ABSTRACT CONTROL STRUCTURES AND THE SEMANTICS OF QUANTIFIERS

Steven Cushing
Computer Science Department
St. Anselm College
Manchester, New Hampshire, 03102 USA

ABSTRACT

Intuitively, a quantifier is any word or phrase that expresses a meaning that answers one of the questions "How many?" or "How much?" Typical English examples include all, no, many, few, some but not many, all but at most a very few, wherever, whoever, whoever there is, and also, it can be argued, only (Keenan, 1971), also (Cushing, 1978b), and the (Chomsky, 1977). In this paper we review an empirically motivated analysis of such meanings (Cushing, 1976; 1982a) and draw out its computational significance. For purposes of illustration, we focus our attention on the meanings expressed by the English words whatever and some, commonly represented, respectively, by the symbols "$\forall$" and "$\exists$", but most of what we say will generalize to the other meanings of this class.

In Section 1, we review the notion of satisfaction in a model, through which logical formulas are customarily imbued implicitly with meaning. In Section 2, we discuss quantifier relativization, a notion that becomes important for meanings other than $\forall$ and $\exists$. In Section 3, we use these two notions to characterize quantifier meanings as structured functions of a certain sort. In Section 4, we discuss the computational significance of that analysis. In Section 5, we elaborate on this significance by outlining a notion of abstract control structure that the analysis instantiates.

I MODELS AND SATISFACTION

Given a semantic representation language L containing predicate constants and individual constants and variables, an interpretation I of L is a triple $\langle D, R, \{f\}\rangle$, where D is a set of individuals, the domain of I; R is a function, the interpretation function of I, that assigns members of D to individual constants in L and sets of lists of members of D to predicates in L, the length of a list being equal to the number of arguments in the predicate to which it corresponds; and $\{f\}$ is a set of functions, the assignment functions of I, that assign members of D to variables in L. A model M for L is a pair $\langle D, R\rangle$, an interpretation of L without its assignment functions. Since "a factual situation comprises a set of individuals bearing certain relations to each other," such "a situation can be represented by a relational structure $\langle D, R_1, \ldots, R_i, \ldots\rangle$, where D is the set of individuals in question and $R_1, \ldots, R_i, \ldots$ certain relations on D," (van Fraassen, 1971, 107), i.e., in this context, sets of lists of members of D. Models thus serve intuitively to relate formulas in L to the factual situations they are intended to describe by mapping their constants into D and $\langle R_1, \ldots, R_i, \ldots\rangle$. The "variable" character of the symbols assigned values by an f relative to those interpreted by R is reflected in the fact that a set of fs corresponds to a fixed $\langle D, R\rangle$ to comprise an interpretation.

The distinction between R and f gives us two different levels on which the satisfaction of formulas can be defined, i.e., on which formulas in L can be said to be true or false under I. First, we define satisfaction relative to an assignment of values to variables, by formulating statements like (i)-(vi) of Figure 1, where "$M \models (A)\ [f]$" is read as f satisfies A in M or M satisfies A given f. Given these statements, we can define "$A \supset B$", read if A then B, as "$\neg(B \,\&\, \neg A)$", and we can define "$(\exists x)$", read for some x or there are x, as "$\neg(\forall x)\neg$". Second, we can define satisfaction by a model, by saying that M satisfies A, written "$M \models (A)$", if $M \models (A)\ [f]$ for whatever assignment functions f there are for M. Intuitively, this can be read as saying that A is true of the factual situation that is represented by the relational structure into which L is interpreted, regardless of what values are given to variables by the assignment functions of an interpretation. For some discussion of the cognitive or psychological significance of these notions, see Miller (1979a,b) and Cushing (1983).

