technical contributions

Some Highlights of the SITBOL Language Extensions to SNOBOL4

James F. Gimpel
Bell Telephone Laboratories
Holmdel, N.J. 07733

INTRODUCTION - SITBOL [3] is an implementation of the SNOBOL4 language [5] for the DECsystem-10 and is owned and distributed by Stevens Institute of Technology (Hoboken, N. J., Attn: Leslie Maltz) from which it derives its name. It was designed for and partially constructed during a design course at Stevens (taught by the author) and was completed at Western Electric's Engineering Research Center at Princeton. SITBOL is a faster smaller alternative to the macro implementation (by a factor of 3 in each case) and has several interesting enhancements which are the subject of this paper.

SITBOL is upward compatible with both SNOBOL4 and SPITBOL [1] and, in the rare cases where it was not possible to be both (error codes for example) the choice was generally in the direction of SPITBOL. There are several interesting SPITBOL extensions within SITBOL such as BREAKX, LPAD, RPAD, REVERSE, lexical comparators, SUBSTR, SETEXIT and context-free array subscripting for which space does not permit more than this passing reference.

The intent of this paper is to describe the linguistic (and implementation) novelties of the SITBOL implementation and the reason for their inclusion in an already 'full' language. Some of these extensions relate directly to shortcomings of the macro implementation as reported by Dunn [2] and this paper tends to supplement the remarks of Hanson [6] in describing SITBOL advantages. There are also the flow-of-control difficulties mentioned by Griswold [4] and several SITBOL innovations pertain to alleviating this problem. Several of the SITBOL extensions were suggested by a number of individuals over the years and this is indicated in the Epilogue. Most of the remarks here are amplified in the user manual [3] which is available on request.

OPERATORS

Assignment Operator (=)

In SITBOL assignment is an operator (of lowest precedence); this has obvious utility. For example, to dump an array one can write

```
                I = 0
LOOP            OUTPUT =  A<I = I + 1>            :S(LOOP)
```

(we will show later how the label disappears). Associativity is to the right so that multiple assignment is easy. Thus

```
        A  =  B  =  C  =  0
```

assigns 0 to the three variables. The assignment operator returns the variable (as opposed to the value) as this is more general. For example:

```
(A = &ALPHABET) ' ' =
```

assigns to A all characters except blank. As another example

```
SUBJECT     PATTERN . (A = )
```

will assign null to A during pattern building (the absence of an expression on the right hand side of any operator is uniformly treated as denoting the null string). If the pattern succeeds, A will be assigned the string matched by PATTERN.

Pattern Matching Operator (?)

In addition to referencing pattern matching in the conventional way (by the first blank operator at the lowest level of a rule), one may use the binary question-mark operator (?) which has the second lowest precedence (next to assignment). The operator requires the subject of the match on its left and the pattern on its right. As a function it succeeds or fails depending on the success or failure of the match. The object returned may be used as variable (left-hand side) or as a value depending on context. When used as a value it returns the string matched. Thus

```
S ? ANY(&ALPHABET ? C REM)
```

will scan S for a character ≥ C (in collating sequence).

When an attempt is made to assign into the result of a pattern match, replacement results. For example

```
S ? ANY(&ALPHABET ? C REM = )
```

will scan for a character < C (in collating order). Note that it is not a sin in SITBOL for the subject of a replacement operation to be a value. The value of &ALPHABET in the above example is, of course, not modified. If the subject is a variable, replacement returns a variable so that

```
(S ? 'A' = ) ? 'B' =
```

removes from S an 'A' and then, assuming an 'A' is found, will remove a 'B'. The result of a pattern match is then a process whose parameters are

(a) the subject string

(b) the cursor positions indicating where the subject string was matched, and

(c) the variable, if any, whose value was the subject of the match.

When a value is assigned into such a process the process constructs the resultant string and assigns it to the variable (if any).

It is possible to associate the result of a pattern match with a pattern to produce truly exotic applications. For example to scan string S1 for pattern P1 and S2 for pattern P2 and if both matches succeed to swap the matched portions of both strings one can write

```
S1 ? P1 . (S2 ? P2 . X2) = X2
```

This cannot be done in SNOBOL4 with fewer then 3 pattern matches.

Inexhaustible Binary Operators

The syntax of SNOBOL4 permits any sequence of operators to be unambiguously regarded as a single binary operator. In the construction of SITBOL it was actually simpler to include this facility than omit it. Thus

```
OPSYN('=/=', 'DIFFER', 2)
...
A =/= B                      :S(DIFFERENT)
```

is valid. Precedence for all user-created binary operators is a default value.

