A PROCEDURE FOR CREATING TABLE LOOK-UP FUNCTIONS FROM SAS DATASETS

A. D. Forbes, British Rail

Introduction

Many SAS applications make use of codes to store values of variables and often it is convenient to have the look-up table for translating the codes in the form of a SAS dataset. The traditional way of looking up codes in SAS datasets is to write a SAS program that converts the dataset to PROC FORMAT statements and ultimately generates a value format. The format can then be used as the second operand in a PUT function to carry out the table look-up. However, the limit on the size of a value format is quite restrictive making it impossible to handle large tables and even in the cases where this method does work satisfactorily we feel that it has an air of clumsiness about it. In this paper we shall present a more efficient method based on a new SAS procedure - FUNCTN.

A SAS dataset with variables V1, V 2, ..., Vn and Vr defines an n-variable function provided that the values of (V1, V2, ..., Vn) are not duplicated. The function maps values of the ARGUMENT (independent) variables (V1, V2, ..., Vn) to values of the RESULT (dependent) variable Vr.

The FUNCTN procedure uses the SAS dataset to generate a SAS function to perform the mapping by a table look-up algorithm. PROC FUNCTN takes the SAS dataset, compiles the corresponding table look-up function and places it on a specified load library. Once it has been created, the function is independent of the SAS dataset and can be used like any standard SAS function as described in the chapter 'SAS function' of the SAS User's Guide: Basics.

To illustrate how the procedure works, let us consider a simple example. Suppose that you have a SAS dataset of location numbers and names which was created with the following statements

```
DATA LOCATION;  INPUT LOCNUM LOCNAME L;  CARDS;
 1018 WATERLOO
 1099 VICTORIA
 1023 EUSTON
  ...    ...
PROC SORT;  BY LOCNUM;
```

You can convert this into a function that takes a location number as its argument and returns the location name as a character value. The SAS statements to create this function and give it the name LOCFNC are

```
PROC FUNCTN NAME=LOCFNC DATA=LOCATION;
     ID  LOCNAME;
     VAR LOCNUM;
```

What you get is a table look-up function that consists of each LOCNUM - LOCNAME pair occurring in the SAS dataset. The load module containing the executable code is placed on the temporary LIBRARY dataset with the member name LOCFNC and can be used in any SAS DATA step following the PROC FUNCTN statement. When the function is invoked with a location number as argument, the value returned is the corresponding location name as in the original SAS dataset. For example the statement

```
Z = LOCFNC(1023);
```

appearing anywhere within a DATA step will give Z the character value 'EUSTON'.

There is a facility to create functions on permanent libraries that works in much the same way as user defined SAS formats. This will be useful if you want to build up a collection of table look-up functions for your SAS applications.

The rest of this paper describes the FUNCTN procedure in detail and is divided into four sections. The SPECIFICATION and DETAILS sections follow the SAS documentation style of presentation and describe how to use the procedure. The next section, PERFORMANCE ANALYSIS, compares FUNCTN with the use of SAS formats for table look-up operations and shows that it is an efficient function generator which can compile very fast algorithms. The final section tells you how to install the FUNCTN procedure from the SUCCESS library tape.

SPECIFICATION

The FUNCTN procedure is controlled by the following statements:

```
          PROC FUNCTN options;
               ID variable;
               VAR variables;
```

PROC FUNCTN Statement

```
  PROC FUNCTN options;
```

The following options may be used in the PROC FUNCTN statement:

NAME=name	This is the name of the function to be created. Must be 1 to 8 letters or numbers, the first of which is a letter. If you are creating a function on a permanent load library, then make sure that the name does not match with any other SAS function of format names.
DATA=SASdataset	Specifies the name of the SAS dataset from which the function is to be created. Standard SAS dataset naming conventions apply. Observations must be sorted by the variables in the VAR list and must be duplicate-free in these variable. The TYPE parameter is optional; the default is _LAST_.
DDNAME=name	Specifies the DDname of a load library in which to store the function. If not given, the function is stored on LIBRARY, which refers to a temporary data set. If you want your function stored permanently for later use, then set up a library and refer to it with this option. See the section 'Creating Temporary and Permanent Formats' of the FORMAT Procedure in the SAS User's Guide: Basics. Functions are similar to formats and existing format libraries can be used to store functions as well. Please note that either CMS nor VSE are supported by the FUNCTN procedure.
TYPE=number	Gives you a choice of table lookup algorithms. TYPE=1 specifies Binary Search. TYPE=2 specifies that the result is accessed directly from the argument variable. TYPE=3 specifies that the result is accessed from the argument variable via an index. For TYPE=2 and TYPE=3, there must be only one argument variable which is numeric and takes integer values. This parameter is optional; the default is TYPE=1.