II SIMPLE AND RELATIVIZED QUANTIFICATION

Statement (v) of Figure 1 characterizes simple quantifications like (1), which represent the meanings expressed by sentences like (2), for which x = "x" and A = (3), while (vi) characterizes relativized quantifications like (4), which

(i) $\underline{M} \models (\underline{x}_1 = \underline{x}_2)[\underline{f}]$ iff (i.e., if and only if) $\underline{f}(\underline{x}_1) = \underline{f}(\underline{x}_2)$

(ii) $\underline{M} \models (\underline{P}(\underline{x}_1,\ldots,\underline{x}_n))[\underline{f}]$ iff $(\underline{f}(\underline{x}_1),\ldots,\underline{f}(\underline{x}_n)) \in \underline{R}(\underline{P})$

(iii) $\underline{M} \models (\underline{A} \,\&\, \underline{B})[\underline{f}]$ iff $\underline{M} \models (\underline{A})[\underline{f}]$ and $\underline{M} \models (\underline{B})[\underline{f}]$

(iv) $\underline{M} \models (\neg\, \underline{A})[\underline{f}]$ iff it is not the case that $\underline{M} \models (\underline{A})[\underline{f}]$

(v) $\underline{M} \models ((\forall\, \underline{x})\, \underline{A})[\underline{f}]$ iff $\underline{M} \models (\underline{A})[\underline{f}']$ for whatever assignments $\underline{f}'$ for $\underline{M}$ are like $\underline{f}$ except perhaps (i.e., at most) at $\underline{x}$

(vi) $\underline{M} \models ((\forall\, \underline{x})(\underline{B};\underline{A}))[\underline{f}]$ iff $\underline{M} \models (\underline{A})[\underline{f}']$ for whatever assignments $\underline{f}'$ for $\underline{M}$ are like $\underline{f}$ except perhaps at $\underline{x}$ for which $\underline{M} \models (\underline{B})[\underline{f}']$

Figure 1: Typical Satisfaction Statements for Logical Formulas (adapted from van Fraassen, 1971, 108)

represent the meanings expressed by sentences like (5), for which $\underline{x}$ and $\underline{A}$ are as for (2) and $\underline{B}$ = (6):

(1) $(\forall\, \underline{x})\, \underline{A}$

(2) Whatever there is is interesting.

(3) Interesting(x)

(4) $(\forall\, \underline{x})(\underline{B};\underline{A})$

(5) Whatever is linguistic is interesting.

(= Whatever there is that is linguistic is interesting.)

(6) Linguistic(x)

In general, $\underline{B}$ and $\underline{A}$ in (4) are lists of formulas in $\underline{L}$, the relativization formulas and the principal formulas, respectively, of (4); both lists for (5) are of length 1, and we will assume lists of that length for the rest of our discussion.

Given (v) and (vi), the relativized quantification (4) is logically equivalent to the simple quantification (7), reflecting the synonymy of (5) with (8), for example, but this fact does not generalize to quantifier meanings other than $\forall$, because there are quantifiers $\underline{Q}$ for which there is no truth-functional connective c for which (9) is logically equivalent to (10):

(7) $(\forall\, \underline{x})(\underline{B} \supset \underline{A})$

(8) Whatever there is, if it is linguistic, then it is interesting.

(9) $(\underline{Q}\, \underline{x})(\underline{B};\underline{A})$

(10) $(\underline{Q}\, \underline{x})(\underline{B}\ c\ \underline{A})$

For a formal proof of this important fact, see Cushing (1976; 1982a). The relativized case must thus be considered separately from the simple one, despite its apparent superfluity in the case of $\forall$, which suffices for our purposes (with $\exists$) in all other respects.

III QUANTIFIER MEANINGS AS STRUCTURED FUNCTIONS

Statement (vi) characterizes the meaning expressed by (4) implicitly, by stating the conditions under which (4) can be said to be either true or false; in general, other "truth values" are also required for natural language (Cushing, 1982a; 1983), but we will not discuss those cases here. Given (vi), we can characterize the meaning expressed by (4) explicitly as a function, (11), that generates a truth value $\underline{u}$ from $\underline{M}$, $\underline{f}$, $\underline{x}$, $\underline{B}$, and $\underline{A}$:

(11) $\underline{u} = \forall(\underline{M},\underline{f},\underline{x},\underline{B},\underline{A})$

