Abstract: SNOBOL4 is a dynamic string-based pattern-matching programming language published in the early 1970s. This paper describes its unique aspects including some simple examples.

SNOBOL4 is based on the original SNOBOL versions developed at Bell Laboratories in the early 1960s. It is the most widely used version of SNOBOL. *The SNOBOL4 Programming Language*, [0] by Griswold, Poage, and Polonsky, published in 1971 is the standard reference. Our textbook mentions SNOBOL briefly as an early dynamic language on pages 71-72. [1]

SNOBOL (as I will refer to the language SNOBOL4) interpreters are available for many platforms. Phil Budne [2] has a free implementation called "Macro SNOBOL4 in C". SPITBOL was a later implementation of the language. SNOBOL influenced the language called Icon but no others directly.

SNOBOL’s name describes some of its most common uses: StriNg Oriented and symBOlic Language. Strings and text manipulation are its strengths. I was surprised when talking with a librarian when he said the only computer programs he had ever written were in SNOBOL. It is wonderful for pattern matching and string manipulation.

Therefore, researchers in linguistics and the classics also use it. [3]

Consider this program that I wrote:

```plaintext
******************************************************************************
* replaces occurrences of the strings 'burg', 'ton', or 'town' with 'ville' *
* lines after end are input                                                 *
******************************************************************************
top     place = input                      :f(end)
oldForm = 'burg' | 'ton' | 'town'
newForm = 'ville'
place oldForm = newForm
output = 'New ' place        :(top)
end
millersburg
allentown
hamburger
```

Unlike many modern languages, the location of the program lines matters here. A line beginning with an asterisk is a comment and is ignored. A line starting with a plus is a continuation from the previous line. Statements may begin with labels (for example, top) or with at least one blank. The end of the program is indicated by the label end. Input
An unusual feature is the conditional transfer method of control flow based on whether the statement succeeded or failed. In the first line of the program, the \( :f(\text{end}) \) means go to the statement with the end label if the statement fails. The statement here is reading one line of input into the place variable. There are also transfers on success indicated with an s and unconditional transfers such as the one for top which creates a loop. There are no loop or conditional control statements other than these transfers.

Pattern matching is an essential part of SNOBOL. This can be done in a pattern matching statement or a replacement statement. A pattern matching statement has a subject string to be examined and a pattern to be found in the subject. The subject and pattern are separated by at least one blank, and that space between them is the pattern matching operator. For example, if

\[
\text{town} = '\text{'Millersville'}' \\
\text{doubleL} = '\text{ll}'
\]

a pattern matching statement would be

\[
\text{town} \ \text{doubleL}
\]

which could either succeed or fail. It succeeds when it matches the ll after the Mi.

The program above has a more complicated pattern of various town endings, separated by the vertical bar OR operator, stored in the oldForm variable. The statement

\[
\text{place} \ \text{oldForm} = \text{newForm}
\]

is a replacement statement where the first (starting from the left) part of the subject that matches the pattern is replaced by the object following the equal sign. Therefore, for each line of input, the program replaces the first occurrence of the town endings stored in oldForm with the string in newForm. If the pattern fails to match, the subject doesn’t change and the statement fails.

SNOBOL has built-in functions as well as programmer-defined functions. The actual parameters are evaluated, and the values are copied into the formal parameters. Therefore, pass-by-value is normal. However, pass-by-reference can be achieved [0, p. 132]. Omitting an actual parameter puts a null string into the formal parameter, and it is an
error to provide too many actual parameters. If a function needs the parameter to have a certain type, an implicit type conversion will be attempted.

Functions are defined using a define statement that gives the name of the function and lists its formal parameters (but not their types). It may also list the local variables for the function and give a label entry point for where the function's code begins. The code should be out of the way of the normal program execution. If the function's entry point label is omitted, it is assumed to be the function name. Values can be returned from functions.

Scoping is dynamic. Therefore, a function has access to the referencing environment of any function that called it. Listing local variables in the function definition ensures that you don't unintentionally change a non-local variable with the same name. Changing a variable's value may also change its type. The types are tied to the values rather than the variable's name.

Consider this solution to the Towers of Hanoi problem given on page 110 of [0]:

* Towers of Hanoi from page 110 of Griswold
* HANOI function moves N disks from NS (source) to ND (destination) using NI (temporary)

```
DEFINE('HANOI(N,NS,ND,NI)', 'HANOI')          :(HANOI.END)
HANOI   EQ(N, 0)                                      :S(RETURN)
HANOI(N - 1, NS, NI, ND)
OUTPUT = 'Move disk ' N ' from ' NS ' to ' ND
HANOI(N - 1, NI, ND, NS)                      :(RETURN)
HANOI.END
```

```
TEST    HANOI(4, 'left', 'right', 'middle')
END
```

When I first I typed this program into a file named hanoi.sno, I used lower case. I then compiled and executed it (on cs) with the command (this will work if snobol4 is installed on cs; it currently isn't):

```
snobol4 -l < hanoi.sno
```

I was reminded that spaces and upper and lower case are very important in SNOBOL. There can be no spaces in the parameter list in the define statement. There must be spaces around the minus sign in \( n - 1 \). My lower case hanoi label in the define statement didn't match the lowercase hanoi label at the beginning of my function because the compiler converted the latter to uppercase. This made me appreciate the flexibility we have in many languages for spacing and using upper and lower case without the translator making them all upper case. This conversion can be turned off and is a leftover from the days of keypunches and having only upper case.
There is no support for abstract data types or classes. The language provides integers, reals, strings, patterns, arrays, and tables. Programmer-defined data types seem to be somewhat like records in other languages because there are fields. The book [0] defines a complex number data type on page 124. There are examples of linked list nodes on page 19.

Originally, the SNOBOL compiler converted the source code to a Polish-prefix object code. That object code would be executed during execution. The version from [2] uses C macros in its translation. Everything is allocated dynamically and automatic garbage collection is done. However, the programmer can use the `collect` function to force garbage collection.

There are more programming language concepts to mention than there is space in this short summary. I include them here without discussion.

Arrays are bounds checked but the failure merely causes the statement to fail. The program can use that failure to transfer control. The lower bound is normally 1 but can be specified as in this two-dimensional array that has indexes from -10 to 10 for the first index and 5 to 25 as the second index with all its elements set to the value of x

```plaintext
board = array('-10:10,5:25',x)
```

One of its elements could be referenced with

```plaintext
board<3,15>
```

Unlike more traditional languages, array elements can take on any value. Therefore, one element could be an integer and others could be strings or patterns. They can change type when given a new value.

Tables are pairs of associated data objects. Think of them as one-dimensional arrays indexed by any data object. You're not limited to just integers or characters as indexes.

The code function takes in a string representing SNOBOL statements and converts it into object code that can then be executed. This allows the programmer to generate code during execution that becomes part of the executing program.
SNOBOL is a complex language to learn because it is unlike any other language in both syntax and semantics. It forces you to approach problem solutions from a different perspective. This summary touched on only some of its features. The book [0] has sample programs that produce sets of bridge hands, compute and print a table of $n$ factorial using numbers that are too large for the computer, syntactically recognize SNOBOL4 statements, and "sing" the song The Twelve Days of Christmas. It is truly a different language.

References:


