MASSACHUSETTS INSTITUTE OF TECHNOLOGY

PROJECT MAC

Computation Structures Group Memo 83

The Flow of Lattice Diagrams
(or What's an S-Expression Like You Doing in a Lattice Like This?)

by

Mike Van De Vanter

This work was submitted for credit in Subject 6.534,
"Semantic Theory for Computer Systems," Spring 1973.

June 1973

TABLE OF CONTENTS

1. INTRODUCTION 1
2. THE SET OF S-EXPRESSIONS 2
3. LATTICES 2
4. CONSTRUCTING THE LATTICE OF S-EXPRESSIONS 4
4.1. METHODS OF CONSTRUCTION 4
4.2. THE BASIC LATTICE 5
4.3. THE HIGHER LATTICES 6
4.4. THE FINAL LATTICE 6
5. COMPLETING THE LATTICE 9
5.1. THE PROJECTION MAPPINGS 9
5.2. THE CONSTRUCTION OF NEW ELEMENTS 10
6. PROPERTIES OF S_∞ 12
6.1. THE FUNCTION Cons 12
6.2. THE DECOMPOSITION OF S_∞ 12
6.3. THE FUNCTIONS Car, Cdr 13
6.4. CONTINUITY OF FUNCTIONS 13
7. CLASSES OF ELEMENTS OF S_∞ 14
7.1. NOTATION 14
7.2. THE PERFECT ELEMENTS 15
7.3. THE FINITE PERFECT ELEMENTS - FP 15
7.4. THE FINITE ELEMENTS - F 17
7.5. THE ALGEBRAIC PERFECT ELEMENTS - AP 18
7.6. THE ALGEBRAIC ELEMENTS - A 22
7.7. THE TRANSCENDENTAL ELEMENTS - TP and T 22
8. COMMENTS ON S_∞ 22
8.1. LOOSE ENDS - THE ELEMENT $(\perp . \perp)$ 23
8.2. LOOSE ENDS - CONSTRUCTING ELEMENTS OF A 23
9. THE JUSTIFICATION FOR LATTICES 24

THE FLOW OF LATTICE DIAGRAMS
(or, WHAT'S AN S-EXPRESSION LIKE YOU DOING IN A LATTICE LIKE THIS?)

1. INTRODUCTION

This paper is an expression of my reflections on some of the recent work done by Dana Scott. In particular I would like to consider his method of using lattices to impose a very fundamental structure on the abstract concept of a "data type". Intuitively it isn't at all clear what one can say about "data types" that doesn't depend on the particular kind of data under scrutiny. It seems even less clear what connection could possibly exist between this ill defined notion and the mathematical structure of lattices! Scott postulates that the set of elements which we would like to call a "data type" should be conceptually expanded, and in the process a structure should be defined on the set. It is this structure which turns out to be a lattice. In an attempt to illustrate how this procedure is carried out and what Scott feels to be the benefits earned, I will consider a familiar (and simple) example of a "data type", the set of symbolic S-expressions (a la LISP!).

2. THE SET OF S-EXPRESSIONS

Before we start to operate on this set, we really ought to stop and consider just what set we are really talking about. A typical definition of S-expressions is given in Dertouzos[1]:

An ATOM is a finite string of characters and/or numbers.

Recursive definition of S-EXPRESSIONS:

> 1). Every atom is an S-EXPRESSION.
> 2). If S and T are S-EXPRESSIONS,
> then (S . T) is also an S-EXPRESSION.
> 3). Only the objects defined by rule 1)
> and by a finite number of applications
> of rule 2) are S-EXPRESSIONS.

We also note the three primitive functions on S-expressions, Car, Cdr, and Cons, and their definitions:

> Car <(S . T)> = S for all S-EXPRS S and T, undefined otherwise.
> Cdr <(S . T)> = T for all S-EXPRS S and T, undefined otherwise.
> Cons <S,T> = (S . T) for all S-EXPRS S and T.

The set of S-expressions, then, could be informally described as the set of finite binary trees with atoms for leaves. We ask then, what kind of structure this set has, i.e. what relationships exists among its various elements? We naturally assume that the atoms themselves are discreet, and bear no interesting relationships to one another. A unary relatiion comes immediately to mind (the property of being an atom), but it seems that the only interesting binary relation might be that of inclusion (like MEMBER in LISP). I claim, however, that this relation is not really very interesting if we want to build a mathematical model for this set. The justification for this claim should become clear as we see how Scott's method works.

3. LATTICES

Scott describes in [3] his motivation for using a lattice structure to deal with his notion of abstract "data type". At this point I will not pause to consider his general defense of the idea, rather let us proceed to use his method for embedding the example set in a lattice. After some of the benefits from the construction become apparent, we can return to the point in a better light.

First, we should note that a "lattice" is defined to be a partially ordered set in which every two elements have both a "meet" and a "join"[2]. The concepts "meet" and "join" correspond well to our common definitions of "greatest lower bound" and "least upper bound" respectively, and are denoted by Scott as $\sqcap$ and $\sqcup$. A "complete" lattice is defined to be one in which every subset of the elements has both a meet and a join (this is the type of lattice we will want). The partial ordering we want to use on our lattice is denoted by Scott as $\sqsubseteq$ and corresponds roughly to the idea of "approximation". He points out in [3] and [4] that we should not confuse this notion with the concept of "distance from". We say that $x \sqsubseteq y$ will mean that y is like x (i.e. consistent with x), but (possibly) more specified than x. Further, since we want to end up with a complete lattice (i.e. one in which every subset has a meet and a join), we will also define two special elements: $\perp$ and $\top$. $\perp$ is called bottom and $\perp \sqsubseteq x$ for all x (i.e. $\perp$ is the least

specified element and approximates everything). $\top$ is called top and $x \sqsubseteq \top$ for all x (i.e. $\top$ is overspecified and is approximated by everything).