RELATIVE (NUMERIC) GOTO

A goto of the form <expression> where the expression is numeric, as opposed to CODE, is taken to indicate that the interpreter is to march this many statements forward (backward if negative). Thus, to print the first 10 integers and their squares (to take an example from [4]) one can write:

```
        I  =  1
        OUTPUT  =  I  ' - '  I ** 2
        LE(I = I + 1, 10)    :S<-1>
```

In a relatively painless manner we avoid the two-label rendition indicated by [4]. Moreover, the predicate can be placed on the same line, yielding one less statement:

```
      I  =  1
      CUTPUT = LE(I = I + 1, 10)  I  -  I ** 2      :S<>
```

Another example of the null goto is:

```
     S   ' '  =                          :S<>
```

which will remove all blanks from the string S. The null is such a useful special case that it represents about half the uses that this writer makes of the numeric goto. The rest are mostly <-1> uses with an occasional <+2>. It is legal but foolish to use large numbers.

Note that any SNOBOL4 rule can be represented as an expression in SITBCL and that arbitrary expressions may be placed with impunity within any other expression if prefixed by a question mark (?). Thus if R_1, R_2, ..., R_n are rules then

(?R_1 ?R_2 ... ?R_n) :S()

will repeatedly execute the sequence of n rules until one of them fails, effecting the action of REPEAT(R_1, ..., R_n) suggested in [4].

<u>The Relative-Goto Controversy</u>

Most programmers, upon hearing that SITBOL has relative goto's instinctively react negatively. SITBOL users, on the other hand, tend to be highly enthusiastic. Why? I believe the negative reaction is based on

(a) it is a technique used in assembly language and

(b) it is not even a commendable technique there for the obvious reason that the relative goto does not tolerate the insertion or deletion of a statement.

While both (a) and (b) may be true, it does not necessarily follow that the relative go to is harmful in SNOBOL4. In assembly language, small relative goto's (like -1) are rare and a relative goto of 0 is meaningless. On the other hand, in SNOBOL4, the self-goto is so common that a frequent request by users of conventional SNOBOL4's is for some method of avoiding explicit labels in the self-goto cases. In SITBOL, the situation is even more compelling since the enhanced operators permit condensing many computational loops to single statements. Thus a relatively poor feature in assembly language can become a very useful feature in a higher level language precisely because of the greater expressivity of the latter.

SYSTEM-RELATED EXTENSIONS

General

An operating philosophy in designing the I/O and system related facilities was to not deny to the SITBOL user that which the system grants to the assembly programmer. In some cases this philosophy proved too awkward or difficult to implement but accounts for the inclusion of a large number of system related facilities. Writing in SITBOL, for example, the user may DELETE a file, RENAME a file, test for the existence of a file, determine the users job number, read his file directory, and run some other job. It is, for example, very straightforward (four statements) to write a program to search every file in a user's directory for the existence of some pattern and print out every line containing the match plus any sequence number. With one or two more statements, he can search every file in the system (that he has access to).

It is also possible to write a multi-pass compiler in which each pass calls in the next pass and where each pass is pre-compiled.

Word I/O vs. Character I/O

It is not the intention of this note to delve too deeply into machine-related matters but it may suffice to say that a file consists of 36-bit words and that characters are 7-bit packed 5 to a word. This means that if the user is restricted to character I/O he would forever be denied the manipulation of binary files since there would be one bit he couldn't reach. It is not reasonable to adopt the attitude that the SNOBOL4 programmer should not be interested in binary files because their manipulation is a normal part of such familiar activities as compiling into machine language, disassembly, searching binary files for certain patterns, etc. For this reason a comprehensive word I/O facility was provided wherein the user can read and write words as strings of 36 ones and zeros, as twelve octal digits, as six 6-bit characters, as five ASCII characters or as INTEGERs or REALs directly. Moreover, the conversions implied by these activities are extended to the user without the necessity of doing I/O by allowing an extra argument to the CONVERT function.

SAV Files

A problem cited by Dunn [2] was that the lack of an object module for standard SNOBOL4 necessitates recompilation each time. In

SITBOL, recompilation can be avoided by using the EXIT function. When called, EXIT will bring the user to command level. The user may then save his core image as a so-called SAV file. When the SAV file is run, the called EXIT function merely returns. SITBOL consists of 10K words pure plus a minimum of 2K words impure which grows with the program. Depending on the argument to EXIT() only the impure section need be saved. Hence copies of SITBOL need not occupy space in a user's directory.

