified presentation. Briefly, the domains of elements are represented set theoretically with the aid of structures called *information systems.* These systems are very familiar from mathematical logic, and their use seems to accord well with intuition. Many things that were done previously axiomatically can now be proved in a straightfoward way as theorems. The present paper discusses many examples in an informal way that should serve as an introduction to the subject.

1. Introduction.

I would like to begin with some personal remarks. When I think of the number of headaches I have caused people in Computer Science who have tried to figure out the mathematical details of the Theory of Domains, I have to cringe. The difficulty in the presentation of the subject is in justifying the level of abstraction used in comparison with the payoff: too often the effort needed for understanding the abstractions does not seem worth the trouble—especially if the notions are unfamiliar or excessively general.

For example, Category Theory, which is much used in discussions of the Theory of Domains, seems far too abstract to many people. On the other hand, Automata Theory—which has often benefited from some use of Category Theory—is quite abstract and, in many aspects perhaps, quite useless, but the level of abstraction is fairly low and the reasons for the definitions are usually evident. Thus, the subject has become by now a standard topic even for undergraduate courses. I believe that Michael Rabin and I had much to do with influencing the subsequent development of the theory by formulating some years ago some basic concepts (many previously known) in a way that made sense to anyone who had been exposed to a little Abstract Algebra, Logic and Set Theory. This expository part of our paper did not require originality, but rather a certain frame of mind and a certain style of writing to put the ideas and problems in sharp relief. (The original paper is Rabin-Scott [1959]; see the excellent historical review of the subject in Greibach [1981].)

Domain Theory has fared less well. I feel I made a mistake in 1969 in using Lattice Theory as a mode of presentation—a mistake as far as Computer Science is concerned. Lattices are very familiar to logicians and those interested in Universal Algebra, and I liked very much the structures I found. But it takes some time to learn the special terminology and to become comfortable with the necessary examples. Indeed, without a stock of examples, it is impossible to have sufficient intuition for making the required constructions. True, some people took to the approach, and the lattice-theoretic definitions were simple enough to motivate. But a much greater percentage of people I came in contact with found the large number of things to learn a definite roadblock to understanding. And the Lattice Theory was only partly standard, since this approach to the theory has to employ complete lattices and topological notions in order to explain *limits* and *continuity* adequately. In fact, Topology of a rather mathematically simple set-theoretical kind could have been used as the whole foundation much as in Scott [1972], but that kind of mathematics seems even less attractive to computer scientists. Again, it is a matter of background, I think, and it is unreasonable to expect people to know everything. Nevertheless, the lattice-theoretic approach has great consistency and coherence; it has been carried out in full detail in Gierz, *et al.* [1980], which also gives the topological connections. That book contains a large number of references to the mathematical literature to show the connections of the theory to many other topics. What we had to miss out there is a clear hook-up with computability theory, owing simply to a lack of space and time to be able to cover everything.

In Scott [1981], I tried out another, different presentation (which is also explained in Scott [1982].) However, I have found that the use of *neighborhood systems* causes confusion in people not used to thinking in terms of sets of sets of sets of sets . . . , even though the set theory required is quite simple. My purpose in the present lecture, then, is to go back once more to the very beginning to try out another plan for making the story easy and natural. The notion to be used I call an *information system.* This is a "static" notion appropriate to bodies of information about coherent groups of elements. The "dynamic" ideas enter in the ways different systems can be related and in the semantical definitions about meanings of programming language constructs. (For discussions of semantical definitions see Stoy [1977], Gordon [1979], and Tennent [1981].) Neighborhood systems are, in a precise sense, equivalent to information systems, but there seems to be an interesting trade-off between what properties of structures are *axiomatized* and what are *implicit* in the form of the structure. Perhaps, despite their simplicity, neighborhoods leave too much implicit, too much to be extracted by definition; information systems, on the other hand, do seem to be more flexible in doing several of the important constructions. It all comes down to just where one imposes the closure conditions on the structures. I have a feeling that I now have found just about the right mix of axioms *vs.* definitions.

Another word about Category Theory: I actually feel that it is particularly significant and important for the theory and for the whole area of semantics. But it must be approached with great caution, for the sheer number of definitions *and* axioms can try the most patient reader. It seems to me to be especially necessary in discussing applications of abstract mathematical ideas to keep the motivation strongly in mind. This is often hard to do if the categories get too thick, but of course it all depends on the writer. Category Theory is especially useful in stating *general* properties of structures and in *characterizing* constructions uniquely; however, there often is a problem actually justifying the *existence* of certain constructions, and a direct approach can be quicker than quoting lots of theorems. But, man cannot live by construction alone: theorems have to be proved in order to get the proper value out of the work. Domain Theory must also be convenient for demonstrating the soundness of various proof rules for properties of recursively defined objects and recursively defined domains, and I think that Category Theory can be helpful here. A step in the right direction has been made in the LCF system (see Gordon-Milner-Wadsworth [1979]), which, however, does not take advantage of general Category Theory; but the whole area needs much more development in my opinion.

The main purpose of the Theory of Domains in Denotational Semantics, as I see it, is to give a mathematical model of a system of *types*—including function-space types—and to explain a notion of *computability* with respect to these types. There are many possible theories of types, but the construction of domains is meant to justify *recursive definitions* at all types, and—most essentially—to justify recursive definitions *of* types. Pursuing these goals has certain consequences, as we shall see. The model presented here is only one approach, and some comparisons with other methods will be made in Section 8. Again some help from Category Theory would be welcome in comparing the different kinds of modelling.

One benefit of Domain Theory is that it is possible to make sense of types containing *infinite* elements. The cost required seems to be that the domains therefore have to contain *partial* elements as well as *total* elements. And this is where the lattice-theoretic definitions entered in the earlier presentations: the partial ordering with relations like $x \sqsubseteq y$ was used to express the fact that the element x was "less defined" (*i.e.* more partial) than y but "contained in" y. The relation $\sqsubseteq$, however, must be subjected to many axioms to have a theory suitable to the desired applications. Instead of such a big group of axioms, I wish to put forward here a construction where the "skeleton" (or better: "backbone") of a domain is introduced by just a few axioms. Then the domain itself is *defined* as a certain construct from the backbone in order to define the appropriate notion of element. There are several advantages I can see to the new approach:

- *Simple* definitions of the basic concepts can be given.
- Detailed properties are *proved* as theorems rather than assumed as axioms.
- Emphasis can be given to the *constructive* nature of the definitions.
- Domains can be made more *visible.*
- The theory of domains is made more available for *applications,* because it is easier to produce the needed complex domains.

It is to be hoped that the reader can judge the validity of these claims from the perhaps too brief exposition to follow.

2. Information systems.

Intuitively, an *information system* is a set of "propositions" that can be made about "possible elements" of the desired domain. We will assume that sufficiently many propositions have been supplied to distinguish between distinct elements; as a consequence, an element can be constructed abstractly as the *set* of all the propositions that are true of it. Partial elements have "small" sets; while total elements have "large" sets (even: maximal). To make this somewhat rough idea precise, we have to explain—by a suitable, but small choice of axioms—how the collection of all propositions relevant to the domain hangs together, or better, is structured as a set of abstract propositions. Fortunately, the axioms for this structure are very simple and familiar, which is a great help in making up examples.

DEFINITION 2.1. An *information system* is a structure

$$(\mathcal{D}, \Delta, \mathrm{Con}, \vdash),$$

where $\mathcal{D}$ is a set (the set of *data objects* or *propositions*), where Δ is a distinguished member of $\mathcal{D}$ (the *least informative* member), where Con is a set of finite subsets of $\mathcal{D}$ (the *consistent* sets of objects), and where $\vdash$ is a binary relation between members of Con and members of $\mathcal{D}$ (the *entailment relation* for objects). Concerning Con, the following axioms must be satisfied for all finite subsets $u, v \subseteq \mathcal{D}$:

(i) $u \in \text{Con}$, whenever $u \subseteq v \in \text{Con}$;

(ii) $\{X\} \in \text{Con}$, whenever $X \in \mathcal{D}$; and

(iii) $u \cup \{X\} \in \text{Con}$, whenever $u \vdash X$.

Concerning $\vdash$, the following axioms must be satisfied for all $u, v \in \text{Con}$, and all $X \in \mathcal{D}$:

(iv) $u \vdash \Delta$;

(v) $u \vdash X$, whenever $X \in u$; and

(vi) if $v \vdash Y$ for all $Y \in u$ and $u \vdash X$, then $v \vdash X$.

In words we may say of Con that, as a set of sets, (i) it is closed under subsets, (ii) it contains all singletons, and (iii) adjunction of an entailed object to a consistent set preserves consistency. Concerning $\vdash$, which should be viewed as a "multiary" relation, we may say that (iv) Δ is entailed by anything, (v) $\vdash$ is reflexive, and (vi) $\vdash$ is transitive. (The last two properties are both expressed in a way appropriate to a multiary relation on members of $\mathcal{D}$.) ∎

The best advice is to think of the members of $\mathcal{D}$ as consisting of *finite* data objects, some of which are more informative than others. The word "finite" should be taken here in the sense of "fully circumscribed"—as regards what is given in **A** these data objects can be comprehended in "one step." It is of course possible to introduce information systems where the data objects are infinite sets, but *relative* to **A** they are finite as data objects.

The member that provides *zero information* is Δ. Data objects are intended to give information about possible *elements* of the domain to be constructed, so if we were to use Δ alone, we would be describing the the least defined element usually written as $\perp$.

Not just any combination of data objects will describe a possible element, however; hence, the need for the notion of *consistency*. If $u \in \text{Con}$ is false, then the "propositions" in u cannot all be applied to the same element at the same time. Finally we have to agree that, in general, the propositions to be allowed are rarely mutually independent. It follows that an entailment relation must be imposed to record the dependencies that do hold among the propositions. With these informal understandings, the axioms chosen should all be self-evident.

A first example. Suppose we let $\mathcal{D}$ be the set of non-negative integers, where we think of an integer n as an abbreviation of the proposition $n \leq x$. Here x is a yet-to-be-determined element, about which one proposition gives only a little information. We can identify Δ with 0, and take Con to be the set of all finite subsets of $\mathcal{D}$. The entailment relation can be defined formally in the way suggested by the intuitive reading of the data objects:

$$\{n_0, \ldots, n_{k-1}\} \vdash m \text{ iff either } m = 0 \text{ or } m \leq n_i \text{ for some } i < k.$$

(Remember to think of the same possible x on both sides of the $\vdash$.) That $\vdash$ is an entailment relation in the sense of our axioms is clear. ∎

A second example. The first example is possibly misleading because all (finite) sets of data objects were allowed to be consistent. The example can be modified in a natural way so this is not so; of course, quite a different system will be obtained. The idea is to let $\mathcal{D}$ be the set of all *pairs* (n, m) of integers with $n \leq m$, where such a pair stands now for the proposition $n \leq x \leq m$. Clearly the two data objects (0,2) and (3,7) taken together are *inconsistent.* (Why?)

Oh, I see, I have left out Δ from $\mathcal{D}$; we must therefore include the somewhat artificial pair $(0, \infty)$ with the obvious interpretation. When I say "obvious interpretation" here I mean that $u \in \text{Con}$ can be defined by saying that there must be an integer satisfying all the "propositions" in u. Further, $u \vdash X$ can be defined by saying that *whenever* an integer satisfies all the propositions in u, *then* it must satisfy X. The notion of "satisfy" really should be obvious from the intuitive reading given to the data objects, and the reader can verify easily that all the axioms hold for this example of an information system. ∎

A third example. Let A and B be two fixed sets, and let $\mathcal{D}$ as a set consist of the ordered pairs (a, b), with $a \in A$ and $b \in B$, plus the extra object Δ. Here the "information" contained in (a, b) is that a is mapped to b by a yet-to-be-determined *function.* With this thought in mind, we will know that a finite set of data objects is consistent just in case it is possible that they can all belong to the *same* function. Formally, we can assert:

$$\{(a_0, b_0), \ldots, (a_{k-1}, b_{k-1})\} \in \text{Con iff for all } i, j < k \text{ whenever } a_i = a_j, \text{ then } b_i = b_j.$$

It is something of a bother having to throw in Δ in this case, but all we have to say is that if u is consistent under the above rule, then so also will $u \cup \{\Delta\}$; and these are the only consistent sets of data objects for this example. Perhaps it would make more sense to use $\{(a, b)\}$ in place of the simpler (a, b), and then we could set $\Delta = \emptyset$ more naturally. From this point of view we can regard each data object as a (very small!) fragment of the graph of a function; the consistent sets then point to larger fragments of graphs.

The definition of the entailment relation is the minimal one:

$$u \vdash X \text{ iff either } X = \Delta \text{ or } X \in u.$$

These examples show that sometimes the main part of the structure is in Con, sometimes it is in $\vdash$, and sometimes it is in the interplay between the two notions. The object Δ is not in itself very important, but it is sometimes useful to have an object there that can always be counted upon to act in the same way in every system. ∎

Already in formulating the axiom about the transitivity of $\vdash$ a relation between sets in Con was suggested; we now make this official.

DEFINITION 2.2. For $u, v \in \text{Con}$ we write $u \vdash v$ to mean that $u \vdash X$ for all $X \in v$. ∎

PROPOSITION 2.3. *For all $u, v, w, u', v' \in Con$, we have:*

(i) $\emptyset \vdash \{\Delta\}$;

(ii) $u \vdash v$ *implies* $u \cup v \in \text{Con}$;

(iii) $u \vdash u$;

(iv) $u \vdash v$ *and* $v \vdash w$ *imply* $u \vdash w$;

(v) $u' \supseteq u, u \vdash v$, *and* $v \supseteq v'$ *imply* $u' \vdash v'$; *and*

(vi) $u \vdash v$ *and* $u \vdash v'$ *imply* $u \vdash v \cup v'$.