These concepts represent a departure from the normal types of structures one talks about when considering data, so the definitions can be expected to seem a bit fuzzy. Hopefully the construction of the lattice of our example set will clarify things somewhat.

4. CONSTRUCTING THE LATTICE OF S-EXPRESSIONS

Before we begin to construct the example lattice, it should be warned that mathematical justifications for certain constructions will be described only in principle. The overall method of construction is patterned after Scott's construction of the "Lattice of Flow Diagrams" in [4], and his presentation is discursive, clear, and fairly complete on these issues.

4.1. METHODS OF CONSTRUCTION

The general plan of construction is to start with a very simple lattice and use basic techniques for combining lattices to build successively larger ones. The two elementary techniques are lattice sum and lattice product. To create the sum of two lattices A and B, it is first assumed that the elements of A and B are disjoint, then the elements of the new lattice (to be called A+B) are taken to be the union of the old elements. We say $x \sqsubseteq y$ in A+B if and only if $x \sqsubseteq y$ in either A or B.

Finally, the two versions each of $\bot$ and of $\top$ are identified to create only one $\bot$ and $\top$ respectively in A+B.

To create the product of two lattices A and B, we let the elements of the new lattice (to be called AxB) be all ordered pairs $\langle x,y \rangle$ where x is in A, and y is in B. We say $\langle x,y \rangle \sqsubseteq \langle x',y' \rangle$ in AxB if and only if $x \sqsubseteq x'$ in A, and $y \sqsubseteq y'$ in B. Clearly, $\langle \bot, \bot \rangle$ and $\langle \top, \top \rangle$ play the roles in AxB of bottom and top respectively.

4.2. THE BASIC LATTICE

We will now create a lattice for the simplest kinds of S-expressions, namely the atoms. Since we want the atoms to be discreet, we say that there are no $\sqsubseteq$ relationships between atoms. This corresponds to our intuition that atoms are somehow "perfect", and no atom really approximates another. Of course $\bot$ approximates every atom, and every atom approximiates $\top$. An intuitive sketch of this lattice is given in Fig. 1. Note that

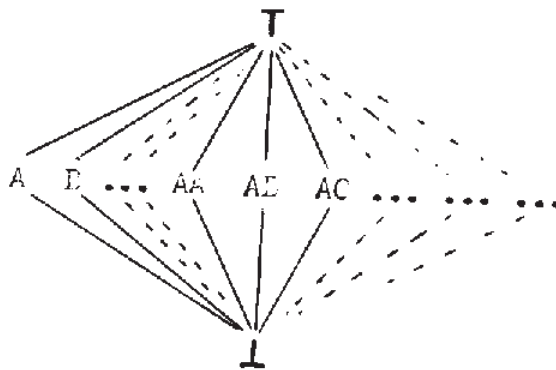


Fig. 1 – The Lattice of Atoms

this lattice is also complete, since every possible subset of atoms has a meet ($\bot$) and a join ($\top$). We call this lattice S_0, since its elements correspond to S-expressions of depth 0.

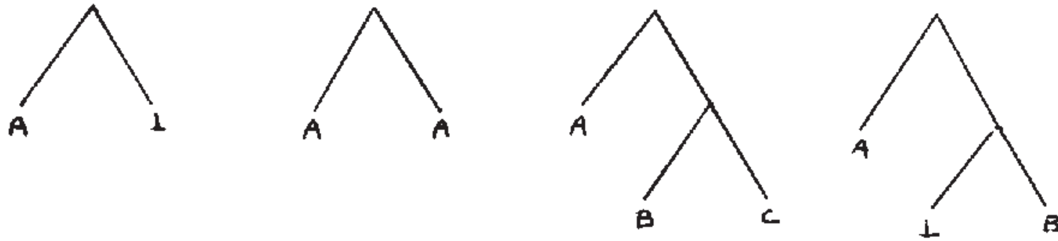
4.3. THE HIGHER LATTICES

We now use our lattice operations to create a new lattice, $S_1 = S_0 + [S_0 x S_0]$. The elements of this lattice correspond to the S-expressions of depth up to 1: atoms, and S-exprs like (x . y) where x and y are atoms. In a like manner we can define in general the complete lattices: $S_{n+1} = S_0 + [S_n x S_n]$ for all n. Where does this lead? We note that $S_0 \subseteq S_1$, and from our definition it follows inductively that $S_n \subseteq S_{n+1}$, for all n. Thus we can see that $S_m \subseteq S_n$ for all $m < n$, and each S_n contains all S-expressions of depth up to n.