If we let σ^* be the function that maps a predicate in $\underline{L}$ to its extension relative to $\underline{M}$, $\underline{f}$, and $\underline{x}$ -- i.e., the subset of $\underline{D}$ whose members make that predicate satisfied by $\underline{M}$ given $\underline{f}$ when assigned individually as values to $\underline{x}$ --, then we can replace the English clause on the right-hand side of the "iff" in (vi) with the equivalent set-theoretic formulation (12), and thus (vi) itself with the equivalent statement (13):

(12) $\underline{D} \cap \sigma^*(\underline{M},\underline{f},\underline{x},\underline{B}) \subseteq \sigma^*(\underline{M},\underline{f},\underline{x},\underline{A})$

(13) $\underline{M} \models (\forall\, \underline{x})(\underline{B};\underline{A})[\underline{f}]$

iff $\underline{D} \cap \sigma^*(\underline{M},\underline{f},\underline{x},\underline{B}) \subseteq \sigma^*(\underline{M},\underline{f},\underline{x},\underline{A})$

In other words, (4) is true if and only if the intersection of $\underline{D}$ with the extension of $\underline{B}$ is wholly contained as a subset in the extension of $\underline{A}$. $\underline{D}$ is omitted from the right-hand side of the "$\subseteq$" in (12) for more general reasons that need not concern us here.

Letting $\underline{a}_i$, $\underline{i}$=0,1,2, be set variables, we can abstract away from the sets in (12) to get the relation -- i.e., in this context, boolean-valued function -- (14), which can be factored into more basic component set-theoretic relations as shown in (15), in which the superscripts and subscripts indicate which argument places a relation is to be

applied to, when the steps in the derivation are reversed:

(14) $\underline{a}_0 \cap \underline{a}_1 \subseteq \underline{a}_2$

(15) $\underline{a}_0 \cap \underline{a}_1 \subseteq \underline{a}_2$

$\subseteq_1^2(\underline{a}_0 \cap \underline{a}_1, \underline{a}_2)$

$(\subseteq_1^2, \cap_1^2)(\underline{a}_0, \underline{a}_1, \underline{a}_2)$

Finally, dropping the arguments $\underline{a}_i$ from the last line of (15), we get the quantificational relation, $\rho_\forall$, expressed by $\forall$, as shown in (16):

(16) $\rho_\forall = (\subseteq_1^2, \cap_1^2)$

The function (11), the meaning expressed by (4), thus consists of instances of two other functions: σ^*, which generates sets from models, assignments, and predicates; and $\rho_\forall$, which generates truth values from sets; all related as in Figure 2. Strictly speaking, the left-most instance of σ^* is really a different function -- viz., the three-input function $\sigma^*(\ ,\ ,\ ,\underline{true})$, rather than the four-input function $\sigma^*(\ ,\ ,\ ,\ )$ --, since true is a constant that must occur there, but this technicality need not worry us here. Each function in Figure 2 provides the same mapping as is provided collectively by the lower-level functions to which it is connected. "Select sets", for example, is a mnemonic dummy-name for the function that consists of the three indicated instances of σ^*, through which these three independent instances interface with $\rho_\forall$. The effect of $\forall$, in turn, is achieved by applying $\rho_\forall$ to whatever three sets are provided to it by Select-sets. Like Select-sets, $\rho_\forall$ can also be further decomposed into subfunctions, as shown in Figure 3, which reflects the structure of (15). The important point here is not the tree notation per se, but the fact that a functional hierarchy is involved, of the indicated sort. Any other notation that is capable of expressing the relevant relationships would be just as -- in certain respects, more (Cushing, 1982a, Figures 10 and 11) -- adequate for our purpose. For some general discussion of meanings as structured functions, see Cushing (1979a).

The two immediate subfunctions of $\forall$ differ in one key respect, namely, in that Select-sets has nothing to do specifically with $\forall$, but would be required in the analysis of any quantifier meaning; everything that is peculiar to $\forall$ is encoded entirely in $\rho_\forall$. An analysis of $\exists$, for example, can be obtained by simply replacing $\rho_\forall$ in Figure 2 with an appropriate $\rho_\exists$, viz., the one in (17), in which Comp is a function that take the complement of a set -- i.e., those members of $\underline{D}$ that are not in the set --, and Pair is a function that duplicates its input:

(17) $\rho_\exists = (\neq_1^2, \mathrm{Comp}_1^1, \cap_2^4, \mathrm{Pair}_1^1)$

This relation unravels to exactly the correct truth condition and satisfaction statement for relativized $\exists$, just as (16) does for $\forall$.