Proof. Obvious. ▮

Remark. It is not worth formulating a formal result at this point, but it should be clear that the properties in 2.3 could have been used as axioms. That is to say, we could have taken $\vdash$ as a binary relation on the set Con, used 2.1 (i)–(ii) and 2.3 (i)–(vi) as axioms. (Actually, 2.3(ii) is redundant.) The old-style entailment relation would then be definable by:

$$u \vdash X \text{ iff } u \vdash \{X\}.$$

The previous axioms in 2.1 are then easily proved from the new ones. ▮

Some notation. As it is not always clear what structure is meant if we refer to a domain simply by naming its underlying set of data objects, it will be better if we use a letter to refer to an information system *as a whole.* We shall therefore write:

$$\mathbf{A} = (\mathcal{D}_{\mathbf{A}}, \Delta_{\mathbf{A}}, \mathrm{Con}_{\mathbf{A}}, \vdash_{\mathbf{A}}),$$

and say that $\mathbf{A}$ is an information system—with its parts as indicated. For a more informal discussion, the notation without subscripts can suffice, but there can be problems when several different systems are involved. ▮

So much for the definition of what a system is as a mathematical structure. The next question is: What is it good for? More precisely, if information systems are the backbones of domains, then where are the elements? That is the topic of the next section.

3. The elements of a system.

Let $\mathbf{A}$ be an information system, and suppose we already had a concept of being an element of $\mathbf{A}$. The data objects are meant to be propositions about elements, so if X is in $\mathcal{D}_{\mathbf{A}}$ and if x is an element, then we can be expected to know what it means to say that X is *true* of x. Since all we are given is the set of data objects $\mathcal{D}_{\mathbf{A}}$, we have to assume that it contains enough objects to distinguish between distinct elements. (If there were not enough of them, we would have to change the set $\mathcal{D}_{\mathbf{A}}$.) Formally, we can write of two elements x and y:

$$x = y \text{ iff all } X \in \mathcal{D}_{\mathbf{A}} \text{ which are true of } x \text{ are also true of } y, \text{ and conversely.}$$

If this principle is accepted, then the elements can be *identified* with the sets of propositions true of them; formally we can assert:

$$x = \{X \in \mathcal{D}_{\mathbf{A}} \mid X \text{ is true of } x\}.$$

There can be no confusion injected here in identifying elements that ought to be distinct, since we are *assuming* that there are enough data objects in the system. If for some reason we felt we ought to have more of them to distinguish between more elements, then we would have to pass to a larger and *different* information system. By agreement, then, the above equation is a tautology.

So, for the sake of simplicity, elements can be taken to be *sets* of data objects. But, we hasten to ask, which sets? The question is really: What are the properties of truth? The answers are well known, and, in fact, we have already used them in our examples in the last section. In words, the set of true propositions about a possible element must be (i) consistent in itself, and (ii) closed under entailment (or *deductively closed*, for short). Condition (i) is clear because we are talking about a *possible* element, not an impossible one. Condition (ii) is acceptable, because, by the very meaning of the word, entailment should be truth preserving. In addition, we are also saying the converse: *any* set having properties (i) and (ii) corresponds to a (partial) element. Here is the formal statement:

DEFINITION 3.1. The *elements* of the information system $\mathbf{A} = (\mathcal{D}_\mathbf{A}, \Delta_\mathbf{A}, \mathrm{Con}_\mathbf{A}, \vdash_\mathbf{A})$ are those subsets x of $\mathcal{D}_\mathbf{A}$ where:

(i) all finite subsets of x are in $\mathrm{Con}_\mathbf{A}$; and

(ii) whenever $u \subseteq x$ and $u \vdash_\mathbf{A} X$, then $X \in x$.

We write $x \in |\mathbf{A}|$ to mean x is an element of the system. This set of elements is the *domain* determined by the given system. An element that is not included in any strictly larger element in the domain is called a *total* element; the set of total elements is denoted by $\mathrm{Tot}_\mathbf{A}$. ∎

Remarks. Any subset of $\mathcal{D}_\mathbf{A}$ satisfying 3.1 (i) can be called consistent, and every consistent set generates an element by closing it under entailment. Note, too, that every element contains $\Delta_\mathbf{A}$ as a member, because the least informative proposition is true of all elements. Moreover, every domain has a least element contained in all other elements. We call it $\perp_\mathbf{A}$ and write formally:

$$\perp_\mathbf{A} = \{X \in \mathcal{D}_\mathbf{A} \mid \{\Delta_\mathbf{A}\} \vdash_\mathbf{A} X\}.$$

In the above we could just as well have used the empty set $\emptyset$ in place of $\{\Delta_\mathbf{A}\}$; and often we can write simply $\vdash_\mathbf{A} X$.

The element $\perp_\mathbf{A}$ is often called the *bottom* element of the domain. There need be no *top* or maximal element, $\top_\mathbf{A}$. Such an element is possible if, and only if, *all* finite subsets of $\mathcal{D}_\mathbf{A}$ are consistent, in which case, as a set, $\top_\mathbf{A} = \mathcal{D}_\mathbf{A}$. The possibility is not excluded—rather, it is not required. In case $\top_\mathbf{A}$ exists, it is the unique total element of the domain, and conversely. ∎

Returning to our question of the balance between axiomatizaton and construction, what we have done is first to axiomatize the properties of $(\mathcal{D}_\mathbf{A}, \Delta_\mathbf{A}, \mathrm{Con}_\mathbf{A}, \vdash_\mathbf{A})$, then to construct the domain $|\mathbf{A}|$ from this structure. The principal claimed advantages are that the axioms for consistency and entailment are already essentially familiar, and that the definition of elementhood is direct and natural from our understanding of the properties of truth.

The examples. Let us look again at the three examples of information systems of the last section. (1) In the first we see that the elements, by the formal definition, are either the finite sets $\{n \mid n \leq m\}$ or the whole set (*i.e.* $\top$). Intuitively, this corresponds to the fact that the chosen propositions only give *lower* bounds on a possible element; thus, no element is "finished" until it becomes infinite. (2) In the second example there is no top, and the elements are the sets of the form:

$$\{(n, q) \mid n \leq m \leq p \leq q\},$$

where $m \leq p$ are given, and where $q = \infty$ is allowed. The total elements here correspond to the (non-negative) integers, and the partial elements to situations where the lower and upper bounds have not come exactly together. (3) In the third example, the elements are just the graphs (*i.e.* sets of ordered pairs) of partial functions from A into B (with the object Δ adjoined to the graph). Total elements correspond to total functions (*i.e.* functions well defined on *all* of the set A). ∎

The reader should be able to obtain some more examples of information systems from his own experience and should then consider what the corresponding domain is like. Examples are really quite easy to make up. Needless to say, Mathematical Logic provides a host of information systems, but logicians do not often consider the domains determined by the the systems they know to be worthy of study *in themselves.* In particular, it does not seem to be widely recognized that the domains obtained from information systems form a rich category, as we shall show later. The word "category" is definitely being used here in the precise technical sense of Category Theory, and the application of the notion is very appropriate since the category of domains has very good closure properties.

4. Domains as lattices and as topological spaces.

I will attempt to keep this section as informal as possible—especially as Sections 2 and 3 were quite formal (*i.e.* mathematical) enough. The main purpose of the discussion of this section is to relate the new presentation of the Theory of Domains to previously published ones. (Some alternatives are discussed at the end of the paper.)

Lattice-theoretic considerations. Let **A** be an information system. Because the elements of $|\mathbf{A}|$ are introduced as sets, these elements can be given structure from what we know already about ordinary sets. For instance, the set-theoretic inclusion relation between sets can certainly be applied to elements. The question is: What does $x \subseteq y$ mean intuitively? In fact, we have already mentioned this relationship before. It means that every proposition (among the ones given by the information system) true of x is also true of y (though not necessarily conversely). In other words, x is perhaps only partially determined, while y is more fully determined in a way that includes everything true of x. Clearly, then, if we grant the interest of partial elements, then this relation is a natural and basic one. We often read "$x \subseteq y$" as "*x approximates y*".

Now here is one of the main points of the new exposition: Because $x \subseteq y$ is *defined* in terms of a familiar mathematical notion, then as a relation between elements it inherits all the well-known properties of the set-theoretical relation. It should not be necessary to write them out, for everyone who can read this far knows that $\subseteq$ is in particular reflexive and transitive. We say that the domain $|\mathbf{A}|$ is a *partially ordered set under inclusion.* We note that this (trivial) result is *proved* from the definition rather than assumed as an axiom.

Let us try out another notion. Elements are consistent, deductively closed sets of data objects. Right? Every subset of a consistent set is consistent, but not every subset is closed under entailment, in general. Right? But suppose that x and y are two elements. It can easily be seen that their intersection $x \cap y$ is again consistent and deductively closed (by a 2 nanosecond proof). Therefore, the set of elements of a domain is closed under intersection. This means that, as a partially ordered set, $|\mathbf{A}|$ is an *inf semilattice.*

Well, the exact terminology is quite unimportant, but the properties of $\cap$ when combined with the properties of $\subseteq$ are pleasantly "algebraic" (and run to several lines when written out). We all know them: $\cap$ is idempotent, commutative, and associative. The element $\perp$, being the smallest element of the domain, is a zero element for $\sqcap$. The operation $\sqcap$ is monotonic with respect to $\sqsubseteq$, and we have the connection:

$$x \sqsubseteq y \text{ iff } x \sqcap y = x.$$

What has just been alluded to in words is the standard mathematical axiomatization of the properties of an infimum (or meet) operation in a partially ordered set. (Remember, however, to have an operation well defined within a set it is necessary also to have *closure* under that operation.)

What about infs of subsets? No problem. For any *non-empty* subfamily of $|\mathbf{A}|$, it is just as easy to prove that the set-theoretical intersection of all the elements in the subfamily is again an element of the domain. This makes $|\mathbf{A}|$ a (conditionally) complete inf semilattice. Even if you do not have any idea what I am talking about, it does not matter because the properties of the domain are built in by the very definition; you do not have to worry *at the start* about their formulation, for you can prove them when you need them (if ever).

What about suprema (or sups or joins)? Here there is a slight problem, because in the way we set things up they *do not* always have to exist. Consider what happens with just two elements x and y. The set-theoretical union of the two elements as subsets of $\mathcal{D}_\mathrm{A}$ need be neither consistent nor deductively closed; and even if $x \cup y$ is consistent, it need not be closed under entailment. So if we want sups *within* $|\mathbf{A}|$, we cannot get by as cheaply as we can with a domain of arbitrary sets for which the simple union is the answer.

Let us write the sup (supposing it exists in $|\mathbf{A}|$) as $x \sqcup y$. What do we have when we actually have it? By the lore of partially ordered sets, $x \sqcup y$ must be the *least element* in $|\mathbf{A}|$ which includes (in the sense of $\sqsubseteq$) both x and y. Now we have just spoken about infs of families of elements: they exist if the family is non empty. This indicates that $x \sqcup y$ exists exactly when there is at least one element z in $|\mathbf{A}|$ such that $x \sqsubseteq z$ and $y \sqsubseteq z$. This turns out to be a way to say—entirely inside $|\mathbf{A}|$—that $x \cup y$ is consistent. In other words (and more generally) a sup of a subfamily exists if, and only if, the union of the family is consistent. (And in that case the sup is just the deductive closure of the union—that is, it is generally larger than the simple set-theoretical union.) This has an axiomatic version, but it is slightly complicated by the fact that sups do not always exist. In case $\top_\mathrm{A}$ exists, then we always have all the sups, and $|\mathbf{A}|$ is a *complete lattice.* (This is discussed in full axiomatic splendor in Gierz, *et al.* [1980].) But, let it hasten to be added, not every complete lattice is isomorphic to a domain. The lattices that correspond to domains are the so-called *algebraic lattices.* (See the discussion *loc. cit.* for the additional axiomatics required.) When the top is missing, the partially ordered structures corresponding to domains are called *conditionally complete, algebraic cpo's.*

There is, however, an important case in $|\mathbf{A}|$ where the union is consistent and is, in fact, the sup in the domain. Suppose we have a sequence of elements such that

$$x_0 \sqsubseteq x_1 \sqsubseteq x_2 \sqsubseteq \cdots \sqsubseteq x_n \sqsubseteq x_{n+1} \sqsubseteq \cdots.$$

As n increases, we can say that the elements x_n are getting "better and better"; thus, they must be approaching something even more desirable *in the limit.* Let

$$y = \bigcup_{n=0}^{\infty} x_n,$$

then there is no question that y is a subset of $\mathcal{D}_{\mathrm{A}}$. But is it an element? Consistency, recall, means that every *finite* subset is in $\mathrm{Con}_{\mathrm{A}}$. The trick is that a finite subset of y must be a subset of one of the x_n, because the sequence of elements x_n is increasing. But all the terms of our given sequence are elements, and so, consistent; therefore, every finite subset of y is consistent.

The same argument also shows that the set y of data objects is closed under entailment, because $\vdash_{\mathrm{A}}$ is defined as a relation between finite (consistent) sets and data objects. It follows that since each x_n is a set closed under $\vdash_{\mathrm{A}}$, then so is y. In other words, y is indeed an element. Otherwise said, the domain $|\mathrm{A}|$ is closed under the formation of unions of increasing chains of elements, and the union is, of course, the sup in the sense of the partial ordering. Again we have proved closure rather than assuming it, and the necessary properties of the sup follow from its definition as a union.