4.4. THE FINAL LATTICE

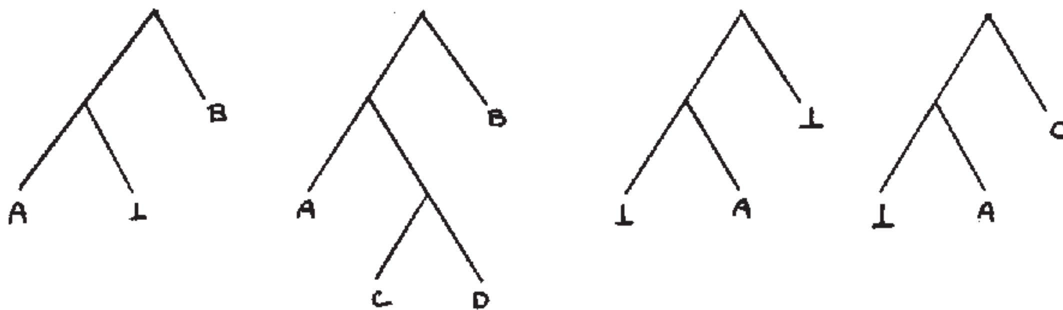
Since we want to talk about the set of all S-expressions, we are naturally led to consider the set union of all the S_n. But does it form a lattice? The answer is that it is a "finitely complete" lattice, since it obeys all definitions but only finite subsets of the lattice have, in general, meets and joins in the lattice. The existence of finite joins follows easily when we consider that any finite set of S-expressions must have an element of maximal depth k. This element and thus all the others in the set, are contained entirely in the subset S_k of the infinite lattice. Since the subset has a join in S_k it has one in the union.

We call this lattice we have just constructed S'. We now pause and ask what sort of elements are contained in S'. Naturally all of the finite S-expressions are in the lattice, but we seem, by way of this peculiar construction, to have included quite a few more. Specifically, there are many elements with $\bot$ and $\top$ as leaves, but it is not obvious what they mean (if we can say that trees mean anything). Recalling our definition of $\bot$ we see that this element in the lattice approximates every other element in S' (i.e. is completely unspecified). What about the element (A . $\bot$)? From our definition of lattice product we see that (A . $\bot$) $\sqsubseteq$ (A . x), for any possible element x. Thus (A . $\bot$) is an element which approximates every other element whose Car is A. We might say that (A . $\bot$) is an element whose only property is that its Car is A. Figure 2 shows this element in tree notation and gives some examples of the objects it approximates. Figure 3 gives more examples of elements which approximate other elements. Elements which contain the top element $\top$ could be described in a manner dual to the treatment of $\bot$. We say that (A . $\top$) is overspecified and is approximated by every element (A . x) for any x. This has less intuitive appeal, so we will not dwell on these elements, nor on expressions with both $\bot$ and $\top$ occuring as leaves.



$(A.\bot)$ $(A.A)$ $(A.(B.C))$ $(A.(\bot.B))$

Figure 2. Elements of S′ approximated by $(A.\bot)$



$((A.\bot).B) \sqsubseteq ((A.(C.D)).B)$ $((\bot.A).\bot) \sqsubseteq ((\bot.A).C)$

Figure 3. Other Approximations

5. COMPLETING THE LATTICE

Although S′ contains all of the elements we were originally interested in (and many more), the fact remains that it is only finitely complete. One might argue that since all of the necessary elements are present, this lattice should suffice as a mathematical model. It turns out, however, that complete lattices have some mathematical properties which are quite powerful and useful in the formulation of "abstract data type". Specifically, we know the "fixed point theorem": a monotonic function on a continuous lattice into itself has a unique least fixed point. This, as well as the fact that adding some infinite elements might lend some additional glamour to our mathematical model, gives motivation for attempting to complete the lattice.

5.1. THE PROJECTION MAPPINGS

Rather than use the mathematical result which simply asserts that any lattice can be completed, Scott [4] chooses to add elements in a constructive way which clarifies exactly what it is that we are adding in the process. The first step is to create a set of "projection" mappings which are called Ψ_n, for all n. The first of these mappings, Ψ_0, maps atoms into themselves and all other elements into $\perp$. The formal definition of the rest of the mappings is:

$$\Psi_{n+1}(s):S_{n+2} \rightarrow S_{n+1} = \begin{cases} s & \text{if } s \text{ is in } S_0 \\ (\Psi_n(x) \;.\; \Psi_n(y)) & \text{otherwise.} \end{cases}$$

Intuitively, each function Ψ_n, when applied to an element x in S_{n+1}, gives us the "projection" of x onto S_n, i.e. we get the "most specified" S-expr of depth n which approximates x. Thus:

$$\Psi_0 \langle A \rangle = A$$

$$\Psi_1 \langle (A.(B.C)) \rangle = (A.\bot)$$

$$\Psi_2 \langle ((A.B).(A.(C.D))) \rangle = ((A.B).(A.\bot))$$

Figure 4 gives examples in tree notation of the projection mappings.