In the general case, we also have to include a third subfunction, $\Pi_{\underline{Q}}$, which generates a numerical parameter, as indicated in Figure 4. The function

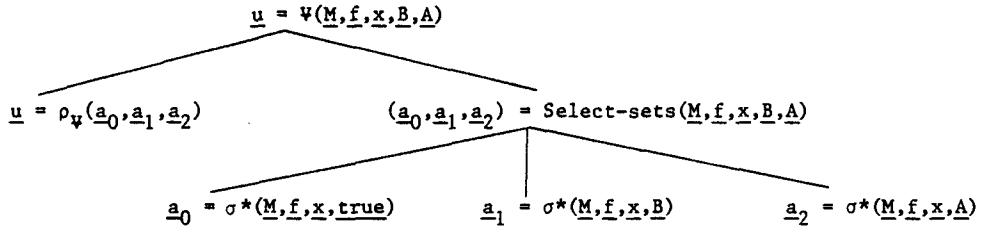


Figure 2: Functional Decomposition of Relativized $\forall$

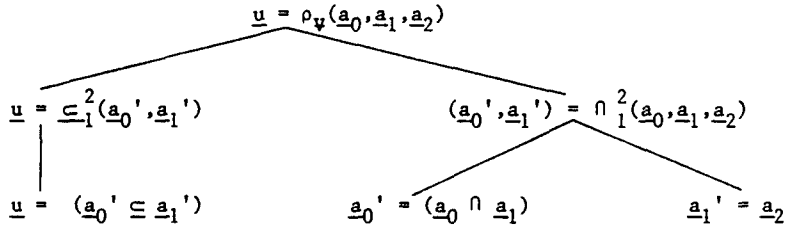


Figure 3: Functional Decomposition of the Quantificational Relation Expressed by Relativized $\forall$

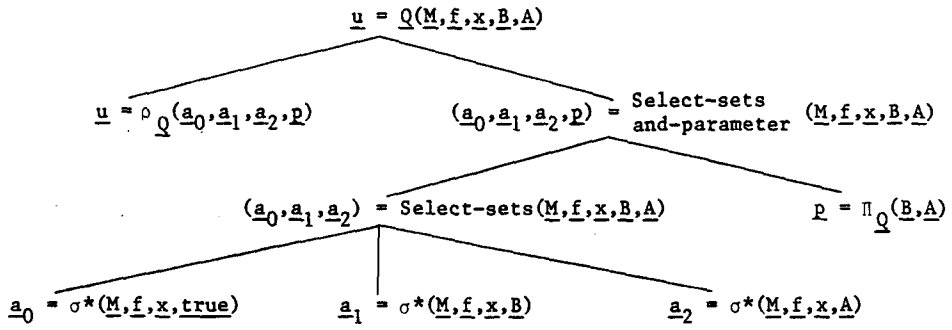


Figure 4: Functional Decomposition of the General Relativized Quantifier Meaning

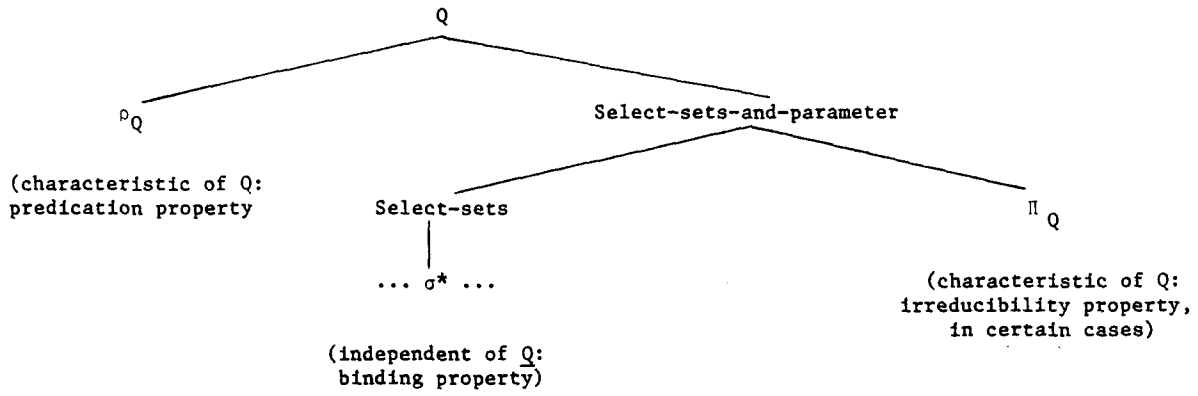


Figure 5: Relationships Among Quantifier Meaning Subfunctions and the Properties They Explicate

Select-sets -- more precisely, its σ* subfunctions -- explicates the binding property common to all quantifier meanings, because it characterizes the extensions of predicates (via σ*) by removing the relevant variable from the purview of the assignment, as can be seen clearly in statement (vi) of Figure 1. The function ρ_Q, the quantificational relation expressed by Q, explicates the predication property of quantifier meanings, by virtue (primarily) of which different quantifier meanings are distinguished. Its quantificational relation is what a quantifier predicates; the extensions of the predicates it is applied to are what it predicates that of. The intuition that quantifiers are in some sense predicational is thus explained, even though the notion that they are "higher predicates" in a syntactic sense has long since failed the test of empirical verification. The function Π_Q is what underlies the irreducibility property of certain quantifier meanings, by virtue of which (9) is not logically equivalent to (10). Like ρ_Q, Π_Q is specifically characteristic of Q. For present purposes, we can consider it to be null in the case of ∀ and ∃. The relationship of these functions to the quantifier meanings they decompose is indicated schematically in Figure 5.

IV COMPUTATIONAL SIGNIFICANCE

It must be stressed in the strongest possible terms that the motivation for the analysis embodied in Figure 4 has absolutely nothing at all to do with computational considerations of any sort. Computational relevance need not imply linguistic or cognitive relevance, any more than mathematical relevance does, and vice versa. See Cushing (1979b) and Berwick and Weinberg (1982) for relevant argumentation. On the contrary, the analysis is motivated by a wide range of linguistic and psychological considerations that is too extensive to review here. See Cushing (1982a) for the full argument. The analysis does have computational significance, however, which follows post facto from its form and consists in the fact that functional hierarchies of exactly the sort it exemplifies can be seen to make up the computational systems that are expressed by computer programs.

If we take a program like the one in Figure 6, for example, and ask what functions -- i.e., mathematical mappings with no side effects -- it

involves, we can answer immediately with the list in (18):

(18) (i) $y = x + 2$

(ii) $z' = (y + x)^2$

(iii) $z = z'^2$

(iv) $z' = (y - x)^2$

(v) $z = -z'^2$

(vi) $w = z - 1$

There is a function that gets a value for y by adding 2 to the value of x, a function that gets a value for z' by squaring the sum of the values of x and y, and so on. Closer examination reveals, however, that there is an even larger number of other functions that must be recognized as being involved in Figure 6. First, there is the function in (19), which does appear explicitly in Figure 6, but without an explicit output variable:

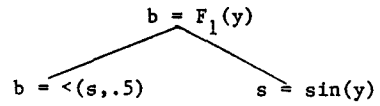
(19) $s = \sin(y)$

Second, there is the boolean-valued function in (20), which also appears in Figure 6, but with no indication as to its functional character:

(20) $b = <(s,.5)$

More significantly, there is a set of functions that are entirely implicit in Figure 6. Since (19) generates a value of s from a value of y, and (20) generates a value of b from that value of s, there is an implicit function -- call it F_1 -- that is defined by that interaction, generating that value of b from that value of y, as indicated in (21):

(21)



Similarly, since (18)(ii) and (iv) generate values of z' independently from values of x and y, and these are then taken by (18)(iii) and (v), respectively, to generate values of z, there are two further implicit functions -- call them F_4 and

```
BEGIN          y = x + 2;

IF sin y <.5   THEN   z' = (y + x) ** 2;
                      z  = z' ** 2;

               ELSE   z' = (y - x) ** 2;
                      z  = -(z' ** 2);

               w = z - 1;

END;
```