This last argument can be generalized—as is well known to mathematicians—to the case of *directed* families of elements. The word "directed," or better "upward-directed," means that every finite number of elements in the family is included in some further element of the family; chains always have this property. A good example of the use of directed sets of elements figures in the discussion of the *finite* elements of the domain.

As we have remarked several times, any consistent set of data objects *generates* an element by closing it up under the entailment relation; thus, in particular, any set $u \in \mathrm{Con}$ generates an element. Let us write

$$\overline{u} = \{X \in \mathcal{D}_{\mathrm{A}} \mid u \vdash_{\mathrm{A}} X\}$$

for the *closure* of u under entailment. (This notation could be used for any consistent subset of $\mathcal{D}$, but the case of finitely generated elements is especially important.) Such a $\overline{u}$ is always an element of $|\mathrm{A}|$; the totality of such elements form, by definition, the *finite elements* of the domain. The notation $\overline{u}$ should of course be decorated with a subscript A, but, as there is no good place to write it in, we leave it out.

We have for all elements x of the domain the basic formula:

$$x = \bigcup\{\overline{u} \mid u \in \mathrm{Con}_{\mathrm{A}} \text{ and } u \subseteq x\}.$$

This can be read intuitively as saying that *every element of the domain is the limit of its finite approximations.* And, having used the word "limit" so often, we ought now to say something about how to make the meaning of this word correct mathematically. Note that in the limit representation for x, the union is a directed one. (Why?)

Topological considerations. Geometrically speaking, a topological space is a collection of "points" which group themselves in various "neighborhoods" providing thereby a sense of "nearness." More specifically, a neighborhood of a point in a *metric* space consists of all those points within a certain positive "distance" from the given point; that is, nearness is limited, but it is not pushed to *zero.* The major reason for leaving some "breathing space" around the point comes out in defining *continuity* of functions.

A function f from one topological space to another is said to be *continuous* provided that, for every point x of the first space, and for every neighborhood in the second space around $f(x)$, it is possible to find—in the first space—a neighborhood around x with the following property. If the function is restricted to this neighborhood, then its values lie entirely in the given neighborhood around $f(x)$. In the metric case we can say that if we do not want the function to wander any great distance from $f(x)$, then there is a restricted distance around x giving a neighborhood over which the function values stay as close to $f(x)$ as specified. This keeps the function from jumping around wildly, since small variations in x lead to only small variations in $f(x)$.

The intuition about continuity is no doubt very clear in the more geometric examples, and gometric intuition can carry us quite far. But the spaces obtained from domain theory are not really geometric spaces, so we need to make some shifts in our metaphors. The geometric notion of a point usually implies that two points are perfectly distinguishable: if x and y are different, then they have *disjoint* neighborhoods. This is certainly true in the metric case, where, if two points are different, then they are at a positive distance apart. Early on in the study of topological spaces, however, it was found that not all spaces were metric, and weaker separation properties of pairs of distinct points were uncovered. One of the very weak versions of such a condition is called the "T_0–axiom." The best way to put it is that if two points have the *same* neighborhoods, then they are the same. The contrapositive statement sounds odd: if two points are distinct, then there is a neighborhood that contains one but not the other. The oddity of this statement lies in the feeling that you cannot make up your mind over which is the better point.

In our domains there is already a notion of "betterness" which proves to be very closely related to the implicit neighborhood structure. The reason why our domains, as spaces of points, are not "geometric" in the familiar sense is that they contain *partial elements.* A geometric point on the other hand is "perfect" or totally determined; it cannot be made better than it already is. The metric in common spaces is measuring something other than betterness, for there are competing notions of approximation afoot here. Thus, we must keep our special goals in defining domains clearly in mind to avoid confusion.

Consider an information system $\mathbf{A}$. For each $u \in \mathrm{Con}_{\mathbf{A}}$, we define a corresponding *neighborhood* of $|\mathbf{A}|$ by the equation:

$$[u]_{\mathbf{A}} = \{x \in |\mathbf{A}| \mid u \subseteq x\}.$$

The neighborhoods *of* an element x are all those sets $[u]_{\mathbf{A}}$ where $u \subseteq x$. Note that this is the same as saying $\overline{u} \subseteq x$ or $\overline{u}$ approximates x. Indeed, the neighborhoods are in a one-one correspondence with the finite elements of $|\mathbf{A}|$. A neighborhood collects together all those elements sharing the same finite amount of information.

If both $u \subseteq x$ and $v \subseteq x$, then it is easy to see that the intersection of the two neighborhoods is again a neighborhood of x by virtue of the formula:

$$[u]_{\mathbf{A}} \cap [v]_{\mathbf{A}} = [u \cup v]_{\mathbf{A}}.$$

In case u and v are inconsistent with each other, then the intersection will be the empty set $\emptyset$. Because every element of our domain $|\mathbf{A}|$ is the union of the finite elements it contains, it is trivial to prove that two elements with exactly the same neighborhoods are the same. For these reasons, then, it follows that $|\mathbf{A}|$ is a T_0–topological space. The immediate reaction to this intelligence is: So what?

The answer to this scepticism will appear in the best light when we discuss in the next section the notion of *function* or *mapping* appropriate to our domains. The argument will be based on the extremely elementary form of the definition that characterizes the general continuous function together with the fact that the geometric intuition is suggestive of theorems to prove. In Section 6 we will find that the space of all continuous functions between two given domains is also a domain, a most useful result that is interesting topologically. But further interest for Domain Theory comes from the circumstance that there is a connection between geometry and domains. We cannot go into the full details here, but I can make some brief remarks.

Before continuing that discussion, though, it should be remarked that once the topology of the domain has been uncovered, then the unions of chains of elements (or directed sets of elements) *are* topological limits according to accepted general definitions. Thus, what we felt intuitively was a limiting process (the getting better and better up to the union) is in fact a limit formation in a precise mathematical sense. This helps convince us that we are on the right track.

Total elements as a space. In any domain it can be proved that any element x can be extended to a *total* element t, which is characterized by the property that it is a *maximal* element of the partial ordering. (Perhaps there are many such t, but there is always at least one with $x \subseteq t$.) Recall that the set of all total elements of $|\mathrm{A}|$ is denoted by Tot_A. Because the set $\mathrm{Tot}_\mathrm{A} \subseteq |\mathrm{A}|$, it is a topological space itself. Suppose s and t are two distinct total elements, then they must be inconsistent with each other—but we shall prove more. If every finite consistent $u \subseteq s$ also satisfied $u \subseteq t$, then $s \subseteq t$ would follow. But this is impossible, because s is maximal. Let, then, $u \in \mathrm{Con}$ with $u \subseteq s$ be chosen so that $u \subseteq t$ fails. Now u must be inconsistent with t, since otherwise there would be an element containing them both. It follows that u must be inconsistent with some $v \in \mathrm{Con}_\mathrm{A}$ with $v \subseteq t$. In this way we show that the neighborhoods $[u]_\mathrm{A}$ and $[v]_\mathrm{A}$ are disjoint.

This argument proves that Tot_A is a *Hausdorff space* (or T_2–space). We cannot go further into the details here, but it is also a totally disconnected, compact Hausdorff space. (These spaces—well known from studies of Logic and the Theory of Boolean Algebras—are also zero-dimensional.) Assuming the countability of Con_A, which is not such a bold assumption, all such spaces can be conveniently embedded into the real line so as to preserve the topological structure. Perhaps this does not seem so surprising, but it shows that the topological nature of Tot_A, the uppermost level of the T_0 space $|\mathrm{A}|$, is indeed quite geometric. And it can also be shown that any arbitrary continuous function on Tot_A into *any* topological space can be extended to a continuous function on $|\mathrm{A}|$, so the whole domain proves to be quite a "roomy" space with many good connections to more usual topological spaces.

A note on neighborhood structures. Topological spaces can be defined in several ways, and in general they do not carry a preferred neighborhood structure. (For example,there is no topological significance to the the usual rational intervals we use to define neighborhoods for the real line: any similarly dense set of points would do in place of the rational numbers.) Domains, the way we have defined them, however, do have preferred neighborhoods. The set of finite elements of $|\mathrm{A}|$ can be defined topologically, and as we have seen they determine a convenient set of neighborhoods. (The reason for all this is the so-called zero-dimensonal character of the domains. There is a higher-dimensional theory, but this leads to continuous lattices and can only be explained using the kind of presentation in Gierz, *et al.*[1980]. It is too long-winded a story for the present paper. The higher-dimensonal domains do not have a preferred set of neighborhoods, however.)

Having realized this, it is a short step to giving an algebraic axiomatization of the neighborhood structure. Smyth [1975] contains a presentation in terms of the equivalent notion of finite element. Scott [1981] turns the story around and defines a set-theoretical representation of a neighborhood as just a family of sets containing a largest "neighborhood" and closed under forming the intersection of any two sets in the family that contain a third. The set-theoretical form of this definition is very simple, and Scott [1981] spells out the details of how the theory of domains can be based on this starting point. Subsequent investigation, however, has convinced the author that the present approach through information systems is even simpler and better for a number of the essential constructions. The theory introduced by Ersov [1970] of F-spaces is another axiomatization somewhat intermediate between the topological and the lattice theoretic. Domains in the present sense then become *complete* F-spaces. There are many interesting aspects to this method, but the author feels that information systems are about as elementary and familiar as we can get, and they are therefore more suggestive of new constructions.

5. Approximable mappings between domains. Once a general notion of set or domain has been defined mathematically, the next major issue is: How are the different domains to be related one to another? The answer to this important question can many times be given by defining an appropriate concept of *mapping* between domains. The answer need not be unique; the same collection of domains may support more than one idea of function depending on the special aspects of the domains that need to be brought out. In the case of the kinds of domains being studied here, there are two principal answers, one of which is to be explained in this section.

In order to understand an appropriate notion of a mapping or a function between domains constructed from information systems, we have to refer back to the way in which elements are introduced. As we have seen in Section 3, an element is a consistent, closed set of data objects. To generate an element, one has to generate more and more of the finite consistent subsets of the element. Note that the separate data objects have to be grouped into these finite consistent sets, because the entailment relation may require a finite set of arbitrary size on the left in making the necessary entailments for closure. It is this kind of passage from a finite set to its closure that is to be generalized in defining mappings.

Consider two domains. To map from one to another, some information about a possible element of the first is presented as *input* to the function f. Then as *output* the function f starts generating an element. If the input were u, a consistent set of the first domain, then *part* of the output in the second domain might be a consistent set v. We could say that there is an input/output relationship set up by f, and indicate by $u\ f\ v$ that this relation holds going from u in the first domain to v in the second. Of course to get the full effect of f, it is necessary to take *all* the v's related to a given u, because even a small finite amount of input may cause an infinite amount of output. But every element of a domain is just the sum total of its finite subsets (finite approximations), so it is sufficient to make the mapping relationship go between finite sets. Here is the formal definition with the exact conditions that the relation f must satisfy.

DEFINITION 5.1. Let $\mathbf{A}$ and $\mathbf{B}$ be two given information systems. An *approximable* mapping $f : \mathbf{A} \to \mathbf{B}$ is a binary relation between the two sets $\mathrm{Con}_{\mathbf{A}}$ and $\mathrm{Con}_{\mathbf{B}}$ such that:

(i) $\emptyset\ f\ \emptyset$;

(ii) $u\ f\ v$ and $u\ f\ v'$ always imply $u\ f\ (v \cup v')$; and

(iii) $u' \vdash_{\mathbf{A}} u, u\ f\ v$, and $v \vdash_{\mathbf{B}} v'$ always imply $u'\ f\ v'$.

We say that **A** is the *source* of f and **B** is the *target*. ∎

Intuitively the relationship $u\ f\ v$ is an input/output passage which can be read informally as: *"if you are willing to give at least u amount of information about the argument, then the mapping f is willing to give at least v amount of information about the value (and possibly even more—if you are patient)."* Condition (i) means that *no* (non-trivial) information about the input merits no information about the output. Condition (ii) implies that all the contributions to the output from a fixed input are consistent, and in fact the union of two of the output sets is again an output set. Output corresponding to a fixed input is *cumulative*. Condition (iii) assures us that the mapping relation f works in harmony with the two entailment relations: if a certain relationship holds, and if the left-hand side is strengthened while the right-hand side is weakened, then the mapping relation must continue to hold. What we must discuss next is what this means for elements.

Before defining the notion of function value, however, it is useful to remark that in the definition of approximable mapping the form of the statement could be simplified by reducing sets on the right to their elements. Specifically, we note the equivalence:

$$u\ f\ v \text{ iff } u\ f\ \{Y\} \text{ for all } Y \in v.$$

In other words an approximable mapping is completely determined by the relation set up between consistent sets on the left and single data objects on the right.

DEFINITION 5.2. If $f : \mathbf{A} \to \mathbf{B}$ is an approximable mapping between information systems, and if $x \in |\mathbf{A}|$ is an element of the first, then we define the *image* (or *function value*) of x under f by the formula:

$$f(x) = \{Y \in \mathcal{D}_{\mathbf{B}} \mid u\ f\ \{Y\} \text{ for some } u \subseteq x\}.$$

Alternatively, we could use the equivalent formula:

$$f(x) = \bigcup\{v \in \mathrm{Con}_{\mathbf{B}} \mid u\ f\ v \text{ for some } u \subseteq x\}. \quad ∎$$

Note that, under Definition 5.2, it has to be *proved* that the image of an element in $|\mathbf{A}|$ lies in $|\mathbf{B}|$ in order to justify the use of the ordinary function-value notation. But both the consistency and the closure under entailment of $f(x)$ are direct consequences of the properties in the definition of approximable mapping.