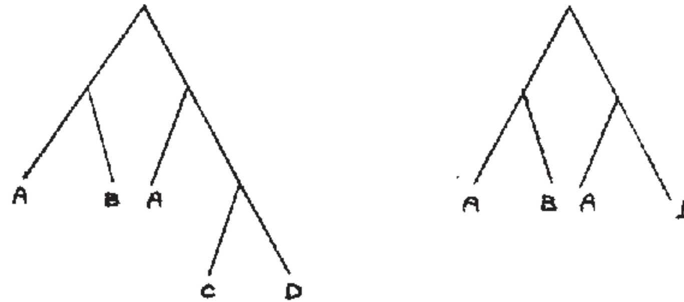
5.2. THE CONSTRUCTION OF NEW ELEMENTS

The way to construct the complete lattice (which we will call S_∞) is to consider infinite "chains" of elements in S'. A "chain" is a sequence of elements $\langle x_i \rangle$, such that:

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

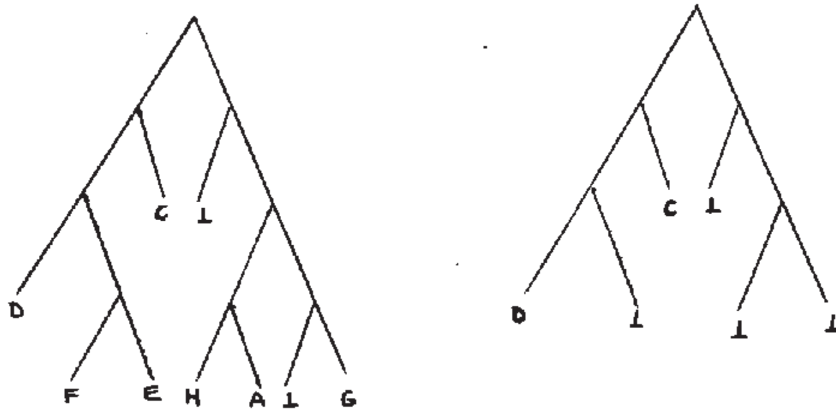
Further, we restrict our attention to those chains for which $\Psi_n \langle x_{n+1} \rangle = x_n$, i.e. in which each element x_n is the maximal element of depth n which approximates the element x_{n+1}. For each such sequence $\langle x_i \rangle$, we will define an element x in S_∞ to be the "infinite join" of the sequence (intuitively, the limit): $x = \bigsqcup_{n=0}^{\infty} x_n$.

A new set of mappings $\Psi_{\infty n}$ is defined on S_∞ which gives us the maximal depth n projection of any element in the new lattice. In the case of x defined above $\Psi_{\infty n}\langle x \rangle = x_n$. In fact, Scott shows that in this type of construction there is a one-one correspondence between elements of S_∞ and all sequences of the type defined above. Thus, every element of S_∞ is defined by its



$$((A.B).(A.(C.D))) \overset{\Psi_2}{\Longrightarrow} ((A.B).(A.\bot))$$

Figure 4a. A projection onto S_2



$$(((D.(F.E)).C).(\bot.((H.A).(\bot.G)))) \overset{\Psi_3}{\Longrightarrow} (((D.\bot).C).(\bot.(\bot.\bot)))$$

Figure 4b. A projection onto S_3

infinite sequence of projections! Note that this holds true even for an element y of finite depth k, since $\Psi_{\infty m}\langle y\rangle$ will be equal to y for all m > k. All lattice properties hold in S_∞, since we now say that $x \sqsubseteq y$ if and only if $x_n \sqsubseteq y_n$ for all n.

6. PROPERTIES OF S_∞

The lattice now contains uncountably many elements and can be shown to be complete. We come finally to the point where we can ask what can be done with it (or at least what interesting things can be said about it).

6.1. THE FUNCTION Cons

It is now time to devote some attention to the long forgotten primitive functions defined on S-expressions. Their definitions on S′ were clear, but now what can we say about them on S_∞? It seems reasonable to be able to talk about the behavior of Cons on infinite elements, so we define it back into existence. Recalling that every element x in S_∞ is uniquely defined by its infinite sequence of projections $\langle x_i\rangle$ we say:

$$\text{Cons}\langle x,y\rangle = \bigsqcup_{n=0}^{\infty}(x_n . y_n) = (x \;.\; y)$$

Thus we have defined the new element (x . y) by its projections. In fact: $(x \;.\; y)_{n+1} = (x_n . y_n)$.

6.2. THE DECOMPOSITION OF S_∞

Under the interpretation given by the new definition of Cons, we can now consider the set $S_\infty x S_\infty$, composed of all ordered pairs of elements of S_∞. This set must be a subset of S_∞. By an

argument similar to Scott's [4], it can be shown that:

$$S_\infty = S_0 + [S_\infty x S_\infty]$$

From this we read intuitively a new definition for S-expressions: "An S-expression is either an atom or a pair of S-expressions". This time there is no restriction to finite constructions.

6.3. THE FUNCTIONS Car, Cdr

With the benefit of the decomposition result above, we can now easily give Car and Cdr their new definitions on S_∞:

$$\text{Car}\langle x\rangle = \begin{cases} x_1 & \text{if } x = (x_1 . x_2) \text{ in } S_\infty x S_\infty \\ \bot & \text{if } x \text{ is in } S_0 \end{cases}$$

$$\text{Cdr}\langle x\rangle = \begin{cases} x_2 & \text{if } x = (x_1 . x_2) \text{ in } S_\infty x S_\infty \\ \bot & \text{if } x \text{ is in } S_0 \end{cases}$$

6.4. CONTINUITY OF FUNCTIONS

It was hinted that the fixed point theorem would be used, which holds for monotonic functions (functions which preserve the partial ordering $\sqsubseteq$). Do our primitive functions on S_∞ satisfy this? The answer is yes, trivially from the new definitions. In fact a stronger result can easily be proved. A subset X is "directed" if every finite subset of X has an upper bound in X. A function is said to be "continuous" on a lattice if it preserves the join of directed subsets. Cons, Car, and Cdr are continuous on S_∞. Finally, Cons (which is technically a function from $S_\infty x S_\infty$ into S_∞) is continuous separately in each of its two arguments.