Figure 6: A Simple Sample Program

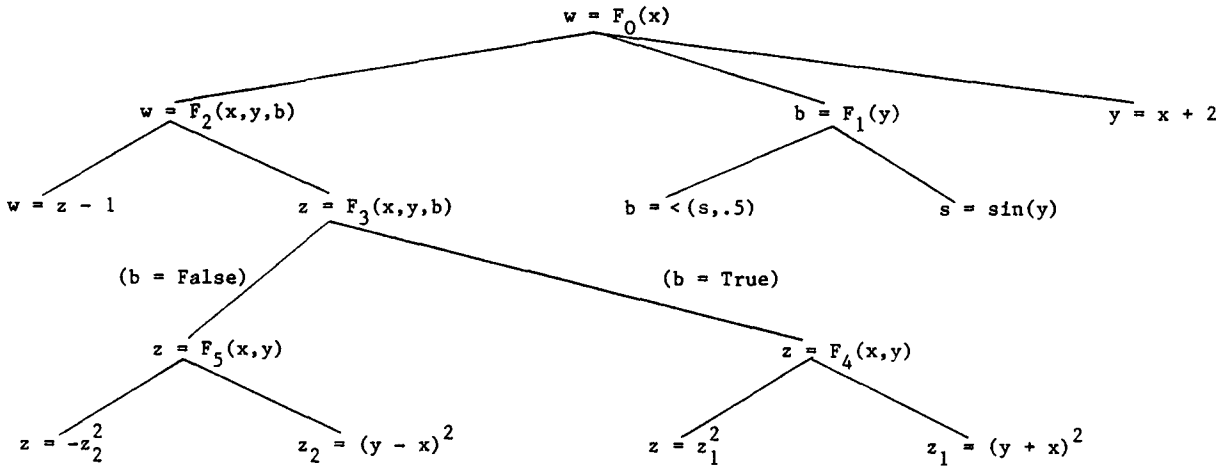
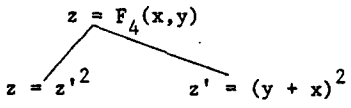


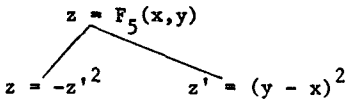
Figure 7: Functional Hierarchy Expressed by the Program in Figure 6.

F_5 -- that are defined by these interactions, as shown in (22) and (23):

(22)

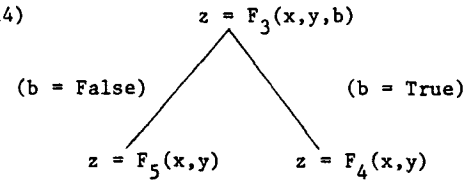


(23)



Since F_4 and F_5 generate different values, in general, for z for the same values of x and y, they interact, in turn, to define a "choice" function -- call it F_3 -- with its alternatives determined, in this case, by the value of b, as indicated in (24):

(24)



Continuing in this way, we can extract two further functions: F_2, which consists of the composition of (18vi) and F_3; and F_0, which consists of the composition of F_2, F_1, and (18i) and defines the overall function effected by the program, as shown in Figure 7.

The variables in Figure 6 are strictly numerical only for the sake of illustration. As we have just seen, even in this case, extracting the implicit functional hierarchy expressed by the program requires the introduction of a non-numerical -- viz., boolean-valued -- variable. In general, variables in a program can be taken to range over any data type at all -- i.e., any kind of object to be processed --, as long as it can be provided with an appropriate implementation, and the same is therefore true, as well, of its implicit functional hierarchy. For an extensive list of references on abstract data types, see Kapur (1980); for some discussion of their complementary relationship with the functional hierarchies expressed by programs, see Cushing (1978a; 1980).