DETAILS

Table Lookup Algorithms

If you do not specify a TYPE parameter in the PROC FUNCTN statement, then the procedure assumes TYPE=1 and will generate a function that uses a binary search to locate the argument vector in a table which contains a vector of argument and result values for each observation in the SAS dataset. The average time taken to evaluate the function is approximately proportional to the logarithm of the number of observations.

For single variable integer argument functions, greater efficiency is possible. If the span between the minimum and maximum argument values is not too large, you can have the function use the argument value directly as a relative pointer in a table of result values. This is TYPE=2 in the PROC FUNCTN statement and is very fast, the function's performance being independent of the SAS dataset size.

TYPE=3 (also for single variable integer argument functions) is similar. The difference is that the look up in the table of result values goes via an index of relative pointers which is accessed directly using the argument value. Again, the function's performance is independent of the SAS dataset size but somewhat inferior to TYPE=2 because of the extra index. For sparse tables TYPE=3 is usually more efficient on space than TYPE=2, but you should check the storage estimates given later in this section.

Missing Values

The generated function returns a missing value of the appropriate type (numeric or character) if the argument vector is not found in the table.

There is nothing special about missing values occuring in the SAS dataset. Observations containing argument or result variables with missing values are included in the function just the same as ordinary observations.

Non-Integer Arguments

If you are creating a binary search function with one or more arguments that take non-integer numeric then be careful. You may have problems with values that are not multiples of powers of 0.5 due to the finite precision with which recurring binary decimals can be represented.

If your argument values are really simple fractions then it might be better if you convert them to integers. Multiply by the common denominator and use the ROUND function.

Sort Sequence

The SAS dataset is assumed to be sorted in strictly increasing order of the argument variables, using the EBCDIC collating sequence for character variables. If it is not, then usually you will get an error message. However, PROC FUNCTN checks up to only 8 characters and it is possible to create a binary search function that contains unsorted data. This will show up later when you try to use the function which will sometimes return missing value results for arguments that are not in their correct sequence in the table.

You can use PROC SORT with the NODUP option to sort and eliminate duplicates from a SAS dataset prior to function generation.

Storage Requirements

The amount of main storage required by the generated function can be estimated using the formulae given below. In each case the answer is an upper bound in bytes. As you can see, the lengths of the variables make a significant contribution. If storage is a problem (as with very large SAS datasets) you might find it necessary to recreate the SAS dataset with the variable lengths reduced to the minimum.

```
TYPE=1:  n*(a + r) + 800,
TYPE=2:  s*r + 800,
TYPE=3:  n*r + s*int(log2(n)/8 + 1) + 800,
```

where

```
a = sum of lengths of the argument variables,
r = length of the result variable,
n = number of observations in the SAS dataset,
s = u - l + 1,
l = minimum value of argument 1,
u = maximum value of argument 1,
```

PERFORMANCE ANALYSIS

Comparison between FUNCTN and FORMAT

It is instructive to compare the performance of FUNCTN with that of FORMAT when they are both used in the same recording application. For simplicity we consider only a single variable, integer to integer function but there is no reason to suppose that the comparative results will differ significantly in the case of non-integer numeric or character variables.

A SAS dataset of 2500 observations is created with the statements

```
    DATA; DO A=1 TO 2500;  Z=10*A;  OUTPUT;  END;
```

Then the FUNCTN procedure is run against it

```
    PROC FUNCTN NAME=TENFNC;  ID Z;  VAR A;
```

to generate a function called TENFNC which returns an integer value. Actually, the function maps x - 10x for x = 1, 2, ..., 2500 and x - MISSING otherwise; this fact is irrelevant in what follows. TENFNC is to be compared with a format - TENFMT - which does essentially the same thing. TENFMT Is generated by the following SAS code.

```
     DATA _NULL_;  SET;  FILE SASINC;
        IF A=1 THEN PUT ' PROC FORMAT; VALUE TENFMT ';
        PUT A '=' Z;
        IF A=2500 THEN PUT ' OTHER=.; ';
     DATA _NULL_;
     %INC SASINC;
```

It was found that creation of the format TENFMT required 5.73 seconds while PROC FUNCTN took only 0.29 seconds. In a similar test with 50000 observations we had insufficient main storage for PROC FORMAT to work while PROC FUNCTN generated the corresponding function in 4.85 seconds.