7. CLASSES OF ELEMENTS OF S_∞

Our next step in talking about S_∞ (and its worth as a mathematical model), is to consider, as we did for S', what sorts of elements are contained in the lattice. Moreover, we seek to discover what correspondences exist between the lattice elements and the various objects we might want to model.

Having mentioned mathematical models, it seems now appropriate to mention a set of objects from the "real world": the set of LISP S-expressions, (rather, the set of symbolic representations for the machine storage of LISP S-expressions). These are admittedly only "real" when considered relative to our uncountably infinite lattice of objects constructed with vaguely defined symbols. Nevertheless this set will give us something interesting to compare our lattice with. For generality, the distinction will not be made between the set of LISP S-expressions and its commonly used subset of "list structures" (as defined in [7]). The functions CAR, CDR, and CONS (distinct from Car, Cdr, and Cons defined on S_∞) will have the obvious definitions on this set.

7.1. NOTATION

We will use the standard "graphical" notation for these LISP structures, using boxes for machine "cells" which act as nodes on a tree[7]. This notation will help make the distinction between the domains clear in the illustrations. We wish to model the set of LISP S-expressions, and these will always be drawn as machine

cells. Elements of our lattice S_∞ will be drawn as binary trees and described in dot notation.

7.2. THE PERFECT ELEMENTS

Since our intuitive understanding of the special elements $\perp$ and $\top$ in our lattice S_∞ is relatively weak, we will define a "perfect" element to be one which has no occurrences of $\perp$ or $\top$. This definition corresponds well to the intuition that trees are really correct only when everything about them is fully specified but nothing is overspecified (whatever "overspecified" means).

7.3. THE FINITE PERFECT ELEMENTS - FP

The set S' of finite S-expressions is a subset of S_∞. We first consider the subset of perfect elements which are in S'. We call this subset FP. In a sense, FP gets us exactly back to where we started: the set of symbolic S-expressions as initially defined in section 2. Given this fact, it should not be surprising that there is a one-one correspondnce between elements of FP and elements of PURE LISP (i.e. allowing no cycles in its structures) [1]. Although painfully obvious, we describe a correspondence mapping ϕ:PURE LISP -> FP as,

$$\phi\langle x\rangle = \begin{cases} x & \text{if } x \text{ is an ATOM symbol} \\ (\phi\langle x_1\rangle . \phi\langle x_2\rangle) & \text{if } x \text{ is a cell with } x_1 \text{ in its left half [CAR] and } x_2 \text{ in its right half [CDR]} \end{cases}$$

Figure 5 gives examples of the correspondence ϕ. At this point, it may seem that the entire construction has been a vacuous exercise in creative algebra, since we could have gotten this far without any mention of lattices. Hopefully, enumeration and

