PROGRAMMING LANGUAGE DATA STRUCTURES:

A COMPARATIVE STUDY

by

Bobbie Ann Othmer

Rutgers University
The State University of N. J.
Department of Computer Science
New Brunswick, New Jersey 08903

Technical Report No. 30

March 4, 1974

PROGRAMMING LANGUAGE DATA STRUCTURES: A COMPARATIVE STUDY

1. Introduction.

Recently, Scott and Strachey [7,8] developed a mathematical semantics for programming languages. In their theory a program is viewed as a function on the set of states of the computation. To identify the function associated with a program via the semantics, two additional functions are defined, the environment function and the store function. In order that these two functions be well-defined, one must specify their domains and ranges; Strachey [9] proposed that a study of these two functions and their domains and ranges would reveal the basic structure of a language and clarify underlying differences between languages. He carried out such a study for Algol 60 and PAL.

In this report we follow Strachey's approach and specify the domain and range structure of the two functions for Algol 60, Fortran IV, and SNOBOL4. The following section contains the definitions of the basic functions, domains, and ranges from the general theory. In section 3 the domain and range structure is given for each of the three languages. In the last section we compare the domain and range structure of the three languages and close with a discussion of the utility of this approach to characterizing programming languages.

2. Domains and functions central to the semantics.

Before discussing the particular domains and ranges of interest to us here we describe several ways sets can be constructed from other sets. In the following we call both domains and ranges domains.

Domains are either elementary or compound. Elementary domains are primitive in the sense that they cannot be constructed from other domains; they correspond to simple types in programming languages.

Compound domains are constructed from other domains using any of several set operations. One such operation is sum, denoted by +. An element of $D_0 + D_1$ is either an element of D_0 or an element of D_1, but not of both. $D_0 \times D_1$ denotes the product of D_0 and D_1. An element of $D_0 \times D_1$ is an ordered pair whose first component is an element of D_0 and whose second component is an element of D_1. D^n is the notation for DxDx...xD (n factors). D^0 is the empty set. D^+ is the infinite sum $D^1 + D^2 + D^3 + \ldots$, and D^* is the infinite sum $D^0 + D^1 + D^2 + D^3 + \ldots$. Finally, $[D_0 \rightarrow D_1]$ denotes the domain of functions which map D_0 to D_1. Scott and Strachey do not include all set-theoretic functions from D_0 into D_1 in $[D_0 \rightarrow D_1]$. However, we do not want to go into the restrictions here; see [7].

The domain of identifiers or names, Id, is important in all programming languages. Id is a very simple domain. Its only relation is equality; two names are either the same or they are not. Id differs from other domains used for the semantics of programming languages in that its members are part of the text of the program.

Let D be the domain whose elements are those things which can be given names. The function, ρ, which maps Id to D is the environment function; it is one of the basic functions of the mathematical semantics. Different programming languages can have widely different environment functions. The domain D, which Strachey calls the domain of denotations, includes local and global variables. It does not include integers or real numbers because we want to think of an identifier as referring to a variable, not its value which may change many times in the course of a computation.

Let L stand for the domain of locations or generalized storage places. A variable can be thought of as having an L-value and an R-value. The L-value of a variable is the fixed location denoted by the name of the variable, and the R-value is the contents of the location, the value of the variable which changes dynamically. L must be a part of D for all languages with an assignment statement. V is the domain of all stored values. Integers, reals, and Boolean values would be in V. In fact, V contains all objects which can be the value of the right

hand side of an assignment statement.

D is taken to contain anything which can be passed to a procedure as a parameter. One can think of the actual parameters as being evaluated and then stored somewhere. The locations may have names in the program as in the case of programmer-supplied procedures which have formal parameters, or they may not have names in the program as in the case of certain built-in functions. Thus we think of a procedure as being defined on D*. If we thought of procedures as being defined on V* (values), it would be difficult to distinguish between different kinds of calls. The major difference among kinds of calls is in what is passed to a procedure, whether a value or an address of a location or of another procedure (as in call by name) is passed. This difference occurs before parameters are evaluated.

Another important domain is the set of all machine states, denoted by S. The state of a machine includes a description of the contents of the locations, so if $\sigma \in S$, σ contains a mapping, the store function, from L into V. The state might also include some indication of which locations are being used, and something about I/O. Scott and Strachey have not yet formalized the properties of S, and we do not want to attempt it here, so in the following we will think of S as an elementary domain.

