MASSACHUSETTS INSTITUTE OF TECHNOLOGY

PROJECT MAC

Computation Structures Group Memo 125

An Examination of a Proposed Mathematical Semantics Specification for SNOBOL4

by

Stephen T. Kent

(Mr. Kent is an NSF Fellow in the Computer Systems Research Group at Project MAC.)

June 1975

An Examination of a Proposed Mathematical Semantics Specification for SNOBOL4

Stephen Thomas Kent

Abstract

This paper will examine some aspects of the mathematical semantics specification for SNOBOL4 as proposed in a recent paper by R.D. Tennent [9]. In particular it will consider how the proposed specification differs from the language represented by the SNOBOL4 interpreter as distributed by Bell Telephone Laboratories [1,2,3,4]. The resolution of some of these differences and the extension of the specification to include all of the language will be discussed.

Introduction

SNOBOL4 is a high level programming language developed by Bell Laboratories in the late 1960's as the successor to SNOBOL3. The SNOBOL family of languages was developed for use in manipulating symbolic formulas and has been implemented on a number of computers

[3]. The language is interpreted, and the compiler/interpreter system is distributed by Bell Laboratories in the form of a collection of macro assembly code for which corresponding macro prototypes must be supplied in order to implement the system [1]. In this respect, SNOBOL is unlike other programming languages, in that the implementation specifications are expressed in the collection of macro code and the associated definition of the macros [2]. The macro specifications are not formal, and thus the need for a formal description of the language exists.

Mathematical Semantics

The mathematical approach to semantics provides a definition of a programming language in terms of functions which map syntactic specifications of a program into a semantic domain specifying the meaning of a program. The concept of mathematical semantics is the result of work by D. Scott and C. Strachey [6]. The domains used to model data types in this theory form complete lattices, and the functions which operate on them are continuous in this lattice framework [5].

This paper assumes a familiarity on the part of the reader with respect to these concepts of mathematical semantics as presented by Strachey [7]. Also the concept of a continuation as a model for a jump is assumed [8].

As Tennent notes in his paper, SNOBOL4 is an interesting language to which to apply the techniques of mathematical semantics . While mathematical semantics is oriented more toward applicative constructs in languages, SNOBOL4 is primarily an imperative language, making permanent modifications to the store rather than invoking new environments in which expressions are evaluated and results returned. The control structure is unusual and varied. In addition to simple jumps, transfer of control can be conditioned on success or failure as signalled by expression evaluation. In pattern matching there is an elaborate backtracking control structure, and the trace facilities impose a control structure similar to PL/1 condition signalling.

The storage structure of SNOBOL4 is unusual in several areas. Input and output are performed by associating a variable with either an input or output specification so that reference to the variable causes the desired data transmission to be performed as part of the storing or retrieving of a value at a storage loaction. Labels, functions and operators can all be assigned through the use of built-in functions. Every string which textually occurs in or is created during the execution of a SNOBOL4 program can act both as a variable name and a data value.

All of these features of the language combine to cause the mathematical semantics of SNOBOL4 to be rather complicated in areas where, for other programming languages, the constructs are more straightforward.

Mathematical Semantics Specification of SNOBOL4

We shall not present here the full details of the mathematical semantics for SNOBOL4 developed by Tennent in his paper. Rather we shall deal with those areas of his mathematical semantics description which are peculiar to SNOBOL4 and describe how they attempt to model some of the more unusual features of the language. We shall also discuss those features of the language which are omitted from or are inaccurately represented in his description.

The function of the environment component of a semantic construct is to map identifiers to their locations, while the store component maps abstract storage locations to the values which are stored in them [7]. In his semantic specification for SNOBOL4, Tennent does not have a separate environment component, but rather a complex store component.

Expression evaluation in SNOBOL4 can either succeed or fail, and transfer of control can be based on this form of signalling by an expression. Hence Tennent proposes two continuations to be associated with each expression, an expression continuation (K) and a failure continuation (FL). The expression continuation maps the value returned by the successful expression evaluation into a conventional store-to-store function of the type usually associated with a continuation.