<u>Teletype Control</u> A rather interesting feature of SITBOL is that the variable TTY is both input-associated and output-associated with the terminal. Since assignment returns a variable (the left-hand side) and associates to the right, the following statement:

```
        X  =  TTY  =  'X = '
```

will (1) prompt the terminal user with a request for the variable X, (2) read the next line typed, and (3) assign the value to X. One may argue that this statement is hard to understand, which it is, but it is also easy to use and remember and is quite useful for writing short interactive programs that operate with several parameters.

For sophisticated terminal control, the function TTY() can be used to query the state of the terminal by a running program. TTY() succeeds returning the null string if a line has been typed by the user, otherwise it fails. For example, if a program is noiselessly grinding away, it is nice for the impatient user to be able to interrupt his program to print out variables, insert TRACE commands, and in general to query the state of the program and influence its behavior. This can be done as follows. Assume a label LOOP exists and is continually being executed. Then, at that label, insert the following simple statement.

```
                             ...
LOOP                TTY  =  TTY()  EVAL(TTY)
                             ...
```

If nothing has been typed at the terminal, the statement simply fails. Otherwise it evaluates the string typed in and prints its value. The expression may have side-effects such as DUMP(2) which gives an alphabetized dump of all variables and aggregates or it may be an assignment expression which will change the program's behavior, or may request the initiation or termination of any of a variety of tracing.

<u>Time Histogram</u>

The operating system permits a running program to interrupt itself every 1/60 second to compile statistics about itself. SITBOL uses this to prepare user-oriented statistics.

To obtain in a simple fashion a time histogram profile of his program the user may set the so-called H switch in the command string. This produces as a side-effect of the run, a file which contains, alongside each statement, the number of times SITBOL interrupted the program while the given statement was being executed. Interruption during a garbage collection allots the tick to the current statement. This may seem imprecise but SITBOL tends to garbage collect quickly and often so that garbage-collect times tend to average themselves fairly rapidly.

A user may prepare his own time histogram by setting the keyword &HISTOGRAM to 1. Thereafter, until reset to 0, the value of &HISTOGRAM will be a constantly incremented array. An undocumented feature is that if &HISTOGRAM is set to -N, an absolute histogram (by storage location) is prepared. Storage is assumed subdivided into regions 2**N words large.

Statement Numbers

Some editors employ line numbers to simplify editing. If SITBOL sees that the source file is line-numbered it will use these numbers, in so far as possible, as statement numbers. This greatly facilitates editing and interpreting traces and renders listings as virtually unneeded. Even though some keywords refer to statement numbers (&STNO, &LASTNO) their primary purpose is descriptive and no difficulty ensues by allowing them to be non-sequential or even non-unique.

Line numbers tend to be 'sticky' in that updated versions of older files retain for the most part (except for deletions and insertions) older line numbers. Hence there is no need to continually regenerate new listings and for a time-sharing terminal this appears to be essential. The net conclusion of all this is quite shocking. For an interpreter like SNOBOL4 which yields diagnostic and trace information on a statement basis the only very useful identification possible seems to be a line number known also to the editor. Whereas line numbers at first sight appear to be a crude unimaginative temporary adaptation they play a remarkably effective role in associating the dynamic behavior of program execution with its static structure.

OTHER FEATURES

&ERRPARM and REPEAT

In order to provide additional diagnostic information a so-called error parameter was introduced. The error parameter is used to further identify the cause of error and is both printed as part of the error message and is available to the program via the keyword &ERRPARM. For example, all errors of the form 'illegal datatype' print the value in error.

Beyond the obvious use of analyzing errors, the facility permits planned error recovery operations. One example is in dynamically loading functions. When an undefined function is encountered the &ERRPARM is set to the name of the function. The user can trap errors (via SETEXIT [1,3]) and check the error type (via &ERRTYPE). If the error type is 'undefined function', the program can process a directory of source code searching for the missing function. Once found, it can be compiled via CODE() and control can be passed back to the statement bearing the previously undefined function via the new built-in label REPEAT.

SORT(A,N) and RSORT(A,N)

A sort facility exists in SITBOL which will sort tables, and singly and doubly dimensioned arrays. The sort will order a hetrogenous mix of different objects by datatype primarily and value secondarily. The sort is ascending if the function SORT is used. It is descending if RSORT is used. If the array is two-dimensional or is a table the sort will take a second argument (1 is default) to indicate the column on which to sort. If the array is singly-dimensioned a second argument can denote the name of a field of a programmer-defined data object which field bears the comparand of the sort. Note that if T is a table, SORT(T) has the same effect as CONVERT(T, 'ARRAY') except that the former sorts the keys.