3. Examples.

Algol 60

Most of the following is due to Strachey [9]. We will point out where our account differs from his.

1. Elementary domains
   - T Booleans
   - N Integers
   - R Reals
   - Q Strings
   - J Labels
   - L Locations
   - S Stores (machine states)

T, N, and R have their usual mathematical properties. Although Algol 60 has no string operations, we include Q because strings may be passed as parameters. Strachey includes J and S here because he does not want to consider their structure at this time.

2. Derived domains

Expression values [1]

$E = J + V$

There are three kinds of expressions in Algol 60: arithmetic, Boolean, and designational. Arithmetic expressions return something in N or R when evaluated; Boolean expressions return true or false, and designational expressions return a label.

Arrays

Each element of an array denotes a location. In Algol all elements of an array must be of the same type. Thus the store function must map all of the locations of the array onto values of the same type.[2] Let A1 be the set of all vectors. Then

$$A1 = L + L^2 + L^3 + \ldots = L^+$$

Let A2 be the set of all two-dimensional matrices. Then

$$A2 = A1 + A1^2 + \ldots. = A1^+ = L^{++}$$

If A is the set of all arrays,

$$A = L^+ + L^{++} + L^{+++} + \ldots = L^{+^+}$$

Procedures

$$P = [D^* \rightarrow [S \rightarrow S]] + [D^+ \rightarrow [S \rightarrow [V \times S]]]$$

---

1 Strachey has E = D+V

2 Strachey doesn't add this restriction.

As we said above, the parameters of a procedure are viewed as being in D. The number of parameters a procedure may have is unspecified. (A procedure may be parameterless). Thus the parameter list is in D^*. A call of an ordinary non-type procedure in Algol 60 causes a state transformation. Thus a non-type procedure is in $[D^* \to [S \to S]]$. A type procedure produces a result which is in V and may have a side effect, altering the store. Thus a type procedure is in $[D^* \to [V \times S]]]$.

Calls by name

$W = S \to [E \times S]$

There is a similarity between formal parameters called by name and type procedures. However, calls by name have no parameters, and the values they produce may be in J as well as in V.

Switches

A switch in Algol 60 is an ordered list of designational expressions. When in the program a call is made of the switch, an expression in the list is chosen on the basis of the subscript of the switch call, and that expression is evaluated giving a result in E and possibly causing a side effect. Thus we think of a switch as being in W^+.

Denotations

$D = L$ locations
$+ P_+$ procedures
$+ L^+$ arrays
$+ W$ calls by name
$+ W^+$ switches
$+ Q$ strings
$+ J$ labels

Q is included in D because a string can be passed to a procedure as an actual parameter and is thus denoted by the formal parameter. In Algol 60 strings cannot be assigned as the value of any variable.

Stored values

$V = T + N + R$

3. The environment function[3]

In Algol 60 the environment function changes each time a block is entered or a procedure is called, assuming that the procedure has local variables declared. Because all identifiers must be declared at the beginning of some block, changes in the environment function are easy to determine.

Fortran IV

1. Elementary Domains

   T Booleans

   N Integers

   R Reals

   Q Strings

   J Labels

   L Locations

In Fortran one can assign strings of length ≤4 to variables and strings of length ≤8 to double word variables, but Fortran then treats the strings as numbers. It has no string handling operations. For this reason, the domain Q is not considered as being in V. As in Algol, however, strings can be passed as actual parameters to subroutines; thus Q is in D.

2. Derived domains

   Expression values

   $E = V$

Fortran IV has only arithmetic and Boolean expressions.

Arrays

$A = L^{+}$

---

3 Strachey does not discuss the character of the environment function for Algol 60.

Arrays in Fortran are like those in Algol. All elements of an array must be of the same type in Fortran as well as in Algol, so the same restriction on the store function holds here.

Procedures

$$P = [D^* \to [S \to S]] + [D^* \to [S \to V]] + [D^* \to [S \to [V^* \times S]]]$$

A statement function in Fortran has no side effects so has functionality $[D^* \to [S \to V]]$. A function subprogram has functionality $[D^* \to [S \to [V \times S]]]$, but this domain is contained in $[D^* \to [S \to [V^* \times S]]]$. A subroutine may have an arbitrary number of parameters or may be parameterless. It may or may not return values, but in general can have side effects, so subroutines have functionality $[D^* \to [S \to S]]$ or $[D^* \to [S \to [V^* \times S]]]$.

Denotations

$$D = L + L^{+}{}^{+} + P + Q + J$$

Q is included in D for the same reason it is included in D for Algol 60.

Stored values

$$V = T + N + R$$

3. The environment function

The environment function is not as easily defined for Fortran as for Algol since in Fortran it is not necessary to declare variables explicitly unless they have special properties. In Fortran the environment function for a program is static. Each subroutine has its own environment function because all of its variables are local except for those declared to be COMMON.

SNOBOL4

SNOBOL 4 is an interesting and complex language designed with operations for handling strings of symbolic data. A central feature of SNOBOL4 is pattern matching. Patterns, which can determine flow of control and provide for value assignment, can be very complex. Some

other features of interest are the lack of restrictions on the type of the current value of a variable, the facility for defining one's own data types, and the success/failure flow of control.

1. Elementary domains
   - N Integers
   - T Booleans
   - R Reals
   - Q Strings
   - K Keywords
   - L Locations
   - J Labels
   - POC Polish-prefix object code

For SNOBOL4 a generalized storage unit is used to contain a value. A variable refers to the location in this generalized store which contains the current value of the variable. Any variable whose name is a nonnull string is a natural variable. Natural variables are organized in a symbol table. Each entry in the table resembles the diagram in Figure 1. Information about a variable such as its name, value, and data type are stored in the corresponding entry. A block of this kind will be treated as one storage location.

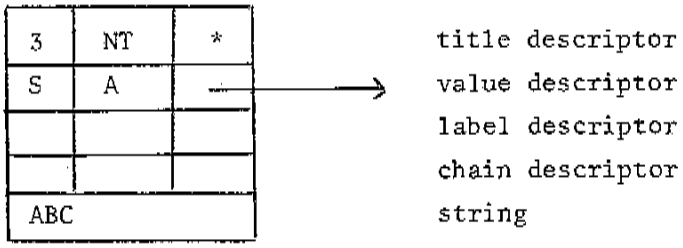

Figure 1. Symbol table entry for a variable whose name is ABC and whose value is a string.

Q is included as an elementary domain because strings are primitives in SNOBOL4 in the sense that a string can't be considered as

being constructed from elements of other data types. The set of strings is an important domain in SNOBOL4.

There is no data type Boolean in SNOBOL4, but predicate functions either fail or return the null value, so the elementary domain T is needed.

Keywords are variables whose values are information internal to the program, e.g., the number of statements which have been executed at the point in the program where the keyword appears. The set of keywords, K, is included as an elementary domain.

A source program in SNOBOL4 is converted into Polish-prefix object code before being interpreted. There is a function, CODE, which will convert a string of characters into object code during execution. In order to deal with functions such as CODE, the domain of Polish-prefix object code, POC, is included as an elementary domain.

2. Derived domains

Arrays

Arrays are more general in SNOBOL4 than in Fortran or Algol since the array elements are not required to be of the same type. Each element of an array can be, for example, an array or pattern.

ARRAY is the set of arrays where an array is defined as follows:

1) An ordered set of locations is an array, i.e., anything in L^+ is an array.

2) An array is an ordered set of locations, arrays, tables, and/or patterns.

Tables

A table can be thought of as an associative array. Of course, the values of the elements can be of any type. A table will be thought of as a set of ordered pairs of locations where the first location of a pair contains or points to the index of the element, and the second contains or points to the value of the element.

TABLE is the set of tables where table is defined as follows:

1) A set of ordered pairs of locations is a table, i.e., anything in L^{2+} is a table.

2) A set of ordered pairs where the first in the pair is a location, and the second is a location, an array, a table or a pattern is a table.

Expression values

$E = V + PAT + J$

Naturally E contains V, but some expressions evaluate to patterns, so E also contains PAT. The goto field can contain expressions which must evaluate to labels, so J is also in E.