K = E -> S -> S

The failure continuation simply maps stores to stores in the usual way.

$$FL = S \rightarrow S$$

This gives rise to an expression interpretation function which has the functionality shown below.

<expression interpretation function> : $EXP \rightarrow K \times FL \rightarrow S \rightarrow S$

There are two major classes of variables in SNOBOL4, natural and created. "A natural variable is any variable whose name is a nonnull string." (1) A created variable comes into existence during program execution as a result of use of a built-in function which constructs a data structure, e.g., ARRAY or TABLE. Every natural variable appears in precisely one place in SNOBOL4 storage and can serve both as a variable name and as a string value for some variable. This dual role of a natural variable is further extended to include the possibility that the string serves as a label in the program text, and thus must be linked to the code with which it is associated in the program text.

Since natural variables can act as both names and values, it is necessary to distinguish between these roles in describing the value of an expression. Thus the expression domain (E) is composed of locations (L) and storable values (V).

$$E = L + V$$

---

(1) Griswold, Poage and Polonsky, pg. 131.

Furthermore, the location domain is the lattice sum of natural variable names and created variable names (NM).

$$L = Q + NM$$

The domain of storable values includes the usual primitive data types, strings (Q), integers (N), reals (R), arrays (AR), and some more complex ones, names (NM), patterns (P), tables (TB), unevaluated expressions (W), code (C) and user-defined data types (DF). The resulting domain equation is shown below.

$$V = Q + N + NM + R + P + AR + TB + W + C + DF$$

Tables form the basis for generalized data structures of heterogeneous composition. Components of tables are accessed by selectors which range over all the data types. A call to the function TABLE to create a table can optionally specify an initial size and the quantum by which the table will grow as more objects are added. (2) (3) We will see in our discussion of garbage collection that these optionallly supplied values must be saved for accurate modelling of SNOBOL4 storage. Tennent does not give an equation for this domain, but using the * notation as exemplified by Strachey for array domain descriptions, we can write: (4)

---

(2) Griswold, 1972, pp. 112-113.

(3) Griswold, Poage and Polonsky, pp. 118-119.

(4) Strachey, 1973, pg. 16.

TB = L*

Arrays, which can also be of heterogeneous composition, are created by calls to the function ARRAY, which takes arguments specifying the number of dimensions and their extents. (5) (6) Again, using Strachey's notation, we can supply the array domain equation omitted by Tennent.

AR = (L*)*

User-defined data types are created through calls to the built-in function DATA, specifying the name of the new type and the prototype which defines it. References to the components of the data type are made by functional references where the prototype-supplied name of the desired field is applied to an instance of the data type. (7) (8) At the level of domain definition, user-defined data types can be represented as linear aggregates of locations, thus this domain equation omitted by Tennent is simply:

DF = L*

To support these data types which were not explicitly defined by Tennent, we shall need to add another component to the store domain in

---

(5) Griswold, 1972, pp. 110-111.

(6) Griswold, Poage and Polonsky, pp. 120-121.

(7) Griswold, 1972, pp. 105-110.

(8) Griswold, Poage and Polonsky, pp. 123-127.

order to identify the type of the data .

Unevaluated expressions (W) are used when one wishes to create an expression which is to be evaluated at a later point in the computation. (9) (10) Such an expression will use the continuations and store provided at the point when it is evaluated so that the functionality is given by:

$$W = K \times FL \to S \to S$$

The data type code (C) is used to denote the Polish prefix form into which SNOBOL4 source programs are translated by the compiler. It is also possible to use a built-in function CODE to compile a program represented as a string in a SNOBOL4 program, hence there is a need to explicitly include this data type in our collection of domains. Tennent represents code as:

$$C = S \to S$$

In SNOBOL4 it is possible to redefine all functions, even built-in functions and operators. Hence all functions evaluate their own arguments at the time of invocation. The domain of functions is thus defined by:

$$F = W^* \to K \times FL \to S \to S$$

---

(9) Griswold, 1972, pg. 140.

(10) Griswold, Poage and Polonsky, pp. 50-55.

As noted earlier, the storage of SNOBOL4 is very complex. We have already seen that natural variables can act as string values, variable names and labels referring to data of type code. Functions, binary operators and unary operators are denoted by character strings in a source program and thus by natural variables in SNOBOL4. These attributes must also be recorded in storage associated with the natural variable which names the function or operator, regardless of the other attributes which the natural variable may possess. When combined with the usual functionality of the store, which is simply a mapping from locations to storable values, we have this complex domain supplied by Tennent:

$$S = [L \rightarrow V] \times [Q \rightarrow F \times F \times F \times C]$$

Despite this degree of complexity in the store function, we still have not included any provision to identify the type of the data, to handle I/O or to allow simulation of storage regeneration, the SNOBOL4 term for garbage collection and subsequent recompaction of the address space. We will discuss these problems later in this paper.

SNOBOL4 automatically converts standard data types where necessary during operations which it performs, e.g., built-in arithmetic, and it provides a function, CONVERT, which can be used to explicitly perform any of the conversions which it would internally perform. It is therefore necessary to coerce values in the mathematical semantics in many places. Additionally, a large number of auxiliary functions are postulated by Tennent to support the actual

expression transformation semantic equations. The extent of this use of auxiliary functions which are not then defined by semantic equations will be questioned later.

User-defined functions are specified through the use of the DEFINE built-in function. The user supplies to this function the name, formal parameters, local variables and an optional entry point label, if different from the user-defined function name. Because of the general facilities utilized by all functions for parameter evaluation and reassociation of entry point names, Tennent maintains that user-defined functions need no special domain definition. We will see below that there is a distinction in the fashion in which missing or excess arguements are handled based on the type of function which is being invoked. (11) (12)

In his mathematical semantic description of function invocation in SNOBOL4, Tennent goes into great detail describing the order in which appropriate variable values are saved and restored. In so doing he precisely mirrors the mechanisms of the interpreter with respect to those actions. However, it does not appear, in the extensive collection of equations which are used to model function invocation, that the problem of the number of actual parameters supplied versus the number of formal parameters declared in the function definition is handled correctly for user-defined functions. SNOBOL4 will automatically supply null strings as values to user-defined functions

---

(11) Griswold, 1972, pp. 100-103.

(12) Griswold, Poage and Polonsky, pp. 92-96.

if too few actual parameters are supplied. Correspondingly, if too many actual parameters are supplied to a user-defined function, the extraneous arguments are discarded after being evaluated. This is in contrast to builtin functions which require that the number of arguements supplied be precisely the number expected. (13)

Pattern matching is one of the most important features of SNOBOL4, and it is in patterns that we find the automatic backtracking control structure alluded to earlier. Tennent models patterns as functions which take five arguments:

a) A character string is the subject of the pattern match.

b) The cursor position acts as an index into the subject string indicating the position at which the local match is to take place.

c) Three continuations are used to specify the action to be taken if the local match is successful (the subsequent (SB)), is unsuccessful (the alternate(ALT)), or if the global match is to be aborted (the failure continuation from before).

d) Code must be provided for conditional assignments which are to be performed if the global match succeeds and this pattern has been used as a component in that successful global match.

e) The current store is needed to allow for side effects due to immediate assignments which result when a pattern is successfully used in a local match, and due to the evaluation of unevaluated expressions embedded in the pattern.

The result of this specification is a rather complex pattern domain equation:

---

(13) Griswold, 1972, pp. 103-105.

$$P = \Omega \times N \times SB \times ALT \times FL \times C \rightarrow S \rightarrow S$$

The two new domains noted above in the description of patterns can further be described by the equations:

$$SB = N \times ALT \times C \rightarrow S \rightarrow S$$

$$ALT = S \rightarrow S$$

Note that a successful local match can reposition the cursor, add conditional assignments to be performed upon the outcome of the global match, change the state of the store through immediate assignment and evaluation of unevaluated expressions and enter into consideration new alternative patterns.

There are two basic modes in which pattern matching can be undertaken in SNOBOL4, FULLSCAN and QUICKSCAN. The later mode causes the interpreter to employ heuristics based on the length of the subject string and the minimum length string required by the pattern component to successfully match. These heuristics allow the interpreter to determine if a proposed pattern match could possibly succeed, without actually trying to perform the indicated match. (14) (15) In this fashion certain alternatives can be avoided, and the global match can be terminated more quickly if it is apparent that the match cannot succeed.

If the only observable difference between these two modes of

---

(14) Griswold, 1972, pp. 126-131.

(15) Griswold, Poage and Polonsky, pp. 62-73.

operation was one of the time required to execute the pattern match, our concern over the difference would be minimal. Because of the possible use of immediate assignment operators and the side effects of evaluating unevaluated expressions, the results of using FULLSCAN mode rather than QUICKSCAN mode can be very different with respect to the contents of the store. Also the heuristics make certain assumptions about the minimum length pattern which will be matched by an unevaluated expression and can thus cause global pattern match failure when the same pattern match run under FULLSCAN mode would succeed. Hence, even conditional assignments may have different results, and the flow of control can be different depending on the mode in use at the time of the pattern match.

In the model proposed by Tennent, only the FULLSCAN mode is treated. But the primary mode used in most SNOBOL4 programs is QUICKSCAN. This represents a serious departure from the implemented language characteristics. In order to accurately model SNOBOL4 it would be necessary to provide for both modes. Since the heuristics are based on length values which are intrinsic to the distinct pattern matching functions, it would be necessary to modify each such pattern matching function defining equation to detect the current mode of operation and act accordingly. This can be accomplished through the use of two additional components in the pattern function indicating the minimum length string needed to successfully match this pattern and the number of characters remaining in the subject string . The actual current mode of operation would be determined by examining the

value associated with the keyword FULLSCAN. Keywords are discussed later in this paper. Each pattern matching function definition can then be modified to perform in the appropriate manner depending on the mode in which the system is currently operating.

In presenting the mathematical semantics for SNOBOL4, Tennent uses $\perp$ (bottom) to indicate both non-terminating computations and error conditions signalled by the interpreter. This leads to difficulties in that it does not accurately reflect the behaviour of a SNOBOL4 program. In particular, it is possible for some of the error conditions detected by the interpreter to be handled as failures in expression evaluations or global pattern match aborts if the user takes appropriate precautions. (16) This points out the need to provide far more extensive facilities in the mathematical semantics specification in order to accurately model the error handling mechanisms of the SNOBOL4 interpreter . (17)

Some of the auxiliary functions used by Tennent in providing the mathematical semantics description of SNOBOL4 tend to be rather powerful, quite unlike the usual class of primitive auxiliary functions which one expects to encounter. In particular, the suffix "as Exp" is used to indicate the parsing of a string into an

---

(16) Of the 28 execution time errors detected by the interpreter, only 12 are unconditionally fatal. The remaining 16 can be transformed into failure signals or pattern match aborts by setting the value of the keyword ERRLIMIT to a positive value. Each time one of the conditionally fatal errors occurs, the value of this keyword is decremeted by 1 (if it is > 0) and if the resulting value is > 0 the program is not terminated by the error condition.

(17) Griswold, Poage and Polonsky, pp. 179-183.

expression in support of the definition of the EVAL function, which evaluates unevaluated expressions. (18) Even more impressive is the suffix "as Prog", used to parse a string into a complete SNOBOL4 program in support of the CODE function, which converts a string into the data type code. (19)

The CODE function is obviously very powerful, and it has side effects which are not explicitly reflected in Tennent's mathematical semantic description of it. If the string being converted to code contains a label which already appears in the currently executing program, the code attribute associated with that label is modified to point to the newly generated code which has this label associated with it. Thus any subsequent transfer based on that label will result in a transfer into the new code. The old code attribute of the label is no longer in effect. (20) (21) This modification of the label attribute of a natural variable is not reflected in the defining equations for the CODE function as provided by Tennent.

One feature of SNOBOL4 which Tennent does not deal with is keywords. The unary operator "&" can be applied to certain identifiers in order to allow communication between a user program and the SNOBOL4 system. The identifiers used in this fashion are referred to as keywords and are divided into two classes protected and

---

(18) Tennent, pg. 100

(19) Tennent, pg. 104.