Since TENFNC and TENFMT both use binary search table look-up algorithms, we would expect them to perform similarly. This was tested by executing the function and the format 2500 times each.

```
     DATA _NULL_;
       DO A=1 TO 2500;  X=TENFNC(A);  END;
```

Time taken = 0.07 seconds.

```
     DATA _NULL_;  LENGTH X L5;
       DO A=1 TO 2500;  X=PUT(A,TENFMT5.);  END;
```

Time taken = 0.08 seconds.

Comparison between FUNCTN Types 1, 2 and 3

Since direct look-up is much faster that binary search, we would expect a TYPE=2 or TYPE=3 algorithm to perform significantly better than a TYPE=1. This is confirmed by executing a SAS program which creates a 50000 observation SAS dataset, generates one function of each type and then executes each function 50000 times.

```
     DATA;  LENGTH A Z 4;
            DO A=1 TO 50000;  Z=10*A;  OUTPUT;  END;

     PROC FUNCTN NAME=TENFN1 TYPE=1;  ID Z;  VAR A;
     (Time taken : 4.85 seconds)

     PROC FUNCTN NAME=TENFN2 TYPE=2;  ID Z;  VAR A;
     (Time taken : 4.41 seconds)

     PROC FUNCTN NAME=TENFN3 TYPE=3;  ID Z;  VAR A;
     (Time taken : 5.52 seconds)

     DATA _NULL_;  DO A=1 TO 50000;  X=TENFN1(A);  END;
     (Time taken : 1.61 seconds)

     DATA _NULL_;  DO A=1 TO 50000;  X=TENFN2(A);  END;
     (Time taken : 0.36 seconds)

     DATA _NULL_;  DO A=1 TO 50000;  X=TENFN3(A);  END;
     (Time taken : 0.38 seconds)
```

Summary of results

The results presented above are collected together in a table.

FUNCTN or FORMAT	FUNCTN type	Number of values	Creation time (seconds)	Mean execution time (microseconds)
FORMAT	-	2500	5.73	32
FUNCTN	1	2500	0.29	28
FUNCTN	1	50000	4.85	32
FUNCTN	2	50000	4.41	7
FUNCTN	3	50000	5.52	8

All timings relate to SAS 82.3 running under MVS on a NAS 9000.

INSTALLATION INSTRUCTIONS : 'SUCCESS' LIBRARY

The FUNCTN procedure is available free to SAS users in Europe through the SUCCESS library. To find out how to obtain a copy, you should contact your SAS representative.

When you unload your SUCCESS library tape you will find four members associated with the product :

SAS.SUCCESS.LIBRARY(SO2501RL) : Load module for the FUNCTN parser.
Copy to your SAS load library and change the member name to FUNCTN.

SAS.SUCCESS.LIBRARY(SO2501PL) : Load module for the FUNCTN procedure.
Copy to your SAS load library and change the member name to FUNCTN2.

SAS.SUCCESS.SAMPLE(SO2501ES) : 80-byte card image SAS statements for testing.

SAS.SUCCESS.SOURCE(SO2501D) : Documentation. Similar to what you are reading now.
It can be printed with the following JCL.

```
//GEN    EXEC PGM=IEBGENER
//SYSPRINT DD DUMMY
//SYSIN    DD DUMMY
//SYSUT2   DD SYSOUT=A,UCS=TN,
//            DCB=(RECFM=VBA,LRECL=84,BLKSIZE=6144)
//SYSUT1   DD SAS.SUCCESS.SOURCE(SO2501D),DISP=SHR
```

Further information and support can be obtained from the author at

British Railways Board,
Blandford House Computer Centre,
Melbury Terrace, London NW1, England.

Telephone 01-262 3232 ext 5564.