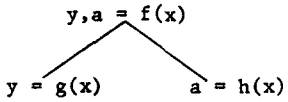
The hierarchy expressed by an assembly language program, for example, might well involve variables that range over registers, locations, and the like, and bottom-node functions that store and retrieve data, and so on, just as Figure 4 has bottom-node functions that assign extensions to predicates and form the intersections of sets. Given implementations of these latter functions, Figure 4 defines a computational system, just as much as Figure 7 does, and so can be naturally implemented in whatever programming language those implementations are themselves formulated in.

V ABSTRACT CONTROL STRUCTURES AS FUNCTIONAL HIERARCHIES

The control structure indicators -- the words IF, THEN, ELSE, the semi-colons, the sequential placement on the page, and so on -- in Figure 6 are ad hoc syntactic devices that really express semantic relationships of functional hierarchy, viz., those shown in Figure 7. In general, we can identify a control structure with such a functional hierarchy. For some background discussion relevant to this notion, see Hamilton and Zeldin (1976). A control structure can be said to be legitimate, if its interfaces are correct, i.e., if the subfunctions do effect the same mappings as the functions they purportedly decompose. Of the three structures in Figure 8, for example, only (ii) is legitimate, because (i) and (iii) each generates a value of a as a side effect -- i.e., a is generated by a subfunction, but not by the overall function --, and b in (i) appears from nowhere -- i.e., as an input to a subfunction, but not as an input to the overall function, or as an output from another subfunction on the same level.

Structure (iii) can be made legitimate by adding a to the output list of its top-most function, as indicated in (25):

(25)



This structure represents one sort of parallel process, in which two functions operate independently on the same input variable to generate values for different output variables. Structure (i) can be made legitimate by adding a to the output list of its top-most function, as in the

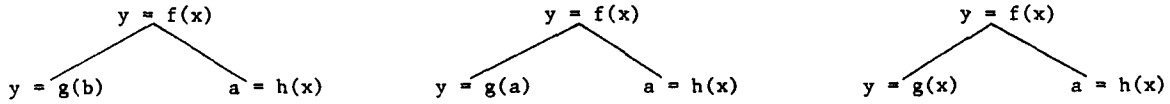
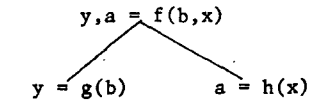


Figure 8: Three Possible Control Structures

case of (iii), and also adding b to that function's input list, as indicated in (26):

(26)



This structure represents a different sort of parallel process, in which two functions operate independently on different input variables to generate values for different output variables. Structure (ii) represents a non-parallel, sequential process in which two functions operate dependently, one generating a value for an output variable that is used by the other as an input variable.

In general, the variables in these structures can be interpreted as really representing lists of variables, just as "B" and "A" in (4) can be interpreted as representing lists of predicates. Of these three legitimate structures, then, only (ii) can be seen as occurring in Figure 7. Figure 4 also contains a different structure (for Select-sets) that combines the features of (25) and (26).

The important point here is that functional hierarchies comprising legitimate control structures are inherent in the systems expressed by workable programs. As such, they have proven useful both as a verification tool and as a programming tool. For some discussion of the relationship that ought to exist, ideally, between these two different modes of application, see Hamilton and Zeldin (1979).

Through interaction with those who have written an existing program, one can derive the abstract control structure of the system expressed by the program, make that structure legitimate, and then make the corresponding changes in the original program. In this way, subtle but substantial errors can be exposed and corrected that might not be readily revealed by more conventional debugging techniques.

Conversely, given a legitimate control structure -- such as the one for quantifier meanings in Figure 4, for example --, the system it comprises can be implemented in any convenient programming language -- essentially, by reversing the process through which we derived Figure 7 from Figure 6, adapted to the relevant language. For some discussion of software that automates this process, see Cushing (1982b) and Wasserman and Gutz (1982). For a good description of the vision that motivates the development of this software -- i.e., the ideal situation toward which its development is directed --, see Hamilton and Zeldin (1983). Our present concerns are primarily theoretical and thus do not require the ultimate perfection of this or any other software.

A number of interesting variants have been proposed to make this notion of control structure applicable to a wider class of programs. See Martin (1982), for example, for an attempt to integrate it with more traditional data base notions. Harel (1979) introduces non-determinacy, and Prade and Vaina (1980) attempt to incorporate concepts from the theory of fuzzy sets and systems. Further development of the latter of these efforts would be of particular interest in our present context, in view of work done by Zadeh (1977), for example, to explicate quantifier and other meanings in terms of fuzzy logic.