PROPOSITION 5.3. *Let $f, g : \mathbf{A} \to \mathbf{B}$ be two approximable mappings between two information systems. Then*

(i) *f always maps elements to elements under Definition 5.2;*

(ii) $f = g$ *iff* $f(x) = g(x)$ *for all* $x \in |\mathbf{A}|$; *and*

(iii) $f \subseteq g$ *iff* $f(x) \subseteq g(x)$ *for all* $x \in |\mathbf{A}|$.

Moreover, the approximable mappings are monotone in the sense that

(iv) $x \subseteq y$ *in* $|\mathbf{A}|$ *always implies* $f(x) \subseteq f(y)$ *in* $|\mathbf{B}|$.

All these results follow from the observation that

(v) $\quad u\, f\, v \quad$ *iff* $\quad \overline{v} \subseteq f(\overline{u})$,

for all $u \in \mathrm{Con}_{\mathbf{A}}$ *and all* $v \in \mathrm{Con}_{\mathbf{B}}$. ▮

The proof is straightforward, and it completely justifies the use of the function-value notation. Consequently, the question arises as to whether the characterization of approximable mappings could not have been given in terms of elements in the first place. The answer is yes. Indeed, approximable mappings correspond exactly to functions on elements preserving unions of chains (this, in the case of countable information systems; directed unions must be used in general). This characterization has also been called "continuity," because it is equivalent to saying that the mappings between elements are continuous mappings *in the topological sense*, when $|\mathbf{A}|$ and $|\mathbf{B}|$ are regarded as topological spaces, as explained in the last section. The reason for the word "approximable" is explained in Section 7.

Having justified the idea of an approximable mapping as a transformation on elements, the next step is to combine mappings. It is hardly surprising to learn that *compositions* of approximable mappings are approximable, which in mathematical terms means that the domains together with the approximable mappings form a *category*. The interesting part starts when we want to combine *domains*. But, to fix ideas, it is useful to spell out how compositions take place on the level of the data objects and consistent sets. For instance, it was noted that in any information system $\vdash$ is transitive on Con, and in Definition 5.1 a transitivity condition comes in. There is a perfectly good reason for the parallelism, for, as we see next, the first idea (entailment) is a special case of the second (approximable mapping).

PROPOSITION 5.4. *Let* $\mathbf{A}$ *be an information system. Then the following formula defines an approximable mapping* $\mathrm{I}_{\mathbf{A}} : \mathbf{A} \to \mathbf{A}$:

(i) $\quad u\, \mathrm{I}_{\mathbf{A}}\, v \quad$ *iff* $\quad u \vdash_{\mathbf{A}} v$,

for all $u, v \in \mathrm{Con}_{\mathbf{A}}$. *And we have*:

(ii) $\quad \mathrm{I}_{\mathbf{A}}(x) = x$,

for all $x \in |\mathbf{A}|$. ▮

In other words, the given entailment relation itself defines the identity function on the domain in question; and of course, the identity function is an approximable mapping.

PROPOSITION 5.5. *Let* $f : \mathbf{A} \to \mathbf{B}$ *and* $g : \mathbf{B} \to \mathbf{C}$ *be two approximable mappings. Then the following formula defines an approximable mapping* $g \circ f : \mathbf{A} \to \mathbf{C}$:

(i) $\quad u\, (g \circ f)\, w \quad$ *iff* $\; u\, f\, v$ *and* $v\, g\, w$ *for some* $v \in \mathrm{Con}_{\mathbf{B}}$,

for all $u \in \mathrm{Con}_{\mathbf{A}}$ *and* $w \in \mathrm{Con}_{\mathbf{C}}$. *And we have*:

(ii) $\quad (g \circ f)(x) = g(f(x))$,

for all $x \in |\mathbf{A}|$. ▮

In other words, composition of input/output relations is effected by putting the input into the first, getting some output, and then putting that in as input to the second. The correctness of this recipe on elements (*i.e.* formula 5.5(ii)) follows from the basic formula 5.3(v). Because the class of *all* sets and *arbitrary* mappings forms a category—in the formal meaning of the word—the above three propositions show that the totality of information systems and approximable mappings also forms a category: a sub-category of the category of sets. We have given a special representation to the category by our use of data objects and consistent sets, and we have thereby put the elements into a secondary position—for a good reason. But the elements are there to use. The axioms for a category could also be verified *directly* using the basic definitions like 5.4(i) and 5.5(i); the details are bland. The more interesting topic is: what are the closure conditions on our domains; how do we construct new domains given old ones?

Before we go on to the first basic constructs of sums and products of domains, we remark on some other easy examples of approximable mappings: *constant maps.*

PROPOSITION 5.6. *Given information systems* $\mathbf{A}$*,* $\mathbf{B}$*,* $\mathbf{C}$*, and* $\mathbf{D}$*, and given a fixed element* $b \in |\mathbf{B}|$*, then there is a unique approximable mapping* $\mathrm{const}\, b : \mathbf{A} \to \mathbf{B}$ *such that:*

(i) $(\mathrm{const}\, b)(x) = b,$

for all $x \in |\mathbf{A}|$*. Moreover, we have:*

(ii) $f \circ (\mathrm{const}\, b) = \mathrm{const}\, f(b),$

for all approximable mappings $f : \mathbf{B} \to \mathbf{C}$*; and*

(iii) $(\mathrm{const}\, b) \circ g = \mathrm{const}\, b,$

for all approximable mappings $g : \mathbf{D} \to \mathbf{A}.$ ∎

The proof is immediate, since we need only interpret the input/output relation $u\ (\mathrm{const}\, b)\ v$ as meaning $v \subseteq b$. We note that our notation is somewhat ambiguous since, for example, in 5.6(iii) we are using $\mathrm{const}\, b$ in two senses: once as a mapping from $\mathbf{A}$, and once as a mapping from $\mathbf{D}$. In fact, we should hang *two* subscripts on the thing to fix *both* the source *and* the target of the mapping. This is too painful, however, and we generally rely on context to make the meaning clear. (Formal rules of type checking can be given, so on a computer-based system all the necessary subscripts could be put back in. Thus, the kind of ambiguity we are dealing with here is not very serious.)

6. Products and sums of domains. In the category of sets, the product (the cartesian product) of two sets is by definition just the set of ordered pairs of elements. We could use the same definition in the category of domains—provided we worked in terms of the elements of the domains. The disadvantage is that the determination of the product domain from data objects is lost, or at least pushed into the background. We shall therefore give a construction of products directly in terms of the given data objects, and then show how to prove that the product is just the one expected when we look at the elements.

The idea for the definition is a simple one. Think of two domains $\mathbf{A}$ and $\mathbf{B}$. A data object $X \in \mathcal{D}_{\mathbf{A}}$ can be giving information about a possible element $x \in |\mathbf{A}|$, and a data object $Y \in \mathcal{D}_{\mathbf{B}}$ can be giving information about a possible element $y \in |\mathbf{B}|$. How do we wish to give information about the pair (x, y)? The first piece of advice is not to say all that is known all at once. For example, if pressed, we can say we know X about the first coordinate of the pair; only later, if really pressed, we can say we also know Y about the second coordinate. The point is that data objects provide only partial information, and it does not matter if it takes several of them to give fairly complete information about the whole element. This attitude motivates, then, the particular form of our official definition.

DEFINITION 6.1. Let $\mathbf{A}$ and $\mathbf{B}$ be two information systems. By $\mathbf{A} \times \mathbf{B}$, the *product system*, we understand the system where:

(i) $\mathcal{D}_{\mathbf{A}\times\mathbf{B}} = \{(X, \Delta_{\mathbf{B}}) \mid X \in \mathcal{D}_{\mathbf{A}}\} \cup \{(\Delta_{\mathbf{A}}, Y) \mid Y \in \mathcal{D}_{\mathbf{B}}\}$;

(ii) $\Delta_{\mathbf{A}\times\mathbf{B}} = (\Delta_{\mathbf{A}}, \Delta_{\mathbf{B}})$;

(iii) $u \in \mathrm{Con}_{\mathbf{A}\times\mathbf{B}}$ iff $\mathrm{fst}\, u \in \mathrm{Con}_{\mathbf{A}}$ and $\mathrm{snd}\, u \in \mathrm{Con}_{\mathbf{B}}$;

(iv′) $u \vdash_{\mathbf{A}\times\mathbf{B}} (X', \Delta_{\mathbf{B}})$ iff $\mathrm{fst}\, u \vdash_{\mathbf{A}} X'$; and

(iv″) $u \vdash_{\mathbf{A}\times\mathbf{B}} (\Delta_{\mathbf{A}}, Y')$ iff $\mathrm{snd}\, u \vdash_{\mathbf{B}} Y'$;

where, in (iii), u is any finite subset of $\mathcal{D}_{\mathbf{A}\times\mathbf{B}}$, in (iv′) and (iv″), $u \in \mathrm{Con}_{\mathbf{A}\times\mathbf{B}}$, and we let:

$$\mathrm{fst}\, u = \{X \in \mathcal{D}_{\mathbf{A}} \mid (X, \Delta_{\mathbf{B}}) \in u\}, \text{ and}$$
$$\mathrm{snd}\, u = \{Y \in \mathcal{D}_{\mathbf{B}} \mid (\Delta_{\mathbf{A}}, Y) \in u\}. \quad \blacksquare$$

Note that in 6.1(i) the two sets in the union are just two copies of $\mathcal{D}_{\mathbf{A}}$ and $\mathcal{D}_{\mathbf{B}}$, respectively. There is a shared member, however, the object $(\Delta_{\mathbf{A}}, \Delta_{\mathbf{B}})$, which is indeed the least informative data object—whether it is looked at from the point of view of $\mathbf{A}$ or of $\mathbf{B}$. We could have used all the pairs (X, Y), but if we did, the consistent set $\{(X, Y)\}$ would be deductively equivalent to the set $\{(X, \Delta_{\mathbf{B}}), (\Delta_{\mathbf{A}}, Y)\}$ according to the definition of $\vdash_{\mathbf{A}\times\mathbf{B}}$, and so there is not much point in having this redundancy. Strictly speaking, we should not make these remarks until we have actually verified that 6.1 is indeed a proper definition.

PROPOSITION 6.2. *If* $\mathbf{A}$ *and* $\mathbf{B}$ *are information systems, then so is* $\mathbf{A} \times \mathbf{B}$*, and we have mappings*

$$\mathrm{fst} : \mathbf{A} \times \mathbf{B} \to \mathbf{A} \text{ and } \mathrm{snd} : \mathbf{A} \times \mathbf{B} \to \mathbf{B},$$

such that, for approximable mappings

$$f : \mathbf{C} \to \mathbf{A} \text{ and } g : \mathbf{C} \to \mathbf{B},$$

there is **one and only one** *approximable mapping* $\langle f, g\rangle : \mathbf{C} \to \mathbf{A} \times \mathbf{B}$ *such that*

$$\mathrm{fst} \circ \langle f, g\rangle = f \text{ and } \mathrm{snd} \circ \langle f, g\rangle = g.$$

Proof. Checking that 6.1 does indeed define an information system is straightforward, but there are six things that must be proved according to Definition 2.1. We have to leave the details to the reader. Next using the notation of 6.1, where fst and snd were applied as operations on consistent sets $u \in \mathrm{Con}_{\mathbf{A}\times\mathbf{B}}$, we redefine matters to have approximable mappings, where, for $v \in \mathrm{Con}_{\mathbf{A}}$ and $w \in \mathrm{Con}_{\mathbf{B}}$,

(1) $u \,\mathrm{fst}\, v$ iff $\mathrm{fst}\, u \vdash_{\mathbf{A}} v$;

(2) $u \,\mathrm{snd}\, w$ iff $\mathrm{snd}\, u \vdash_{\mathbf{B}} w$; and

(3) $s \,\langle f, g\rangle\, u$ iff $s\, f\, (\mathrm{fst}\, u)$ and $s\, g\, (\mathrm{snd}\, u)$.

Because we defined $\mathbf{A} \times \mathbf{B}$ so that consistency and entailment worked independently on the two halves of the set of data objects, it is easy to check that (1)–(3) define approximable mappings having the desired properties.

The uniqueness of $\langle f, g\rangle$ comes out of the observation that, if z and z' are two elements of $\mathbf{A} \times \mathbf{B}$ for which

$$\mathrm{fst}(z) = \mathrm{fst}(z') \text{ and } \mathrm{snd}(z) = \mathrm{snd}(z'),$$

then $z = z'$. The reason is that fst and snd just divide elements into the two kinds of data objects, and then strip off the parentheses. (Look back at Definition 6.1.) No information is lost, so if z and z' are transformed into the same elements both times, then they have to be the same.

That lemma treats one pair of elements at a time, but $\langle f, g\rangle$ is a function. But if $\langle f, g\rangle'$ were another function satisfying the conditions of the above proposition, then the two functions would be pointwise equal. We could then quote 5.3 to assure ourselves that they are the same function. ∎

Ordered pairs. By using the definition

$$(x, y) = \langle \mathrm{const}(x), \mathrm{const}(y)\rangle(\bot_{\mathbf{C}}),$$

which invokes 6.2 on any convenient fixed domain $\mathbf{C}$, it is easy to prove that $|\mathbf{A} \times \mathbf{B}|$ is in a one-one correspondence with the set-theoretical product of $|\mathbf{A}|$ and $|\mathbf{B}|$. Indeed, it can be shown that for $x \in |\mathbf{A}|$ and $y \in |\mathbf{B}|$,

(1) $(x, y) = \{(X, \Delta_{\mathbf{B}}) \mid X \in x\} \cup \{(\Delta_{\mathbf{A}}, Y) \mid Y \in y\} \in |\mathbf{A} \times \mathbf{B}|$;

(2) $\mathrm{fst}(x, y) = x$;

(3) $\mathrm{snd}(x, y) = y$;

and, for all $z \in |\mathbf{A} \times \mathbf{B}|$,

(4) $z = (\mathrm{fst}\, z, \mathrm{snd}\, z)$.