The SORT function seems to be particularly useful in conjunction with SNOBOL4's TABLE facility. Obvious applications include sorting the cross-index listing of a compiler and preparing a KWIC index. The following is a complete program to sort and print all the identifiers (simply defined) found in the input file. The identifiers are to be sorted primarily according to frequency of occurrence (which is also printed) and secondarily by alphabetic order.

```
        S.ID = SPAN('ABCDEFGHIJKLMNOPQRSTUVWXYZ') . ID
        T = TABLE()
        X = INPUT                              :F(SORT)
        X  S.ID  =  ?(T<ID> = T<ID> + 1)       :S<>F<-1>
SORT    T = RSORT(SORT(T),2)
        OUTPUT = T<I = I + 1,1> ' ' T<I,2>     :S<>
END
```

The type of sort used was the merge sort. This was done for speed and to avoid inverting the order of equal elements so that a composite sort of the kind used in the above example would work correctly.

RANDOM(N)

A random number generator is available. RANDOM() will return a real between 0 and 1; RANDOM(N) will return an integer uniformly distributed between 1 and N. &RANDOM is a keyword specifying the generator's state. It may be set at any time to any nonnegative integer. Its initial value is 0.

Aggregate Dumping

A point raised by Dunn [2] is the general absence of aggregate-dumping in SNOBOL4. SITBOL has followed SPITBOL's lead in providing a comprehensive dump facility.

At the point of call of DUMP(N) (or at program termination if the keyword &DUMP is set to N) a dump is given if N ≥ 1. If N is 1, all natural variables are dumped in alphabetic order. If N = 2, the dump will additionally contain all aggregates, i.e., arrays, tables and programmer-defined data objects. Each aggregate is given an identification number at creation and is used to pair variable values with dumped aggregates. Also, trace messages contain the id number to identify aggregates. Finally, DUMP(A) where A is some aggregate will dump the aggregate.

EPILOGUE - To the author's knowledge the binary ? operator to represent pattern matching was first proposed by Morris Siegel in a 1968 letter to Ralph Griswold and has been sponsored enthusiastically by R. Dewar and R. Griswold in the interim. Its current form of returning the string matched as value and a complex left-hand-side as variable are original with SITBOL. A form of the relative go to in assembly language syntax (with an asterisk) was proposed to the author by Mike Radow of Columbia at a '69 Share meeting. It was only after being distressed over the compiler problems implicit in that proposal that the current simpler scheme was discovered. The SORT facility as implemented in SITBOL owes much of its richness to numerous suggestions by R. Dewar. The use of editor line numbers by SITBOL was suggested by Dick Stone. The value of time profiling has been stressed by D. Knuth [7] and has also been introduced in another SNOBOL4 system by P. Santos [8]. The facility within SITBOL was advocated strongly by Len Bosack. Many SITBOL environmental additions were suggested by enthusiastic DECsystem-10 users among whom are Dave Hanson, Larry Samberg, Don Lewine, Jeff Snitzer and probably others whose names escape me at the moment. Finally, thanks to Ed Kozdrowicki for a careful reading of this manuscript and for his many criticisms and suggestions.

REFERENCES

[1] Dewar, R. B. K. SPITBOL, Version 2.0. SNOBOL4 Document S4D23, Illinois Institute of Tech., Chicago, February 12, 1971.

[2] Dunn, R. SNOBOL4 as a language for bootstrapping a compiler. Sigplan Notices, 8:5 (May 1973), 28-32.

[3] Gimpel, J. F. SITBOL, Version 3.0. SNOBOL4 Document S4D30b, Bell Laboratories, Holmdel, N. J. June 1, 1973.

[4] Griswold, R. E. Suggested revisions and additions to the syntax and control mechanisms of SNOBOL4. Sigplan Notices, 9:2 (February 1974), 7-23.

[5] Griswold, R. E., Poage, J. and Polonsky, I. P. "The SNOBOL4 Programming Language," 2nd Ed. Prentice Hall, Englewood Cliffs, N. J.

[6] Hanson, D. R. Letter to the editor. Sigplan Notices, 8:8 (August 1973), 3-8.

[7] Knuth, D. E. An empirical study of Fortran programs. Software Theory and Practice, 1:1 (January-March 1971), 105-134.

[8] Santos, P. FASBOL, A SNOBOL4 Compiler. Memo No. ERL-M314, Electronic Research Laboratory, College of Engineering, University of California, Berkeley.