(20) Griswold, 1972, pp. 131-133.

(21) Griswold, Poage and Polonsky, pp. 135-138.

unprotected. The 13 unprotected keywords may have values assigned to them by the user, while the 15 protected keywords can only be examined by the user. Thus the unary operator "&" returns either a name or a value depending on its argument. (22) (23)

We have already seen how the unprotected keyword ERRLIMIT can be used to affect the way in which certain errors are handled by the system. To aid the user who has disabled the conditional error termination mechanism, the keyword ERRTYPE provides the number of the last error which occurred, thus allowing the user to selectively respond to errors (see the discussion on tracing later in this paper). We also mentioned the unprotected keyword FULLSCAN which disables pattern matching heuristics, thus turning off the default QUICKSCAN mode.

Many keywords can be modelled by adding a component to the store domain to accomodate this new attribute of some natural variables. Most keywords are simply counters or flags and thus can be easily examined or modified in other defining equations in a straightforward manner. Some keywords, however, are more closely related to the source program representation and would require additional primitive fuctions to support them. The protected keywords LASTNO and STNO contain, respectively, the listing-based statement numbers of the last statement executed and the statement currently being executed.

Input and output are performed by establishing an association

---

(22) Griswold, 1972, pp. 115-116.

(23) Griswold, Poage and Polonsky, pp. 128-130.

among a variable, a logical unit number and a FORMAT specification, through the use of the built-in functions INPUT and OUTPUT. Default associations exist for the variables INPUT, OUTPUT and PUNCH, but they may be changed by calls to DETACH. In this fashion I/O is not explicitly performed, but rather is a side effect of assigning a value to an output-associated variable or fetching the contents of an input-associated variable. Since the I/O of SNOBOL4 is defined in terms of FORTRAN IV constructs and restricted FORMAT specifications, if one could provide a suitable semantic specification for these operations in FORTRAN IV, then the SNOBOL4 I/O could be easily modelled with only the addition of store components to account for input/output associations. (24)

A major feature of SNOBOL4 not treated by Tennent is tracing. Tracing is performed by associating variables,keywords, labels and function names with tracing routines provided by the user or the system. The built-in function TRACE is used to establish these associations, and the function STOPTR is used to selectively break the associations. Global on/off control of tracing in general, and function tracing in particular, is provided through the use, respectively, of the keywords TRACE and FTRACE, which act as automatically decremented counters each time control is transferred to a trace routine. The types of tracing facilities provided are thus very flexible and very powerful.

Since user-provided functions can be invoked by the tracing

---

(24) Griswold, 1972, pp. 38-39.

system in response to a variety of actions occurring anywhere in the program, a control structure similar to PL/1 conditions and signalling is provided. (25) (26) It appears that tracing can be provided in a mathematical semantics description by adding still more components to the store domain to represent attributes indicating whether and in what fashion a variable is being traced, and associating a trace function with it (either a user-supplied trace function or the system trace function).This appears to add a substantial burden to the description since all updates to the store, all transfers of control and all function invocations need to be monitored for these associated trace attributes.

The SNOBOL4 system allocates all storage for variables, code and internal data in a linear address space in a sequential fashion. It thus becomes necessary to perform storage regeneration periodically in order to recover space occupied by data which is no longer accessible to the user. (27) If this process were initiated only by the system and had no measurable effects on the user (other than timing considerations), there would probably be no need to consider it in our semantic description. However, there is a built-in function, COLLECT, which can be called by the user to force storage regeneration, and which returns as an integer value the number of basic addressable units of storage which are available after regeneration. COLLECT can

---

(25) Griswold, 1972, pp. 135-136.

(26) Griswold, Poage and Polonsky, pp. 149-162.

(27) Griswold, 1972, pp. 142-154.

also be supplied with an argument which specifies the number of such units desired, and if the regeneration does not secure as many or more than that number of units, then the function signals failure. (28)

Since the function is a standard language feature and since it is possible to construct programs which, regardless of the units used in an actual implementation, could detect an inconsistent definition of this function, it appears necessary to make provisions for it in any semantic description of SNOBOL4. Such provisions would probably entail a substantial modification of the store domain and many of the semantic equations in order to allow for some consistent modelling of storage utilization and referenceability.

A more fundamental storage-related problem occurs with respect to the definition of the built-in functions IDENT and DIFFER. These functions accept variables of any data type and are used to determine if the arguments represent the same object. In the case of integers or real numbers the results are easily defined since the objects being compared must be in the same domain (thus IDENT(3,3.0) yields false). (29) Strings present the exceptional case here, as there is never more than one instance of a given string in storage in SNOBOL4. Thus the following statements will result in IDENT(X,Y) signalling success.

```
X = "SNOBOL"
Y = "SNO" "BOL"          (concatenation)
```

---

(28) Griswold, Poage and Polonsky, pg.195.

(29) Griswold, Poage and Polonsky, pp. 79-80.

Patterns are not treated in this special fashion, so that the following statements will result in IDENT(X,Y) signalling failure.

```
X = "ABC" | "DEF"
Y = "ABC" | "DEF"          (alternation in a pattern)
```

The following statement will, however, result in a successful evaluation of IDENT(X,Y).

```
Y = X
```

Tennent does not deal with this problem in his paper, and the functions IDENT and DIFFER are not discussed. A scheme which tags values with unique identifiers and assures the uniqueness of strings in storage is needed. Thus we will likely have to further complicate the store domain to help solve this problem.

Conclusions

Tennent's paper does provide a basis for constructing a mathematical semantics specification for SNOBOL4, as stated in his abstract. In the overview of his paper, Tennent states that he believes that the specification presented is consistent with the descriptions of SNOBOL4 given in the literature [1,3], although he does not claim to be providing an "official" definition of the language. (30) Our examination of his paper has pointed out that the

descriptions provided for function invocation and the built-in function CODE do not accurately model the language as described in current literature. The use of $\perp$ to represent all program errors also seems questionable in light of our discussion of error handling in SNOBOL4. We see that the location domain description (L) is not adequate to allow modelling of IDENT and DIFFER, although these functions were not excluded from consideration in the overview. The use of auxiliary functions which are disproportionately powerful also seems questionable.

We have provided domain equations for the remaining data types, and we have considered the problems attendant with trying to provide a mathematical semantics specification for the remainder of the language. Some of the remaining constructs can be modelled with only minor extensions to the description provided by Tennent. Other features of SNOBOL4 seem to require substantial modification and extension of the description. In many cases this seems to be the result of allowing some of the underlying implementation to be visable to the user, a pitfall which mathematical semantics specifications could help one to avoid.

---

(30) Tennent, pg. 95.

References

[1] Griswold, R., SNOBOL4 Source and Cross-Reference Listings for Version 3.6, SNOBOL Project Document S4026, Bell Telephone Laboratories, Inc., 1971.

[2] Griswold, R., A Guide to the Macro Implementation of SNOBOL4, SNOBOL Project Document S408d, Bell Telephone Laboratories, Inc., 1971.

[3] Griswold, R., The Macro Implementation of SNOBOL4, Freeman, 1972.

[4] Griswold, R., Poage, J., and Polonsky, I., The SNOBOL4 Programming Language, Prentice-Hall, (second edition) 1971.

[5] Scott, D., Outline of Mathematical Theory of Computation, Proc. 4th Annual Princeton Conf. on Information Sciences and Systems, 1970.

[6] Scott, D., and Strachey, C., Towards a Mathematical Semantics for Computer Languages, Proc. Symposium on Computers and Automata, Brooklyn, 1972.

[7] Strachey, C., The Varieties of Programming Languages, Oxford University Computing Laboratory, Technical Monograph PRG-10, 1973.

[8] Strachey, C., and Wadsworth, C., Continuations -- A Mathematical Semantics for Handling Full Jumps, Oxford University Computing Laboratory, Technical Monograph PRG-11, 1974.

[9] Tennent, R., Mathematical Semantics for SNOBOL4, Record of the ACM Symposium on Principles of Programming Languages, 1973.