Also, using the notation of 6.2, we can say that

(5) $\langle f, g\rangle(t) = (f(t), g(t))$,

for all $t \in |\mathbf{C}|$.

There are also remarks that could be made about the pointwise nature of the partial ordering of $|\mathbf{A} \times \mathbf{B}|$, but we will not formulate them here. We do remark, however, that there is also a trivial product of *no* terms, $\mathbf{1}$, called the *unit* type or domain. It is such that $\mathcal{D}_{\mathbf{1}} = \{\Delta_{\mathbf{1}}\}$, and that equation determines it up to isomorphism. The domain $\mathbf{1}$ has but one element, namely $\perp_{\mathbf{1}}$. Note also that all approximable mappings $f : \mathbf{1} \to \mathbf{A}$ are *constant*, which shows how Definition 5.1 is a generalization of Definition 3.1. Note finally that there is but one approximable mapping $f : \mathbf{A} \to \mathbf{1}$, namely $f = 0 = \mathrm{const}(\perp_{\mathbf{1}})$. ∎

We turn now to the definition and properties of sums of domains.

DEFINITION 6.3. Let $\mathbf{A}$ and $\mathbf{B}$ be two information systems. By $\mathbf{A} + \mathbf{B}$, the *separated sum system*, we understand the system where, after choosing some convenient object Δ belonging neither to $\mathcal{D}_{\mathbf{A}}$ nor to $\mathcal{D}_{\mathbf{B}}$, we have:

(i) $\mathcal{D}_{\mathbf{A}+\mathbf{B}} = \{(X, \Delta) \mid X \in \mathcal{D}_{\mathbf{A}}\} \cup \{(\Delta, Y) \mid Y \in \mathcal{D}_{\mathbf{B}}\} \cup \{(\Delta, \Delta)\}$;

(ii) $\Delta_{\mathbf{A}+\mathbf{B}} = (\Delta, \Delta)$;

(iii) $u \in \mathrm{Con}_{\mathbf{A}+\mathbf{B}}$ iff either $\mathrm{lft}\, u \in \mathrm{Con}_{\mathbf{A}}$ and $\mathrm{rht}\, u = \emptyset$ or $\mathrm{lft}\, u = \emptyset$ and $\mathrm{rht}\, u \in \mathrm{Con}_{\mathbf{B}}$;

(iv′) $u \vdash_{\mathbf{A}+\mathbf{B}} (X', \Delta)$ iff $\mathrm{lft}\, u \neq \emptyset$ and $\mathrm{lft}\, u \vdash_{\mathbf{A}} X'$;

(iv″) $u \vdash_{\mathbf{A}+\mathbf{B}} (\Delta, Y')$ iff $\mathrm{rht}\, u \neq \emptyset$ and $\mathrm{rht}\, u \vdash_{\mathbf{B}} Y'$; and

(iv‴) $u \vdash_{\mathbf{A}+\mathbf{B}} (\Delta, \Delta)$ always holds.

where, in (iii), u is any finite subset of $\mathcal{D}_{\mathbf{A}+\mathbf{B}}$, in (iv′)–(iv‴), $u \in \mathrm{Con}_{\mathbf{A}+\mathbf{B}}$, and we let:

$$\begin{aligned} \mathrm{lft}\, u &= \{X \in \mathcal{D}_{\mathbf{A}} \mid (X, \Delta) \in u\}, \text{ and} \\ \mathrm{rht}\, u &= \{Y \in \mathcal{D}_{\mathbf{B}} \mid (\Delta, Y) \in u\}. \quad ∎ \end{aligned}$$

The plan of the sum definition is very similar to that for product, except that (1) for reasons to be made clear in examples, the parts *do not* share the least informative element (*i.e.* the data objects $(\Delta_{\mathbf{A}}, \Delta)$, $(\Delta, \Delta_{\mathbf{B}})$, and (Δ, Δ) are inequivalent in this system), and (2) instead of defining consistency and entailment in a conjunctive way, these notions are defined disjunctively. The effect of these changes over Definition 6.1 is to produce a system $\mathbf{A} + \mathbf{B}$ whose elements divide into disjoint copies of those of $\mathbf{A}$ and $\mathbf{B}$ (*plus* an extra element $\perp_{\mathbf{A}+\mathbf{B}}$). These remarks can be made more precise in the following way:

PROPOSITION 6.4. *If* $\mathbf{A}$ *and* $\mathbf{B}$ *are information systems, then so is* $\mathbf{A} + \mathbf{B}$, *and we have approximable mappings*

$$\mathrm{inl} : \mathbf{A} \to \mathbf{A} + \mathbf{B} \quad \textit{and} \quad \mathrm{inr} : \mathbf{B} \to \mathbf{A} + \mathbf{B},$$

such that, for approximable mappings

$$f : \mathbf{A} \to \mathbf{C} \quad \textit{and} \quad g : \mathbf{B} \to \mathbf{C},$$

there is one and only one *approximable mapping* $[f, g] : \mathbf{A} + \mathbf{B} \to \mathbf{C}$, *such that*

$$[f, g] \circ \mathrm{inl} = f\,, \quad [f, g] \circ \mathrm{inr} = g\,, \textit{ and } \quad [f, g](\perp_{\mathbf{A}+\mathbf{B}}) = \perp_{\mathbf{C}}.$$

Proof. The proof that 6.3 defines a system satisfying the basic axioms of 2.1 has to be left to the reader. Next using the notation of 6.1, where lft and rht were applied as operations on consistent sets $u \in \mathrm{Con}_{\mathbf{A}+\mathbf{B}}$, we redefine matters to have approximable mappings, where, for $v \in \mathrm{Con}_{\mathbf{A}}$ and $w \in \mathrm{Con}_{\mathbf{B}}$,

(1) $\quad v \mathbin{\mathrm{inl}} u \;$ iff $\; \{(X,\Delta) \mid X \in v\} \vdash_{\mathbf{A}+\mathbf{B}} u;$

(2) $\quad w \mathbin{\mathrm{inr}} u \;$ iff $\; \{(\Delta,Y) \mid Y \in w\} \vdash_{\mathbf{A}+\mathbf{B}} u;$ and

(3) $\quad u\,[f,g]\,s \;$ iff either $\vdash_{\mathbf{C}} s$, or $\mathrm{lft}\, u \neq \emptyset$ and $\mathrm{lft}\, u\; f\; s$,
or $\mathrm{rht}\, u \neq \emptyset$ and $\mathrm{rht}\, u\; g\; s$.

Because we defined $\mathbf{A}+\mathbf{B}$ so that consistency and entailment worked on the two halves of the set of data objects just as they worked on $\mathbf{A}$ and $\mathbf{B}$, respectively, it is easy to check that (1)–(3) define approximable mappings, and that the desired properties hold.

The uniqueness of $[f,g]$ comes from the fact that the elements of $\mathbf{A}+\mathbf{B}$, apart from the bottom element of the domain, are just the elements in the ranges of inl and inr. Since the function $[f,g]$ takes bottom to bottom (in the indicated domains), it will be uniquely determined by what it does on the two halves of the sum. The last equations of the theorem just say that the function *is* completely determined on these elements. ∎

It can also be shown that Propositions 6.2 and 6.4 uniquely characterize the domains $\mathbf{A}\times\mathbf{B}$ and $\mathbf{A}+\mathbf{B}$ up to isomorphism, and they give us the existence of additional mappings that are needed to show that product and sum are *functors* on the category of domains. We can also show from these results that the domain

$$\mathbf{BOOL} = \mathbf{1}+\mathbf{1}$$

has two elements true and false, such that any mapping on **BOOL** is uniquely determined by its action on true, false and $\perp_{\mathbf{BOOL}}$, and the values on the first two elements may be arbitrarily chosen.

7. The function space as a domain. Functions or mappings between domains are of basic importance for our theory, since it is through them that we most easily transform data and relate the structures into which the elements defined by the data objects enter. There are many possible functions, and large groups of them can be treated in a uniform manner. For instance, if the source and target domains match properly, any pair of functions can be composed—composition is an operation on functions of general significance. Now, if in the theory we could combine functions into domains themselves, then an operation like composition might become a mapping *of* the theory. Indeed, this is exactly what happens: suitably construed, composition is an approximable mapping of two arguments. Of course, for each configuration of linked source and target domains, there is a separate composition operation.

In order to make approximable mappings elements of a suitable domain, we have to discover first what their appropriate data objects are. In Section 5 this was hinted at already. To determine an approximable mapping $f : \mathbf{A} \to \mathbf{B}$, we have to say which pairs (u,v) with $u \in \mathcal{D}_{\mathbf{A}}$ and $v \in \mathcal{D}_{\mathbf{B}}$ stand in the mapping relation $u\; f\; v$. One such pair gives a certain (finite) amount of information about the possible functions that contain it, and an approximable mapping is completely determined by such pairs. Therefore, if there are appropriate notions of consistency and entailment *for these pairs*, we will be able to form a domain having functions as elements. Let us try out a formal definition first, and then look to an explanation of how it works.

DEFINITION 7.1. Let **A** and **B** be two information systems. By $\mathbf{A} \to \mathbf{B}$, the *function space*, we understand the system where:

(i) $\mathcal{D}_{\mathbf{A}\to\mathbf{B}} = \{(u, v) \mid u \in \mathrm{Con}_{\mathbf{A}} \text{ and } v \in \mathrm{Con}_{\mathbf{B}}\}$;

(ii) $\Delta_{\mathbf{A}\to\mathbf{B}} = (\emptyset, \emptyset)$; and where,

for all n and all $w = \{(u_0, v_0), \ldots, (u_{n-1}, v_{n-1})\}$, we have:

(iii) $w \in \mathrm{Con}_{\mathbf{A}\to\mathbf{B}}$ iff whenever $I \subseteq \{0, \ldots, n-1\}$ and $\bigcup\{u_i \mid i \in I\} \in \mathrm{Con}_{\mathbf{A}}$, then $\bigcup\{v_i \mid i \in I\} \in \mathrm{Con}_{\mathbf{B}}$; and

(iv) $w \vdash_{\mathbf{A}\to\mathbf{B}} (u', v')$ iff $\bigcup\{v_i \mid u' \vdash_{\mathbf{A}} u_i\} \vdash_{\mathbf{B}} v'$,

for all $u' \in \mathrm{Con}_{\mathbf{A}}$ and $v' \in \mathrm{Con}_{\mathbf{B}}$. ∎

We have already explained the choice of data objects in (i) above, and the least informative pair in (ii) is clearly right. Remember that as a data object (u, v) should be read as meaning that if the information in u is supplied as input, then at least v will be obtained as output. It is pretty obvious that one such data object by itself is consistent (they make constant functions, don't they?), but a *set* of several of these pairs may not be consistent. Hence, the need for part (iii) of the definition. It can be read informally as follows: Look for a selection I of the indices used in setting up w where the u_i for $i \in I$ are jointly consistent. Since the pairs in w are meant as correct information about a *single* function, then the combined input from all these selected u_i must be allowable. The function will then be required to give as output at least all the v_i for $i \in I$, owing to the fact that we are given that w is true of the function we have in mind. As a consequence, the set $\bigcup\{v_i \mid i \in I\}$ has got to be consistent, because it comes as output from consistent input for a single approximable function. What we are arguing for is the *necessity* of (iii)—the word "consistency" should mean that the data objects in the set are all true of at least one function.

Finally we have to argue that (iv) must give the right notion of entailment for these data objects. This can be seen by noting that for a fixed consistent w the set of pairs (u', v') satisfying the right-hand side of (iv) *defines* an approximable function. In checking this we have to remark that, for each $u' \in \mathrm{Con}_{\mathbf{A}}$, the set $\bigcup\{v_i \mid u' \vdash_{\mathbf{A}} u_i\}$ is consistent, so the definition makes sense. The transitivity properties needed for proving that we have an approximable mapping are easy to establish. This shows in particular that w is true of at least one approximable function, since the separate pairs (u_i, v_i) all satisfy the definition. But it is also simple to argue that for *any* approximable function, if w is true of it, then so is any pair (u', v') satisfying the definition of (iv). Consequently, what we find in (iv) is the definition of the *least* approximable function generated by w. The argument we have just outlined thus shows that the relationship $w \vdash_{\mathbf{A}\to\mathbf{B}} (u', v')$ means exactly that whenever w is true of an approximable mapping then so is (u', v'). It follows at once that $\vdash_{\mathbf{A}\to\mathbf{B}}$ is an entailment relation, and that the elements of $\mathbf{A} \to \mathbf{B}$ are just the approximable mappings, as we indicate in the next theorem.