Procedures

$P = [D^* \rightarrow [S \rightarrow [V \times S]]] + [D^* \rightarrow [S \rightarrow S]] + [D^* \rightarrow [S \rightarrow [PAT \times S]]]$

Many kinds of functions in SNOBOL4 fit into the domain P. Primitive functions, predicates, and programmer defined functions are in P. Predicates return success or failure. Primitive functions and programmer-defined functions return a value which may be in V or PAT or which may return failure. Programmer-defined functions can return in three different ways: RETURN which returns a value, FRETURN which indicates failure of the function call, and NRETURN which returns a name. We have interpreted name return as returning a location, and since L is in V, this case is included in the first part of P.

DEFINE is a primitive function which is used to define procedures. It sets up a function descriptor and returns the value null. Its domain structure adds nothing more to P.

DATA is another primitive function of SNOBOL4. Its purpose is to provide for programmer-defined data types. A call of DATA(p) sets up a description of the new data type defined by p = name(f_1, ..., f_n) and returns a null value. When a call of name(e_1, ..., e_n) is made, n new variables are created which may be referenced by f_i (name). The function DATA and the functions which create objects of defined data types are contained in P.

Patterns

The central feature of SNOBOL4 is pattern matching. Patterns are constructed by the concatenation and alternation of strings, functions and unevaluated expressions. Conditional and immediate value assignment may also be specified as part of a pattern. The programmer is able to construct very complex patterns; given any recursively enumerable set of strings, there is a pattern which matches the set. It is not always obvious that a pattern corresponds to a set of strings because a substring of a longer string may or may not match a pattern depending on the location of the substring in the longer string. This location may be a function of the longer string. It is, nevertheless, possible to consider a pattern as corresponding to a set of strings. Given a pattern, there is a partial recursive function $f : Q \to T$, which is defined by the SNOBOL4 interpreter, such that $f(q) =$ true if q matches the pattern. Therefore, a pattern corresponds to a set of strings.

J. F. Gimpel [2] has suggested defining a pattern as a function P(S,c) where S is a string and c is the pre-cursor position, with the value of P being a sequence of post-cursor positions in the order of their preference, i.e., $P : Q \times N \to N^*$. The concatenation of two patterns P_1 and P_2 is defined as

$$(P_1\ P_2)\ (S,c) = P_2\ (S,\ P_1\ (S,c)),$$

and the alternation of P_1 and P_2 is

$$(P_1/P_2)\ (S,c) = P_1\ (S,c)\ P_2\ (S,c).$$

He states several properties of patterns and notes that the built-in pattern ABORT doesn't satisfy them. But ABORT is a special case anyway.

R. D. Tennent [10] has given another formulation of patterns. He considers a pattern to be a function of six variables: a subject string, a cursor, a subsequent to be applied if the local match is successful, an alternative to follow if the local match is unsuccessful, a failure route to follow if the global match fails, code for

conditional assignments, and the current store. Its domain structure is as follows:

$$PAT = Q \times N \times Sb \times Alt \times Fl \times C \to S \to S$$

where $Sb = N \times Alt \times C \to S \to S$, $Alt = S \to S$, $Fl = S \to S$, and $C = S \to S$.

Tennent's formulation appears to be too complicated and is not useful for this discussion. Gimpel's formalization is interesting and probably useful. For our purposes, however, we want something more suggestive of what is meant by pattern matching.

Patterns will be treated as predicates of strings. Given a string the pattern will return "success" if the string contains a given configuration of characters, and "failure" otherwise. So we have for $p \in PAT$, the set of patterns, $p : L \to T$. But the presence of unevaluated expressions in patterns complicates matters. Pattern matching might have to stop while a function call or an expression is evaluated, thus possibly causing side effects. So we have

$$PAT = [L \to [S \to [T \times S]]].$$

Denotations

$$D = K + L + P + J + ARRAY + TABLE + PAT$$

Stored values

$$V = N + R + Q + T + L + POC$$

L is included in V as well as in D. Functions can return a variable, and some expressions evaluate to variables. Here we interpret variable as meaning location. Also, the unary name operator · returns the location of a variable which can then be assigned to another variable. For example, the statement

```
Z = .X<3>
```

assigns to Z the location of the subscripted variable X<3>.

POC is included in V because the primitive function CODE returns executable object code as its value.

3. The environment function.

In SNOBOL4 all variables are global unless they are specified by a DEFINE statement as being local. Labels are also considered to be global. There are no declarations which state that a variable is of a certain data type. The environment function changes only when a function with local variables is entered. In SNOBOL4 one name may simultaneously denote a location, a function, and a label. Thus the environment function is a function of two variables, a name and a context.

4. Conclusions

Our study of the domain structure of Algol, Fortran and SNOBOL yielded some interesting results about the differences among the three languages. Note that the elementary domains and V are the same for Fortran and Algol, but D for Algol contains two domains, W and W^+, which aren't in D for Fortran. The presence in Algol of calls by name and switches is a major difference between the two languages. Also, the fact that Algol expressions can have a label as value while Fortran expressions take values only in V is apparent from the domain structure.

More differences appear when we add SNOBOL4, a string processing language, to our consideration. SNOBOL4 is a more complex language with many features not available in Algol or Fortran, and much of this complexity is apparent in the domain structure. Domains such as PAT, TABLE, the keywords K, and the object code POC found in SNOBOL indicate some of the richer structure. SNOBOL's domain of values V contains domains not found in V for Algol or Fortran. The fact that Q is in V indicates that strings are treated as data in SNOBOL. The definition of array is much more general in SNOBOL than in Fortran or Algol, and the extent of the differences is reflected in the domain structure.

One can distinguish some interesting differences among the languages from the descriptions of procedure structure. Fortran has the statement function which has no side effects. It also has subroutines which can return more than one value. Algol has calls by name which neither Fortran nor SNOBOL has. SNOBOL has pattern-valued functions. Our domain structure analysis does not distinguish calls by value, found in SNOBOL and Algol, from calls by value-result found in Fortran, but it does distinguish calls by name from other kinds of calls.

The environment function, even in the brief description given here, differs considerably among the languages. In Algol every variable must be declared at the beginning of a block or procedure. In Fortran and SNOBOL all variables need not be explicitly declared, and thus it is not as easy to determine the environment function for them as it is for Algol. The environment function for Algol can change every time a block or procedure is entered. In Fortran the environment function is static. The environment function for SNOBOL can change upon entering a procedure if the procedure has local variables. Also, SNOBOL's environment function is a function of two variables instead of one, since in SNOBOL the same name can be used for a variable, a label, and a function simultaneously.

As expected our consideration of the domains of these languages has yielded little about the flow of control or about the kinds of operators and statements a language has. We need to specify what we mean by the store S and by state transformations before we can characterize these features of a language.

We think that an effort to determine the domains of a programming language is useful; one gains a more complete understanding of the structure of the language than one would get from learning the language only for the purpose of programming. Additionally, the information obtained does help to characterize languages and to distinguish different features available in the languages studied.

ACKNOWLEDGEMENT

I would like to thank Professor Richard J. Orgass for many helpful discussions about the work in this paper and for his help in writing it.

References

1. Froberg, Ekman, "Introduction to ALGOL Programming", Oxford University Press, London, 1967.

2. Gimpel, J. F., "A Theory of Discrete Patterns and Their Implementation in SNOBOL4", CACM 16, 2 (Feb., 73), 91-100.

3. Griswold, R., "The Macro Implementation of SNOBOL4", Freeman, San Francisco, 1972.

4. Griswold, R., Poage, J., and Polensky, I., "The SNOBOL4 Programming Language", Prentice-Hall, Englewood Cliffs, N. J., (second edition) 1971.

5. IBM System Reference Library, IBM System 1360, FORTRAN IV Language.

6. Naur, P., "Revised Report on the Algorithmic Language ALGOL 60".

7. Scott, D., "Outline of a Mathematical Theory of Computation", Proc. 4th Annual Princeton Conference on Information Sciences and Systems, 169-176, 1970.

8. Scott, D., and Strachey, C., "Towards a Mathematical Semantics for Computer Languages", Symposium on Computers and Automata, Microwave Inst. Symposia Series 21, Polytechnic Institute of Brooklyn (1972).

9. Strachey, C., "The Varieties of Programming Languages", Oxford University Computing Laboratory, Technical monograph PRG-10, 1971.

10. Tennent, R. D., "Mathematical Semantics of SNOBOL4", Proc. ACM Symposium on Principles of Programming Languages, Oct. 1973, 95-107.