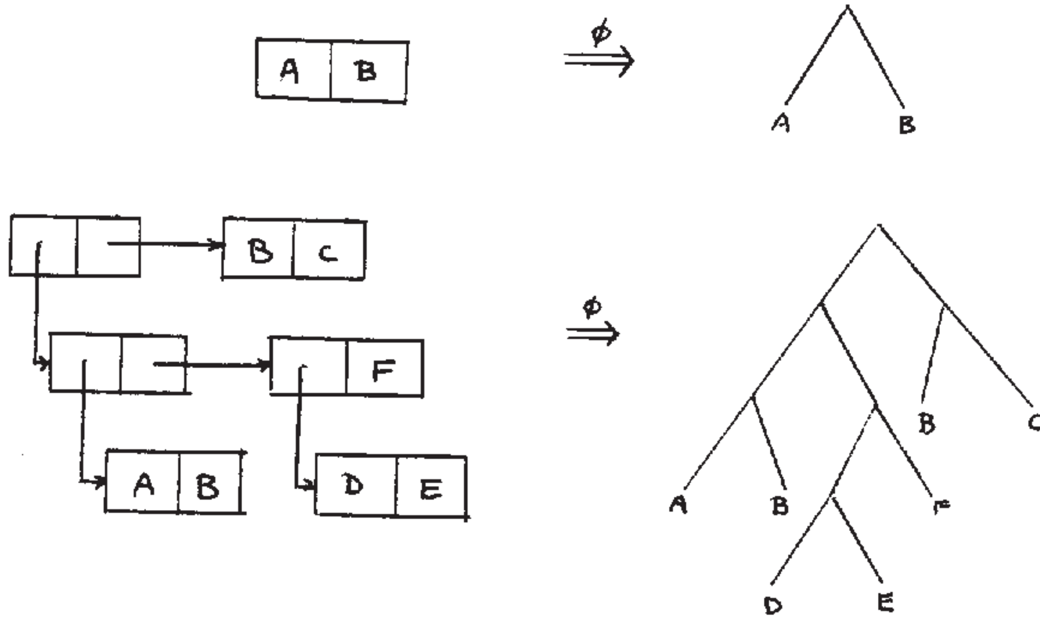


Figure 5. The correspondence ϕ:PURE LISP -> FP

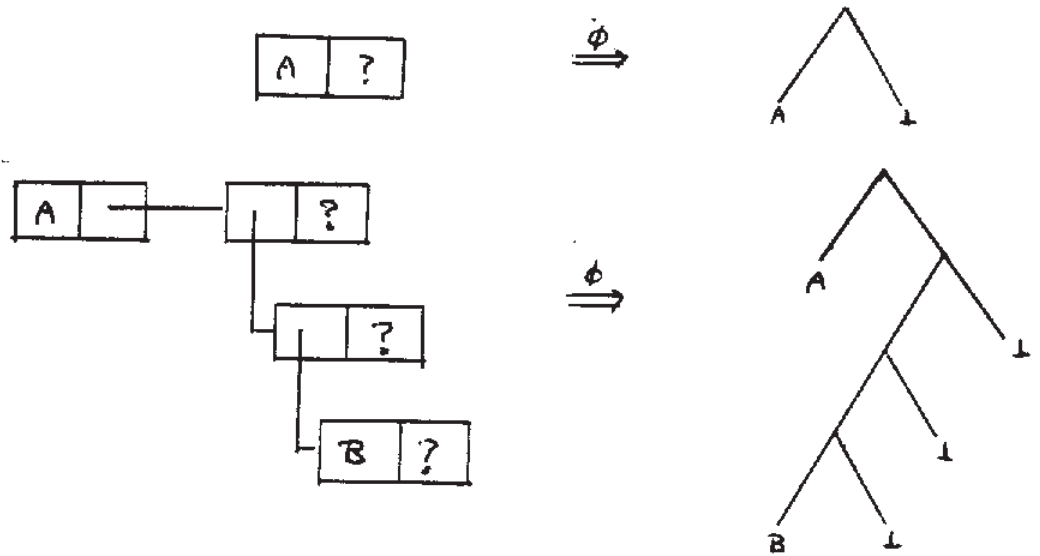


Figure 6. The correspondence ϕ:PURE LISP with "?" -> F

comment on some other classes will alleviate this unpleasant condition.

7.4. THE FINITE ELEMENTS - F

For completeness, we mention the general class of finite elements which we call F. This name is purely for notational consistency, since F = S′. The "imperfect" elements of F have already been mentioned in section 4.4. We reiterate that F contains many elements which can approximate in different ways the finite perfect elements FP (FP is naturally a subset of F). One could say that, given a partial description of some element x of FP, there is an element of F which agrees exactly with that specification and contains no other information.

To make this slightly more formal, we now extend our set of PURE LISP expressions to allow the symbol "?" to be written in the cells. This means that we simply don't know what goes there. We now extend the correspondence mapping

ϕ :PURE LISP with "?" -> F to be,

$$\phi\langle x\rangle = \begin{cases} \bot & \text{if } x \text{ is "?"} \\ x & \text{if } x \text{ is an ATOM symbol} \\ (\phi\langle x_1\rangle . \phi\langle x_2\rangle) & \text{if } x \text{ is a cell with } x_1 \text{ in its left half [CAR] and } x_2 \text{ in its right half [CDR]} \end{cases}$$

Figure 6 gives examples of this correspondence. This version of ϕ is still almost one-one (the exception will be discussed in section 8.1.), and maps onto those elements of F which don't contain $\top$. For lack of intuition, the rest of the elements in F (those which do contain $\top$) are ignored.

7.5. THE ALGEBRAIC PERFECT ELEMENTS - AP

PURE LISP, however, is of very limited practicality, and extended LISP (which we call simply LISP) is the commonly used version [1]. Convenience is gained by allowing side effects to occur, and the more powerful primitives REPLACA and REPLACD are used to do the necessary structure manipulation. The result is a much wider class of possible LISP structures, since the CAR and CDR pointers in cells may be arbitrarily assigned. The LISP programmer is warned in [1] and [7] about infinite search around circular structures. These possibly circular structures are present, though, and can be of practical use, so we consider them.

The simplest example of a circular structure is a cell whose CAR is the atom A and whose CDR points back at the cell itself. This is clearly circular and exhibits infinite behavior, since no matter how long you follow the CDR of the structure, the CAR of the result is still A. Now we can't talk about circular binary tres, but we do have infinite elements in S_∞. In fact, we observe that there is an element in S_∞ which acts just like this circular list. It is shown in figure 7.

There is a better way than observation to assert the existence of this particular infinite element in S_∞. If we let x be a variable over LISP cells and atoms, we see that this cell satisfies and is completely described by the equation: x = CONS<A,x>. Looking at S_∞, we see that Cons<A,x> is a

continuous function from S_∞ into itself. By the fixed point theorem, there is a unique minimal x which satisfies: x = Cons<A,x>. This element x in S_∞ clearly must be the element shown in figure 7.

What about more general structures, i.e. some finite collection of cells with pointers going incomprehensibly in all directions (should we call this a "swarm")? We assert that these all have counterparts in S_∞ by the following method. Label the cells $x_0, x_1, \ldots, x_{n-1}$ where x_0 is the "top" or "first" cell. For each cell x_i, write an equation describing it: x_i = Cons<y,z> where y and z may be atoms or the names of some of the n cells. This gives us a system of n equations in n variables. This represents a function from $(S_\infty)^n$ into $(S_\infty)^n$. It can be easily shown that $(S_\infty)^n$ is a complete lattice and that the function is continuous on this lattice. Thus there is a least fixed point $(a_0, a_1, \ldots, a_{n-1})$ satisfying all of the equations. The element a_0 is a member of S_∞ and exhibits the desired behavior.