THEOREM 7.2. *If* $\mathbf{A}$, $\mathbf{B}$, *and* $\mathbf{C}$ *are information systems, then so is* $\mathbf{A} \to \mathbf{B}$, *and the approximable mappings* $f : \mathbf{A} \to \mathbf{B}$ *are exactly the elements* $f \in |\mathbf{A} \to \mathbf{B}|$. *Moreover we have an approximable mapping* apply : $(\mathbf{B} \to \mathbf{C}) \times \mathbf{B} \to \mathbf{C}$ *such that whenever* $g : \mathbf{B} \to \mathbf{C}$ *and* $y \in |\mathbf{B}|$, *then*

$$\text{apply}(g, y) = g(y).$$

Furthermore, for all approximable mappings $h : \mathbf{A} \times \mathbf{B} \to \mathbf{C}$, *there is* **one and only one** *approximable mapping* curry $h : \mathbf{A} \to (\mathbf{B} \to \mathbf{C})$ *such that*

$$h = \text{apply} \circ \langle (\text{curry } h) \circ \text{fst}, \text{snd} \rangle.$$

Proof. We have already remarked on the essentials of the proof above. Definition 7.1 was devised to characterize exactly in $\text{Con}_{\mathbf{A} \to \mathbf{B}}$ the finite subsets of approximable functions, which, as binary relations, are being regarded as sets of ordered pairs. If $f : \mathbf{A} \to \mathbf{B}$ and if $w \subseteq f$, then from the properties of approximable functions, it can be checked directly that w satisfies the right-hand side of 7.1(iii). Conversely, if $w \in \text{Con}_{\mathbf{A} \to \mathbf{B}}$, then, as we have said, the relation which is defined by 7.1(iv) and may be notated by:

$$\overline{w} = \{(u', v') \mid w \vdash_{\mathbf{A} \to \mathbf{B}} (u', v')\},$$

is an approximable mapping, as can be proved using the right-hand side of 7.1(iv) and the usual properties of $\vdash_{\mathbf{A}}$ and $\vdash_{\mathbf{B}}$. Since $w \subseteq \overline{w}$, we see that $w \vdash_{\mathbf{A} \to \mathbf{B}} w'$ if, and only if, for all approximable $f : \mathbf{A} \to \mathbf{B}$, $w \subseteq f$ implies $w' \subseteq f$. (This is also the same as $w' \subseteq \overline{w}$, of course.) From these considerations it follows that not only is $\mathbf{A} \to \mathbf{B}$ an information system, but all approximable mappings are elements. Finally, if $f \in |\mathbf{A} \to \mathbf{B}|$, then—as a binary relation —it must be an approximable mapping, because the properties of Definition 5.1 are built into 7.1.

The construction of the special mapping apply as an approximable mapping also uses the idea of 7.1(iv). The consistent sets of the compound space $(\mathbf{B} \to \mathbf{C}) \times \mathbf{B}$ are essentially pairs of consistent sets, say $w \in \text{Con}_{\mathbf{B} \to \mathbf{C}}$ and $u' \in \mathbf{B}$. Now the relation we want from such pairs to consistent sets $v' \in \text{Con}_{\mathbf{B}}$ is just nothing more or less than $w \vdash_{\mathbf{B} \to \mathbf{C}} (u', v')$. Our discussion in the previous paragraph hints at why apply does in fact reproduce functional application when we evaluate apply(g, y).

The definition of curry h uses the same trick of regarding a binary relation with one term in a relationship being a pair as corresponding to another relation with one coordinate of the pair shifted to the other side. Specifically, we can think of an approximable mapping $h : \mathbf{A} \times \mathbf{B} \to \mathbf{C}$ as a relation from pairs (u, v) of consistent sets for $\mathbf{A}$ and $\mathbf{B}$, respectively, over to consistent sets w for $\mathbf{C}$. What we want for curry h is the relationship that goes from u to the pair (v, w). Of course (v, w) is just *one* data object for $\mathbf{B} \to \mathbf{C}$, but the input/output passage from the consistent sets of $\mathbf{A}$ to these objects is sufficient to determine curry h as an approximable mapping. The exact connection between the two mappings is given in terms of function values as follows:

$$h(x, y) = (\text{curry } h)(x)(y),$$

for all $x \in |\mathbf{A}|$ and $y \in |\mathbf{B}|$. From this equation it follows that curry h is uniquely determined. But, from what we know about apply, this is actually the same equation as that stated at the end of the theorem. ∎

Approximations to functions. Why have approximable functions been given this name? In general, elements of domains are the limits of their finite approximations. We have just indicated why the approximable mappings from one domain into another do form the elements of a domain themselves. We have explicitly shown how to construct the *finite* approximable mappings $\overline{w}$. A closer examination of the definitions would emphasize the very constructive nature of this analysis. It follows that the approximable mappings can therefore be approximated by simple functions. It *does not* follow that all approximable mappings are simple or constructive, since what takes place in the limiting process can be very complex. But the result does show how we can start to make distinctions once a precise sense of approximation is uncovered. ∎

Higher-type functions and the combinators. In the above discussion we have already combined the function-space construction with other domains by means of products. But there is nothing now stopping us from iterating the arrow domain constructor with itself as much as we like. This is how the so-called *higher types* are formed. In certain categories, such as the category of sets, this is a non-constructive move leading to the higher cardinal numbers. In the category of domains, however, the construct *is* constructive, because we have shown how to define all the parts of $\mathbf{A} \to \mathbf{B}$ in terms of very finite data objects (assuming, it need hardly be added, that $\mathbf{A}$ and $\mathbf{B}$ are constructively given).

Once the higher types have been formed as spaces, it must be asked what we are to do with them. The answer is that there are many, many mappings between these spaces that can be defined in terms of the simple notions we have been working with. These mappings are useful for the following reason: the higher types provide remarkabe scope for modelling notions (as those needed in denotatonal semantics for example), but the various aspects of the models have to be related—and this is where these mappings come into play. We have already seen a preliminary example in the last theorem, which can be interpreted as saying why the two domains shown are *isomorphic*:

$$\mathbf{A} \times \mathbf{B} \to \mathbf{C} \cong \mathbf{A} \to (\mathbf{B} \to \mathbf{C}).$$

We have neither the time nor the space to present a full theory of higher-type operators here, so some further examples will have to suffice. First, we have already made use of constant mappings. Since the construction of them is very uniform, there ought to be an associated operator. In fact, we have already been using it notationally. We have the approximable mapping $\mathrm{const} : \mathbf{B} \to (\mathbf{A} \to \mathbf{B})$ that takes every element of $\mathbf{B}$ to the corresponding constant function. (It has to be checked that this is an approximable mapping.) Note that there is a different mapping for each pair of domains $\mathbf{A}$ and $\mathbf{B}$, because the resulting types of const are different.

As another example, take the pairing of functions explained in Proposition 6.2. We can think of the operator in this case being

$$\mathrm{pair} : (\mathbf{C} \to \mathbf{A}) \times (\mathbf{C} \to \mathbf{B}) \to (\mathbf{C} \to (\mathbf{A} \times \mathbf{B})),$$

where for functions of the proper type we have:

$$\mathrm{pair}(f, g) = \langle f, g \rangle.$$

There will be a similar operator for the construct of Proposition 6.4.

Of course the most basic operator of function composition is also approximable of the appropriate type. We can write:

$$\text{comp} : (\mathbf{B} \to \mathbf{C}) \times (\mathbf{A} \to \mathbf{B}) \to (\mathbf{A} \to \mathbf{C}),$$

where for functions of the right types we have:

$$\text{comp}(g, f) = g \circ f.$$

The approximability has to be checked, of course. But once a number of the more primitive operators have been established as being approximable, then others can be proved to be so by writing them as combinations of previously obtained operators. ∎

Categories again. All of what we have been saying about operators ties in with category theory very nicely—as the category theorists have known for a long time. The technical term for what we have been doing in part is *cartesian closed category*—that is a property of the category of domains. Without going into details, that is essentially what 6.2 and 7.2 show of our category. But domains have many other properties beyond being a cartesian closed category. For example the possibility of forming sums is an extra (and useful) bonus, and there are many others. Nevertheless, the categorical viewpoint is a good way of organizing the properties, and it suggests other things to look for from our experiences with other categories. The next result gives a particularly important notion that can be expressed as an operator. ∎

THEOREM 7.3. *Let* $\mathbf{A}$ *be an information system. Then there is a unique operator,* **the least fixed-point operator**, *such that*

(i) $\text{fix} : (\mathbf{A} \to \mathbf{A}) \to \mathbf{A}$; *and,*

for all approximable mappings $f : \mathbf{A} \to \mathbf{A}$, *we have:*

(ii) $f(\text{fix}(f)) \subseteq \text{fix}(f)$; *and*

(iii) *for all* $x \in |\mathbf{A}|$, *if* $f(x) \subseteq x$, *then* $\text{fix}(f) \subseteq x$.

Moreover, for this operator, condition (ii) *is an equality.*

Proof. This is a well-known result—especially the fact that the conditions above uniquely determine the operator. The only question is the existence of the operator. The inclusion of condition (ii) gives the hint, for $\text{fix}(f)$ is the least solution of $f(x) \subseteq x$. Suppose x is any such element, then if $u \subseteq x$ and $u \; f \; v$ hold, it follows that $v \subseteq x$. Now, since $\emptyset \subseteq x$ always holds, if we wish to form the least x, we start with $\emptyset$ and just follow it under the action of f.

Specifically, we define $\text{fix}(f)$ to be the union of all $v \in \text{Con}_{\mathbf{A}}$ for which there exist a sequence $u_0, \ldots, u_n \in \text{Con}_{\mathbf{A}}$ where:

(1) $u_0 = \emptyset$;

(2) $u_i \; f \; u_{i+1}$ for all $i < n$; and

(3) $u_n = v$.

Because f is approximable, it is clear that $\mathrm{fix}\, f$ is closed under entailment. To prove that it is consistent, suppose both v and v' belong to the sets thrown into the union. We have to show that $v \cup v'$ is consistent and also is thrown in. Consider the two sequences $u_0, \ldots, u_n \in \mathrm{Con}_{\mathbf{A}}$ and $u_0', \ldots, u_n' \in \mathrm{Con}_{\mathbf{A}}$ that are responsible for putting v and v' in. It is without loss of generality that we assume they are of the same length, since we can always add lots of $\emptyset$'s onto the front of the shorter one and still satisfy (1)–(3). Now one just argues by induction on i that the sequence of unions $u_i \cup u_i'$ satisfies (1)–(3) with respect to $v \cup v'$.

But why is fix approximable? The method of proof is to replace f by $\overline{w}$ in (2) above, and to use the condition that there exists a sequence satisfying (1)–(3) as *defining* a relation between sets $w \in \mathrm{Con}_{\mathbf{A}\to\mathbf{A}}$ and sets $v \in \mathrm{Con}_{\mathbf{A}}$. It is not difficult to prove that this is an approximable mapping in the sense of the official definition. Clearly this relation determines fix as an operator. ∎

The result above not only proves that every approximable mapping of the form $f : \mathbf{A} \to \mathbf{A}$ has a fixed point as an element of $\mathbf{A}$, but that the association of the *least* fixed point is itself an approximable operator. The formulation makes essential use of the partial ordering of the domains, but Gordon Plotkin noticed as an exercise that the characterization of the operator can be given entirely by equations.

PROPOSITION 7.4. *The least fixed-point operator is uniquely determined by the following three conditions*:

(i) $\mathrm{fix}_{\mathbf{A}} : (\mathbf{A} \to \mathbf{A}) \to \mathbf{A}$, *for all systems* $\mathbf{A}$;

(ii) $\mathrm{fix}_{\mathbf{A}}(f) = f(\mathrm{fix}_{\mathbf{A}}(f))$, *for all* $f : \mathbf{A} \to \mathbf{A}$; *and*

(iii) $h(\mathrm{fix}_{\mathbf{A}}(f)) = \mathrm{fix}_{\mathbf{B}}(g)$, *whenever* $f : \mathbf{A} \to \mathbf{A}$, $g : \mathbf{B} \to \mathbf{B}$, $h : \mathbf{A} \to \mathbf{B}$, *provided that* $h \circ f = g \circ h$ *and* $h(\bot_{\mathbf{A}}) = \bot_{\mathbf{B}}$. ∎

Remarks on the space of strict mappings. In 7.4 and many other places we have had occasion to make use of mappings that take the bottom element of one domain over to the bottom element of the other domain. Such mappings are called *strict* mappings because they take a strict view of having empty input. As notation we might write

$$f : \mathbf{A} \to_{\mathbf{s}} \mathbf{B}$$

to mean that f is a *strict* approximable mapping (*i.e.* $f(\bot_{\mathbf{A}}) = \bot_{\mathbf{B}}$). The totality of domains and strict mappings forms an interesting category in itself, but it is best used in connection with the full category of all approximable mappings.

The collection of strict mappings forms a domain, too. The way to see this is to refer back to Definition 7.1 and add an additional clause ruling out non-strict mappings as inconsistent. What has to be added to 7.1(iii) is the conjunct on the right-hand side to the effect that if the condition $\emptyset \vdash_{\mathbf{A}} \bigcup\{u_i \mid i \in I\}$ holds, then $\emptyset \vdash_{\mathbf{B}} \bigcup\{v_i \mid i \in I\}$ holds too. By the same arguments we used before, it follows that this is the appropriate system for the domain of strict mappings. We can denote it by $(\mathbf{A} \to_{\mathbf{s}} \mathbf{B})$

There is also a useful operator

$$\mathrm{strict} : (\mathbf{A} \to \mathbf{B}) \to (\mathbf{A} \to_{\mathbf{s}} \mathbf{B})$$

defined by the condition that for $f : \mathbf{A} \to \mathbf{B}$ we have:

$$u \ \mathrm{strict}(f)\ v \quad \text{iff} \quad \text{either } \emptyset \vdash_{\mathbf{B}} v \text{ or } \emptyset \not\vdash_{\mathbf{A}} u \text{ and } u\ f\ v,$$

for all $u \in \mathrm{Con}_{\mathbf{A}}$ and $v \in \mathrm{Con}_{\mathbf{B}}$. This operator converts every approximable mapping into the largest strict mapping contained within it.

Since every strict mapping *is* an approximable mapping, there is also an obvious operator going the other way. The pair of operators shows that $\mathbf{A} \to_s \mathbf{B}$ as a domain is what is called a *retract* of $\mathbf{A} \to \mathbf{B}$. There is an interesting theory of this kind of relationship between domains, but we cannot enter into it here.

As a very small application of the use of strict mappings, we remark that the following two domains are isomorphic:

$$\mathbf{A} \times \mathbf{A} \cong (\mathbf{BOOL} \to_s \mathbf{A}).$$

The mapping from right to left is called the *conditional* operator, cond, and we have for all elements $x, y \in |\mathbf{A}|$ and $t \in |\mathbf{BOOL}|$

$$\mathrm{cond}(x,y)(t) = \begin{cases} x, & \text{if } t = \text{true}, \\ y, & \text{if } t = \text{false}. \end{cases} \quad \blacksquare$$

8. Some domain equations. Having outlined the theory of several domain constructs, the final topic for this paper will be the discussion of the *iteration* of these constructs in giving *recursive*, rather than direct definitions of domains. These recursively defined systems have often been called "reflexive," because the domains generally contain copies of themselves as a part of their very structure. The way that this self-containment takes place is best expressed by the so-called *domain equations*, which are really isomorphisms that relate the domain as a whole to a combination of domains—usually with the main domain as a component. This description is rough, since recursion equations for domains can be as complex as recursion equations for functions. We will not enter into a full theory of domain equations now but will just review some preliminary examples to illustrate how the new presentation makes the constructions more explicit.

A domain of trees or S-expressions. This is everyone's favorite example. And a very nice example it is, but we should not think that it contains all the meat of the theory of domain equations. Even if we generalize the kinds of equations to contain all iterations of the domain constructs $+$ and $\times$, the full power of the method has not been exploited. We will try to make this clear in the further examples.

Let a domain (information system) $\mathbf{A}$ be given. What we want to construct is a domain $\mathbf{T}$ of "trees" built up from elements of $\mathbf{A}$ as "atoms". For simplicity we consider unlabelled binary trees here, but more complex trees are easy to accommodate. The domain equation we want to "solve" is this one:

$$\mathbf{T} \cong \mathbf{A} + (\mathbf{T} \times \mathbf{T}).$$

If such a domain exists, then we can say that (up to isomorphism) the elements of the domain $\mathbf{T}$ are either bottom, or elements of the given domain $\mathbf{A}$, or *pairs of elements* from the domain $\mathbf{T}$ itself. And these are the only kinds of elements that $\mathbf{T}$ has.

To prove that such a domain exists it is only necessary to ask what information has to be given about a prospective element. The answer may involve us in a regress, but the running backwards need not be infinite—at least for the finite elements. As we shall see, the infinite elements of $\mathbf{T}$ can be self-replicating; but, to define a domain fully, all we have to do is to build up the finite elements out of the data objects in a systematic way. Fortunately, in order to satisfy the above equation, the required closure conditions on data objects are simple to achieve.

In the first place, we need copies of all the data objects of $\mathbf{A}$ to put into the sum. The easy way to do this is to take an object Δ not in $\mathcal{D}_{\mathbf{A}}$ and to let, by definition,

$$\Delta_{\mathbf{T}} = (\Delta, \Delta).$$

That gives us one member of $\mathcal{D}_{\mathbf{T}}$, the one we always have to have in any case. The copy of an $X \in \mathcal{D}_{\mathbf{A}}$ is just going to be (X, Δ). The other members of $\mathcal{D}_{\mathbf{T}}$ will be of the form (Δ, U), where U gives us information about the other kind of elements of $\mathbf{T}$. The point is that $\mathbf{T}$ has to be a sum, and we are using just the scheme of Definition 6.3 to set this up.

Next we have to think what kind of information the U above should contain. Because we want a product, we refer back to Definition 6.1 and imagine we have already defined $\mathcal{D}_{\mathbf{T}}$. What 6.1(i) suggests is that we throw in a bunch of other data objects into $\mathcal{D}_{\mathbf{T}}$. The only point that needs care is that the data objects for the product must be *copied* into the overall sum. With this in mind, the following clauses give us the inductive definition of $\mathcal{D}_{\mathbf{T}}$:

(1) $\Delta_{\mathbf{T}} \in \mathcal{D}_{\mathbf{T}}$;

(2) $(X, \Delta) \in \mathcal{D}_{\mathbf{T}}$ whenever $X \in \mathcal{D}_{\mathbf{A}}$; and

(3) $(\Delta, (Y, \Delta_{\mathbf{T}})) \in \mathcal{D}_{\mathbf{T}}$, and $(\Delta, (\Delta_{\mathbf{T}}, Z)) \in \mathcal{D}_{\mathbf{T}}$ whenever $Y, Z \in \mathcal{D}_{\mathbf{T}}$.

Of course,when we say "inductive definition," we mean that $\mathcal{D}_{\mathbf{T}}$ is the *least* class satisfying (1)–(3). By standard arguments it can be shown that $\mathcal{D}_{\mathbf{T}}$ satisfies this set-theoretical equation:

$$\mathcal{D}_{\mathbf{T}} = \{\Delta_{\mathbf{T}}\} \cup \{(X, \Delta) \mid X \in \mathcal{D}_{\mathbf{A}}\} \cup \{(\Delta, (Y, \Delta_{\mathbf{T}})) \mid Y \in \mathcal{D}_{\mathbf{T}}\} \cup \{(\Delta, (\Delta_{\mathbf{T}}, Z)) \mid Z \in \mathcal{D}_{\mathbf{T}}\}.$$

In fact, with some very mild assumptions about ordered pairs in set theory, $\mathcal{D}_{\mathbf{T}}$ is the *only* solution to the above equation.

Defining the data objects is but part of the story: the same data objects can enter into quite different information systems. Data objects are just "tokens" and are only given "meaning" when $\mathrm{Con}_{\mathbf{T}}$ and $\vdash_{\mathbf{T}}$ are defined. Let us consider the problem of consistency first. We already understand the notion as it applies to sum and product systems, so we must merely copy over the parts of the previous definitions in the right position for the definition of $\mathrm{Con}_{\mathbf{T}}$. There are two forms we could give this definition; perhaps the best is the inductive one. We have:

(4) $\emptyset \in \mathrm{Con}_{\mathbf{T}}$;

(5) $u \cup \{\Delta_{\mathbf{T}}\} \in \mathrm{Con}_{\mathbf{T}}$ whenever $u \in \mathrm{Con}_{\mathbf{T}}$;

(6) $\{(X,\Delta) \mid X \in w\} \in \mathrm{Con}_{\mathbf{T}}$ whenever $w \in \mathrm{Con}_{\mathbf{A}}$;

(7) $\{(\Delta,(Y,\Delta_{\mathrm{T}})) \mid Y \in u\} \cup \{(\Delta,(\Delta_{\mathrm{T}},Z)) \mid Z \in v\} \in \mathrm{Con}_{\mathbf{T}}$
whenever $u, v \in \mathrm{Con}_{\mathbf{T}}$.

Conditions (4)–(7) certainly make the inductive character of $\mathrm{Con}_{\mathbf{T}}$ clear—again, let us emphasize, the set being specified is the least such. Also clear from the definition is the fact that a consistent set of **T**—aside from containing Δ_{T}—is either a copy of a consistent set of **A** or a copy of a consistent set of $\mathbf{T} \times \mathbf{T}$. We could thus state a set-theoretical equation for $\mathrm{Con}_{\mathbf{T}}$ similar to the one for $\mathcal{D}_{\mathbf{T}}$.

It remains to define entailment for T. Here are the inductive clauses which are pretty much forced on us by our objective of solving the domain equation:

(8) $u \vdash_{\mathbf{T}} \Delta_{\mathrm{T}}$ always;

(9) $u \cup \{\Delta_{\mathrm{T}}\} \vdash_{\mathbf{T}} Y$ whenever $u \vdash_{\mathbf{T}} Y$;

(10) $\{(X,\Delta) \mid X \in w\} \vdash_{\mathbf{T}} (W,\Delta)$ whenever $w \vdash_{\mathbf{A}} W$;

(11) $\{(\Delta,(Y,\Delta_{\mathrm{T}})) \mid Y \in u\} \cup \{(\Delta,(\Delta_{\mathrm{T}},Z))\} \vdash_{\mathbf{T}} (\Delta,(X,\Delta_{\mathrm{T}}))$
whenever $u \vdash_{\mathbf{T}} X$ and $v \in \mathrm{Con}_{\mathbf{T}}$; and

(12) $\{(\Delta,(Y,\Delta_{\mathrm{T}})) \mid Y \in u\} \cup \{(\Delta,(\Delta_{\mathrm{T}},Z))\} \vdash_{\mathbf{T}} (\Delta,(\Delta_{\mathrm{T}},X))$
whenever $u \vdash_{\mathbf{T}} X$ and $u \in \mathrm{Con}_{\mathbf{T}}$.

Inductive definitions engender inductive proofs. It now has to be checked that consistency and entailment for **T** satisfy the axioms of 2.1. The steps needed for this check are mechanical. (The proof may be aided by noting that the cases in (4)–(7) and in (8)–(12) are disjoint—except for a trivial overlap between (8) and (9). The cases get invoked typically by asking, when confronted with an entailment to prove, for the nature of the data object on the right of the turnstile.)

Having defined and verified that **T** is an information system, the validity of the domain equation for **T** is secured by forming the right-hand side and noting that **T** is *identical* to $\mathbf{A} + (\mathbf{T} \times \mathbf{T})$. The reason is that we carefully chose the notation to match the official definitions of sums and products. (In general, in solving domain equations some transformation might have to take place to "re-format" data objects if things are not set up to be literally the same.)

It should be remarked that the sense can be made precise in which **T** is the *least* solution of the given domain equation. (It is an initial algebra in a suitable category of algebras and algebra homomorphisms.) It is pretty obvious that it is minimal in some sense, because we put into it only what was strictly required by the problem and nothing more.

It is also fairly obvious that there are *many* solutions to this domain equation. A non-constructive way to obtain non-minimal solutions is to interpret the whole construction of **T** in a non-standard model of set theory. Though, in the definition of $\mathcal{D}_{\mathbf{T}}$, it looks like we are only working with very finite objects, everything we did could be made abstract and could be carried out in some funny universe. The result would be a system of "finite" data objects having all the right formal properties but containing things not in the standard minimal system. We would then take the notions of consistency and entailment that also exist in the funny universe and restrict them to sets of data objects that are actually finite in the standard sense. It can be seen from the formal properties of the construction that the resulting notions satisfy our axioms for an information system and that the domain equation holds—BUT the system would have many different elements beyond what we put into the original **T**. To make this construction work, by the way, we would have to force **A** to be absolute in the model: if it is actually finite (say, $\mathbf{A} = \mathbf{BOOL}$), then there is no problem. (Constructive methods for introducing "nonstandard" data objects can also be given.)

Finally, we must remark on why we called T a domain of *S-expressions.* The answer becomes clear when we structure **T** as an algebra. First, there is an approximable mapping

$$\text{atom} : \mathbf{A} \to \mathbf{T},$$

which injects **A** into **T** making the elements of **A** "atoms" of **T**. Then there is a truth-valued predicate on **T** which decides whether an element is an atom:

$$\text{isatom} : \mathbf{T} \to \mathbf{BOOL}.$$

Finally, since $\mathbf{T} \times \mathbf{T}$ is a part of **T**, we can redefine the paring functions so that:

$$\text{pair} : \mathbf{T} \times \mathbf{T} \to \mathbf{T},\ \ \text{fst} : \mathbf{T} \to \mathbf{T},\ \text{and}\ \ \text{snd} : \mathbf{T} \to \mathbf{T}.$$

In LISP terminology, these operations are the same as the familiar cons, car, and cdr. This makes **T** into an algebra where, starting from atoms, elements—expressions—can be built up by iterated pairing.

But why is our system different from the usual way of regarding S-expressions? The answer is that by including partial expressions (those involving $\perp_{\mathbf{T}}$) and by completing the domain with limits, *infinite* expressions are introduced. For instance, if $a \in |\mathbf{T}|$, then we can solve the fixed-point equation:

$$x = \text{pair}(\text{atom}(a), x),$$

which is an infinite list of a's. This is but one example; the possibilities have been discussed in many papers too numerous to mention here.

As is common to remark, S-expressions can also be thought of as trees: the parse tree that gives the grammatical form of the expression. What we have added to the idea of a tree is possibility of having infinite trees, and having all these trees as elements of a domain. ∎

A domain for λ-claculus. A lengthy discussion with many references on λ-calculus models can be found in Longo [1982]. All we wish to remark on here is how the method of construction by solving a domain equation can be fit into the new presentation. What I have added to the previous ideas (that in any case came out of an analysis of finite elements of models) is the general view of information systems. In particular the models obtained this way are *not* lattices—hence, the need for the calculations with Con. I hope that the presentation here makes it clearer how "pure" λ-calculus models can be related to other domains having other types of structures—for instance, those needed in denotational semantics.