ACKNOWLEDGEMENTS

I would like to thank Fred Barrett, Mitka Golub, and Robert Kuhns for helpful comments on an earlier draft, and Margaret Moore for typing the final manuscript.

REFERENCES

Berwick, Robert C. and Amy S. Weinberg. 1982. "Parsing Efficiency, Computational Complexity, and the Evaluation of Grammatical Theories." Linguistic Inquiry. 13:165-191.

Chomsky, Noam. 1977. Essays on Form and Interpretation. New York: North-Holland.

Cushing, Steven. 1976. "The Formal Semantics of Quantification." UCLA doctoral dissertation. Ann Arbor, Michigan: University Microfilms.

_____. 1978a. "Algebraic Specification of Data Types in Higher Order Software." Proceedings, Eleventh Annual Hawaii International Conference on System Sciences. Honolulu, Hawaii.

_____. 1978b. "Not Only Only, But Also Also." Linguistic Inquiry. 9:127-132.

_____. 1979a. "Lexical Functions and Lexical Decomposition: An Algebraic Approach to Lexical Meaning." Linguistic Inquiry. 10:327-345.

_____. 1979b. "Semantic Considerations in Natural Language: Crosslinguistic Evidence and Morphological Motivation." Studies in Language. 3:181-201.

_____. 1980. "Software Security and How to Handle It." Chapter 4 of Advances in Computer Security Management, Volume 1. Rullo, Thomas A. (ed.). Philadelphia: Heyden & Son.

_____. 1982a. Quantifier Meanings: A Study in the Dimensions of Semantic Competence. North-Holland Linguistic Series, Volume 48. Amsterdam: North-Holland.

_____. 1982b. Letter to ACM Forum. Communications of the ACM. 25:951.

_____. 1983. "Dynamic Model Selection in the Interpretation of Discourse." In Cognitive Constraints on Communication: Representations and Processes. Vaina, Lucia and Jaakko Hintikka (eds.). Dordrecht: Reidel.

van Fraassen, Bas C. 1971. Formal Semantics and Logic. New York: Macmillan.

Hamilton, Margaret and Saydean Zeldin. 1976. "Higher Order Software -- A Methodology for Defining Software." IEEE Transactions on Software Engineering. SE-2:9-32.

_____. 1979. "The Relationship Between Design and Verification." Journal of Systems and Software. 1:29-56.

_____. 1983. "The Functional Life Cycle and Its Automation." Journal of Systems and Software. 3:25-62.

Harel, David. 1979. "And/Or Programs: A New Approach to Structured Programming." Specifications of Reliable Software. IEEE Catalog No. 79 CH1401-9C.

Kapur, Deepak. 1980. "Towards a Theory for Abstract Data Types." TR-237. Laboratory for Computer Science. Massachusetts Institute of Technology.

Keenan, Edward L. 1971. "Quantifier Structures in English." Foundations of Language. 7:255-284.

Martin, James. 1982. Program Design Which Is Provably Correct. Carnforth, England: Savant Institute.

Miller, George A. 1979a. "Construction and Selection in the Mental Representation of Text." Cahiers de l'Institut de Linguistique de Louvain. 5:185-197.

_____. 1979b. "Images and Models, Similes and Metaphors." In Metaphor and Thought. Ortony, Andrew (ed.). Cambridge: Cambridge University Press.

Prade, Henri and Lucia Vaina. 1980. "What 'Fuzzy HOS' May Mean." Proceedings, Fourth International Computer Software and Applications Conference. IEEE Catalog No. 80 CH1607-1.

Wasserman, Anthony I. and Steven Gutz. Reply to Letters to ACM Forum. Communications of the ACM. 25:951-2.

Zadeh, Lotfi A. 1977. "PRUF - A Language for the Representation of Meaning in Natural Languages." Proceedings, Fifth International Joint Conference on Artificial Intelligence, Volume 2. Cambridge, Massachusetts.