To formalize this we define a new correspondence function ϕ':circular LISP structures -> S_∞ as,

> ϕ'<x> = if x contains n cells $(x_0, \ldots, x_{n-1})$ then a_0 where $(a_0, \ldots, a_{n-1})$ is the least fixed point of the mapping $(S_\infty)^n \rightarrow (S_\infty)^n$ defined by the n equations x_i = Cons<y,z>.

An example of this correspondence is given in figure 8.

We define the set AP to be the range of this correspondence mapping when the domain is considered to be all finite LISP structures with at least some circularity. Note that

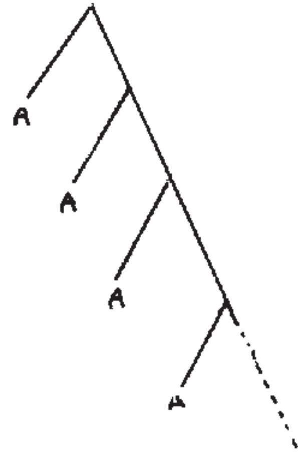
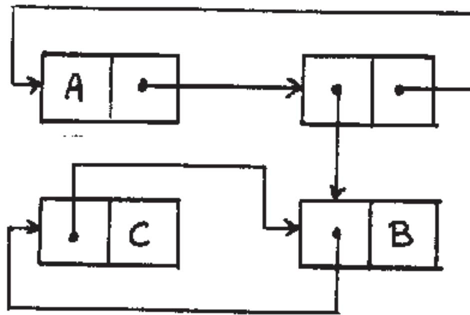


Figure 7. A circular list and its corresponding tree



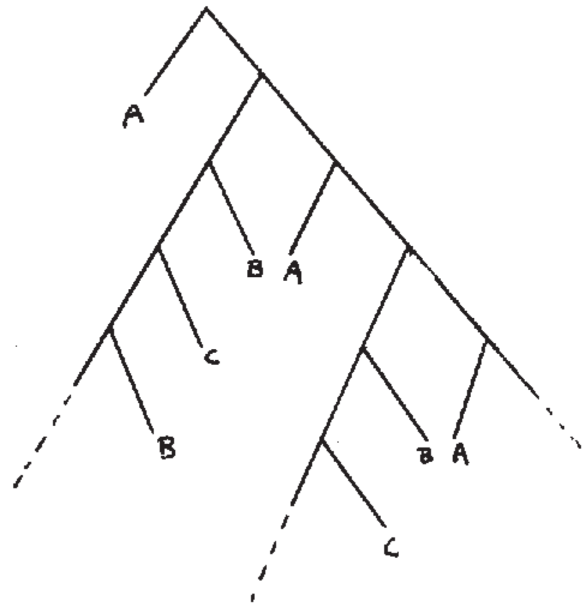


Figure 8. The correspondence ϕ':circular LISP structures -> AP

this mapping would work for PURE LISP -> FP (i.e. on non-circular elements) by the same definition, but we restrict it to keep FP and AP from overlapping.

Is it really true that the value of this mapping is something which acts like its corresponding LISP structure? By "act like" we mean exhibit identical behavior when tracing around the structure with CAR and CDR (or Car and Cdr in S_∞ respectively). The answer is yes, since the mapping ϕ' can be shown to map atoms to corresponding atoms, and is a homomorphism under the operations CAR (Car) and CDR (Cdr).

Our final question about AP is whether ϕ' is one-one. The answer is immediately no, since there are many LISP structures which have identical images in S_∞. Figure 9 indicates that there are infinitely many LISP structures which correspond to the tree shown in figure 7. This is not really so terrible, though, because we only want our elements in S_∞ to act like their counterparts from LISP. It just so happens that many LISP structures act alike under CAR and CDR. Note that we can't extend our definition of "act alike" to include the operations REPLACA and REPLACD. We can only model the structures they build.

Figure 9. Similar LISP structures

7.6. THE ALGEBRAIC ELEMENTS - A

The class A in S_∞ is defined to be the range of the new correspondence mapping $\emptyset'$ when "?" is allowed in cells. As with the set F, the symbol "?" means that we don't know what goes there and gets mapped onto $\perp$. The formal extension of $\emptyset'$ is easy and is not given. Thus A contains infinite elements which correspond to finite circular LISP structures with unspecified cells. Again, the elements with $\top$ are not mentioned.

7.7. THE TRANSCENDENTAL ELEMENTS - TP and T

The set TP corresponds to the rest of the perfect elements in S_∞ when FP and AP are removed. These are infinite unrepetitive trees, and they model those LISP structures which require infinite storage. Again, the more general class T allows imperfect elements which have this same property.

8. COMMENTS ON S_∞

We have seen the wide variety of elements S_∞ contains. It could be argued that this lattice really isn't so remarkable. If we consider just the real elements, i.e. the perfect ones, we see that they form a trivial sublattice which looks like the lattice of atoms (although uncountably infinite). That is, no perfect element approximates another. The advantages of this construction seem to be twofold. First, we are allowed elements which represent partially specified structures. Second, interesting functions are continuous on this lattice (note here

that we could use conditional expressions and other LISP primitives and maintain this continuity). Thus, we can use the fixed point theorem to assert the existence of many interesting elements. This may seem of only moderate interest here, but then this application of Scott's method has been of a purely syntactic nature. Other applications will be mentioned in section 9.