The domain equation we wish to solve is:

$$\mathbf{D} \cong \mathbf{A} + (\mathbf{D} \to \mathbf{D}).$$

We proceed in much the same way we did for **T**, except we must now put in data objects appropriate to the function space. Here is construction, where again Δ is chosen outside $\mathcal{D}_{\mathbf{A}}$ and $\Delta_{\mathbf{D}} = (\Delta, \Delta)$:

(1) $\Delta_{\mathbf{D}} \in \mathcal{D}_{\mathbf{D}}$;

(2) $(X, \Delta) \in \mathcal{D}_{\mathbf{D}}$ whenever $X \in \mathcal{D}_{\mathbf{A}}$;

(3) $(\Delta, (u, v)) \in \mathcal{D}_{\mathbf{D}}$ whenever $u, v \in \mathrm{Con}_{\mathbf{D}}$;

(4) $\emptyset \in \mathrm{Con}_{\mathbf{D}}$;

(5) $u \cup \{\Delta_{\mathbf{D}}\} \in \mathrm{Con}_{\mathbf{D}}$ whenever $u \in \mathrm{Con}_{\mathbf{D}}$;

(6) $\{(X, \Delta) \mid X \in w\} \in \mathrm{Con}_{\mathbf{D}}$ whenever $w \in \mathrm{Con}_{\mathbf{A}}$; and

(7) $\{(\Delta, (u_0, v_0)), \ldots, (\Delta, (u_{n-1}, v_{n-1}))\} \in \mathrm{Con}_{\mathbf{D}}$ provided $u_i, v_i \in \mathrm{Con}_{\mathbf{D}}$ for all $i < n$ and whenever $I \subseteq \{0, \ldots, n-1\}$ and $\bigcup\{u_i \mid i \in I\} \in \mathrm{Con}_{\mathbf{D}}$, then $\bigcup\{v_i \mid i \in I\} \in \mathrm{Con}_{\mathbf{D}}$.

What is different here from the definition of **T** is the fact that the concepts $\mathcal{D}_{\mathbf{D}}$ and $\mathrm{Con}_{\mathbf{D}}$ are *mutually recursive* because the data objects are themselves built from consistent sets. The scheme is based on a combination of the sum construct and the function-space construct, but the mutual recursion allows "feedback" to occur.

To complete the definition we have to give the clauses for the inductive definition of entailment. They are:

(8) $u \vdash_{\mathbf{D}} \Delta_{\mathbf{D}}$ always;

(9) $u \cup \{\Delta_{\mathbf{D}}\} \vdash_{\mathbf{D}} Y$ whenever $u \vdash_{\mathbf{D}} Y$;

(10) $\{(X, \Delta) \mid X \in w\} \vdash_{\mathbf{D}} (W, \Delta)$ whenever $w \vdash_{\mathbf{A}} W$;

(11) $\{(\Delta, (u_0, v_0)), \ldots, (\Delta, (u_{n-1}, v_{n-1}))\} \vdash_{\mathbf{D}} (\Delta, (u', v'))$ whenever $\bigcup\{v_i \mid u' \vdash_{\mathbf{D}} u_i\} \vdash_{\mathbf{D}} v'$ and the set on the left is in $\mathrm{Con}_{\mathbf{D}}$.

Obviously these definitions are much shorter if we have a domain in which all sets are consistent, but there are many reasons for retaining the consistency concept throughout. The check that **D** is an information system and satisfies the domain equation is mechanical. We cannot detail here how this construction provides a λ-calculus model.

It is clear that these definitions are constructive, and that, with a suitable Gödel numbering of the data objects, the predicates for consistency and entailment are recursively enumerable. However, the recursion used builds up the predicates by going from less complicated data objects to more complicated ones; therefore, the predicates must be *recursive*, because, for a certain size data object, the derivation that puts it into the predicate is of a bounded length. This observation helps in the discussion of the computability of the operators defined on these domains—another topic we cannot discuss here. ∎

A universal domain. As a final example of building up domains recursively, we give a construction of a "universal" domain **U**. (The reason for the name will be explained presently.) The best way to define **U** seems to be to define a domain **V** with a top element first,and then to remove the top.

The recursion for **V** is remarkably simple. We begin with two distinct objects Δ and ∇ that give information about the top and bottom of **V**, respectively. Thus, $\Delta_{\mathbf{V}} = \Delta$ by definition. We assume that these two special data objects are "atomic" in the sense that they are not equal to any ordered pair of objects. For the definition of $\mathcal{D}_{\mathbf{V}}$ we have these clauses:

(1) $\Delta, \nabla \in \mathcal{D}_{\mathbf{V}}$;

(2) $(X, \Delta) \in \mathcal{D}_{\mathbf{V}}$ and $(\Delta, Y) \in \mathcal{D}_{\mathbf{V}}$ whenever $X, Y \in \mathcal{D}_{\mathbf{V}}$.

In other words, we begin with two objects and close up under two flavors of copies of these objects. (A product result is involved here, so that is the reason for structuring the flavors the way we have.)

For **V** all subsets of $\mathcal{D}_{\mathbf{V}}$ are consistent, so all we have left is to define entailment for this domain. The clauses are:

(3) $u \vdash_{\mathbf{V}} \Delta$ always;

(4) $u \vdash_{\mathbf{V}} \nabla$ whenever either $\nabla \in u$ or $\{X \mid (X, \Delta) \in u\} \vdash_{\mathbf{V}} \nabla$ and $\{Y \mid (\Delta, Y) \in u\} \vdash_{\mathbf{V}} \nabla$;

(5) $u \vdash_{\mathbf{V}} (X', \Delta)$ whenever either $\nabla \in u$ or $\{X \mid (X, \Delta) \in u\} \vdash_{\mathbf{V}} X'$; and

(6) $u \vdash_{\mathbf{V}} (\Delta, Y')$ whenever either $\nabla \in u$ or $\{Y \mid (\Delta, Y) \in u\} \vdash_{\mathbf{V}} Y'$.

The proof that **V** is an information system proceeds as before. Note that, under the above definition of entailment, the data objects $\Delta, (\Delta, \Delta), ((\Delta, \Delta), \Delta)$, *etc.* are all equivalent. There is, however, no other data object equivalent to ∇. The domain equation satisfied by **V** is:

$$\mathbf{V} \cong \mathbf{V} \times \mathbf{V}.$$

Of course, there are an unlimited number of solutions to this equation, so the fact that **V** satisfies it tells us very little.

Because ∇ entails everything, we can regard it as a "rogue" object that ought to be banned from polite company: the only element of **V** it gives any information about is the top element, which is as unhelpful as any element could be. We should simply throw it out as being "inconsistent." What remains is the domain **U**. Formally we have:

(7) $\mathcal{D}_{\mathbf{U}} = \mathcal{D}_{\mathbf{V}} - \{\nabla\}$;

(8) $\Delta_{\mathbf{U}} = \Delta_{\mathbf{V}}$;

(9) $\mathrm{Con}_{\mathbf{U}} = \{u \subseteq \mathcal{D}_{\mathbf{U}} \mid u \text{ finite and } u \not\vdash_{\mathbf{V}} \nabla\}$; and

(10) $u \vdash_{\mathbf{U}} Y$ iff $u \in \mathrm{Con}_{\mathbf{U}}$, $Y \in \mathcal{D}_{\mathbf{U}}$ and $u \vdash_{\mathbf{V}} Y$.

The same style of definition would work in any situation when an information system has a rogue data object that entails everything: there always is a system that results from eliminating all those objects that entail everything. Indeed, we could have always included such an object in any domain and altered the definition to take as elements those deductively closed sets of data objects that do not have the rogue object as a member. We did not do this for the reason that superfluous elements cause lots of exceptions in constructs such as product, where there is a temptation to let them enter into various combinations.

Now in **U** we *do* allow ∇ to enter into combinations—and this is part of the secret of the construction. The consequence is, however, that the domain equation which **U** satisfies is not too easy to state since it involves an unfamiliar functor. So it is not through such equations that we will understand its nature in a direct way. But it is possible to explain how it works by reference to the steps in the construction.

Imagine the full (infinite) binary tree. The data objects of **U** are giving information about possible paths in the tree. We think to the tree starting at the root node at the top of the page and growing down. The object Δ gives no information—so no paths are *excluded.* (If we would have allowed ∇, then the information it would have been giving is that *all paths are excluded.*) The data object (X, Δ) tells us about a path that *either* it is unrestricted on the right half of the tree, *or* on the left, when we start at the node directly below the root, the paths that are excluded from the subtree are those excluded according to X. This makes sense because the subtrees of the binary tree look exactly like the whole tree, so information can be relativized or translated to other positions. With (Δ, Y) the rôles of right and left are interchanged. We could have introduced data objects of the form (X, Y) which tell us information about both halves of the tree at the same time, but the consistent set $\{(X, \Delta), (\Delta, Y)\}$ does the same job. In general consistent sets should be thought of as *conjunctions*; while, in this example, the comma in the ordered pair should be thought of as a *disjunction* when "reading" information objects.

We can now see that a single data object (if it contains ∇) looks down the tree along a finite path to some depth and then *excludes* the rest of the tree below that node. A consistent set of data objects leaves at least one hole, so at least one path is not excluded. The maximal consistent sets of information objects are those giving true information about one single (infinite) path—the total elements of the domain **U** correspond exactly to the infinite paths in the binary tree. The partial elements are harder to describe geometrically, however. In accumulating information into a consistent set, holes can be left all over the tree. A partial object is therefore of an indeterminate character, since the "path" we are describing might sneak through any one of the holes. (There is, by the way, a precise topological explanation of what is happening. The total elements of **U** form a well-known topological space, the so-called Cantor space, and the partially ordered set of elements of **U** is isomorphic to the lattice of *open* sets of the space—save that the whole space is not allowed.)

This is all very well, but what, we ask, is the good of this domain, and why is it called "universal". The proof cannot be given here, but the result is as follows. As a consequence of standard facts about countable Boolean algebras, it can be proved that every "countably based" domain is a subdomain of **U**. More specifically, if **A** is an information system, and if $\mathcal{D}_{\mathbf{A}}$ is countable, then there exists a pair of approximable mappings

$$a : \mathbf{A} \rightarrow \mathbf{U} \text{ and } b : \mathbf{U} \rightarrow \mathbf{A},$$

such that

$$b \circ a = \mathrm{I}_{\mathrm{A}} \text{ and } a \circ b \subseteq \mathrm{I}_{\mathrm{U}}.$$

This makes **A** a special kind of retract of **U**. The mappings a and b are far from unique, but at least there is one way to give a one-one embedding of the elements of **A** into the elements of **U**.

The universal property of **U** can be applied quite widely. For example, since $(\mathbf{U} \to \mathbf{U})$ is a system with only countably many data objects (by explicit construction!), this system is a retract of **U**. Fixing on one such retraction pair as above, makes **U** also into a model of the λ-calculus. Whether different retractions give essentially different models I do not know. But the point of the remark is to show that domains can contain their own function spaces for a variety of interesting reasons. ∎

A domain of domains. Not many details can be presented here, but we would also like to remark that even domains can be made into a domain. One way of getting an idea of how this is possible is to note that since subdomains of **U** correspond to certain kinds of functions on **U**, and since the function space of **U** is also a subdomain of **U**, it might be suspected that the subdomains of **U** form a single subdomain of **U**.

That is a fairly sophisticated way of reaching the conclusion (and many details have to be worked out). A more elementary approach would be just to ask what it means to give a finite amount of information *about a domain.* For the sake of uniformity, suppose that the data objects of the possible domain are drawn from the non-negative integers, and that we conventionally use 0 for Δ. Then to give a finite amount of information about a domain is—roughly—to specify a finite part of Con and a finite part of $\vdash$. To make the formulation easier, we will reserve for 1 a rôle like the one recently played by ∇. What the specifications will boil down to is pairs (u, v) of finite sets of integers used as data objects to convey one piece of information about an entailment relation.

But hold, entailment relations are very closely connected to approximable mappings. Indeed, we remarked before that the identity function as an approximable mapping on a domain is just represented as the underlying entailment relation itself. Suppose we take as our domain the domain of all sets of integers. It is a powerset, so call it **P**. That is to say, the integers are the data objects, all finite sets are consistent, and the entailment relation is the minimal possible one. (As far as elements go, an arbitrary set of integers is equal to the union of all its finite subsets, which means that the elements of the domain are in a one-one correspondence with the arbitrary sets of integers.) The question is: which approximable mappings on **P** into itself correspond to entailment relations on the integers as data objects?

The answer can be expressed most succinctly using our standard notation. If we think of $r : \mathbf{P} \to \mathbf{P}$ as a relation between finite sets in the usual way, then to say that r is reflexive is to say:

(1) $\quad \mathrm{I}_{\mathbf{P}} \subseteq r.$

To say that r is transitive is to say:

(2) $\quad r \circ r = r.$

To say that for r the object 0 plays at being Δ is to say:

$$(3) \quad \bar{0} \subseteq r(\perp),$$

where in general $\bar{n}$ is short for $\overline{\{n\}}$ in the domain **P**. Then, to say that 1 plays at being a rogue object is to say:

$$(4) \quad \top = r(\bar{1}).$$

Finally, to say that 1 is an inconsistent object that has to be excluded is to say:

$$(5) \quad \bar{1} \nsubseteq r(\bar{0}).$$

That's it. The collection of approximable mappings satisfying (1)–(5) gives us all the entailment relations we need. Condition (5) is a consistency condition, and for r the consistent finite sets u are those such that $\bar{1} \nsubseteq r(\bar{u})$. What we are asserting is that the totality of r satisfying (1)–(5) forms the elements of a domain—one that has been derived from $(\mathbf{P} \to \mathbf{P})$ in a way similar to the way we derived **U** from **V** above.

Having made domains into a domain, the next step is to see how constructs on domain (*i.e.* functors) can be made into approximable mappings. But the retelling and development of that story will have to wait for another publication along with the very interesting chapter on powerdomains. I only hope the ground covered here makes the theory of domains seem more elementary and more natural. ▮