8.1. LOOSE ENDS - THE ELEMENT $(\bot . \bot)$

One vaguely unsettling problem with this construction on LISP structures is that all trees with only $\bot$ on their leaves are identified with $\bot$. Recall from the construction of S_n that lattice product yields $(\bot . \bot)$ as the new bottom element, and that lattice sum identifies the two bottoms as one new element. Thus $\text{Cons}\langle \bot, \bot \rangle = (\bot . \bot) = \bot$. The problem is that the mapping ϕ would take the cell with both sides containing "?" and map it into $\bot$. This isn't quite accurate, because this cell could then be called an approximation of an atom: $(\bot . \bot) = \bot \sqsubseteq A$. But the one thing we know about this element is that it isn't an atom. We don't know what its Car and Cdr are, but we know it has them.

There is temptation to leave the elements formed in this way during the construction of the lattices S , and say that $\bot \sqsubseteq (\bot . \bot) \sqsubseteq (\bot .(\bot . \bot))$ etc. It is not clear whether this could be done while maintaining the consistency of the results.

8.2. LOOSE ENDS - CONSTRUCTING ELEMENTS OF A

We defined the class A as the range of the mapping ϕ', but there wasn't an effective method given for constructing the

results of the map. A reasonable way to do this would be to use recursion on the depth of the tree. This is based on the fact that for a continuous mapping F on a complete lattice:

$$\text{least fixed point } \langle F \rangle = \bigsqcup_{n=0}^{\infty} F^{n}(\bot)$$

Our system of n equations defined in sect 7.5. is such a map and can be so examined to any desired depth.

9. THE JUSTIFICATION FOR LATTICES

It was mentioned in section 8. that this entire construction has been a purely syntactic exercise. We have described a set, expanded it, and come up with some interesting elements and categories. The method yields more than that, however. Scott points out in [5] that the result is a "space" in a well defined topological sense. Open sets and topologically continuous functions are well defined on it. It is this fact which makes this approach to the description of "data types" a powerful idea. Not only do we have a very general structure, i.e. a data space, to talk about, but there is a very large class of functions which are continuous on it (in the special and useful sense $\sqsubseteq$).

Scott describes this type of lattice as a "continuous lattice" and goes deeply into their topological properties. His main theoretical result is that every topological space can be embedded in a continuous lattice which is isomorphic to its own function space [6]. It seems counter-intuitive since the set of functions on a set is generally quite a bit larger than the set

itself.

The trick seems to be that he restricts his class of functions. To see how this works, we look at the his general method. He begins with a basic domain D_0 (a continuous lattice, of course). He then considers the set of functions $[D_0 \rightarrow D_0]$ which are continuous on D_0. This definition of continuity is in the $\sqsubseteq$ sense we have been using. He imposes an ordering $\sqsubseteq$ on $[D_0 \rightarrow D_0]$ which agrees well with the notion of approximation on a data type; he says $f \sqsubseteq g$ in $[D_0 \rightarrow D_0]$ if and only if $f\langle x\rangle \sqsubseteq g\langle x\rangle$ for all x in D_0. What $f \sqsubseteq g$ means is that f approximates g everywhere. For every element x, g gives a value consistent with that of f, but possibly more specified. Partial functions are included, just like partially specified S-expressions, because $\perp$ is now a meaningful value for a function to have. Under this ordering, $[D_0 \rightarrow D_0]$ turns out to be another continuous lattice!

If we let $D_1 = [D_0 \rightarrow D_0]$ and $D_{n+1} = [D_n \rightarrow D_n]$ we can construct a sequence of continuous lattices composed of higher and higher "function types". Is there a limit lattice which includes them all? Scott says yes, with appropriate isomorphic embeddings from each D_n into D_{n+1}. In fact, the result is D_∞ where the interesting equation holds:

$$D_\infty = [D_\infty \rightarrow D_\infty] \quad \text{(up to isomorphism)}$$

D_∞ is a space of functions of a general type; each function on D_∞ (as long as it is continuous) can be represented by an element in

D_∞ itself, and each element in D_∞ is itself a continuous function on D_∞.

This could be a very powerful mathematical-semantic device for modelling computation, since the problem of function type and self-application is very present in practical computation. Scott mentions that this could develop into the first sound mathematical model for the lambda-calculus (which is type free, but lacks in mathematical foundation).

One question remaining is just what have we lost by restricting ourselves to the functions which are continuous in the sense $\sqsubseteq$ (or for that matter to data types which can be expanded into continuous lattices)? Scott gives good intuitive arguments that all reasonable data types are covered by this theory and that all practical and computable functions are continuous, but it doesn't seem entirely clear yet.

In any case the generality and mathematical soundness of this approach give it great appeal as a method toward developing an effective way to mathematically define computation.

BIBLIOGRAPHY

[1] Dertouzos, M. L. "Structure and Interpretation of Computer Languages", M.I.T., 1972

[2] MacLane and Birkhoff "Algebra", MacMillan, N.Y., 1967

[3] Scott, D. "Outline of a Mathematical Theory of Computation", Proceedings of the Fourth Princeton Conference on Information Science and Systems, 1970

[4] ________ "The Lattice of Flow Diagrams", Lecture Notes in Mathematics 188, Springer-Verlag, 1971

[5] ________ "Lattice Theory, Data Types and Semantics", Courant Computer Science Symposium 2, 1970

[6] ________ "Continuous Lattices", Lecture Notes in Mathematics 274, Springer-Verlag, 1972

[7] Weissman, C. "LISP 1.5 PRIMER", Dickenson Publ., Belmont Calif., 1967