PharmaSUG 2013 - Paper CC04

SAS® Logic Coding Made Easy – Revisit User-defined Function

Songtao Jiang, Boston Scientific Corporation, Marlborough, MA

ABSTRACT

SAS programmers deal with programming logics on a daily basis. Using regular logical operators can solve a complex logic problem, but oftentimes it is not intuitive, error-prone, and hard to read and maintain. Implementation also heavily depends on how well the individual understands the logics and how good the individual programmer's logical thinking is. In this paper, by using SAS User-defined Function, we lay out a universal way of implementing SAS logics without the dependence of individual's ability and experience. The implementation is straight-forward, error-prove, and clear. Furthermore, these SAS user-defined functions for logical operators can be built into the standard library and utilized by all programmers within an organization. They are extremely useful to all levels of SAS programmers. To demonstrate the idea, the implementations of these functions are introduced. Two sets of the sample codes of a moderately complex logic are compared between regular logic implementation and User-defined Function.

INTRODUCTION

SAS programmers deal with programming logics on a daily basis. It is very important and sometimes critical that a programmer can effectively and correctly program the provided logics in clinical settings. In the traditional way, programmers use the if-then-else statement and the SAS logical operators and to implement these logics. For simple logics, it is effective and easy to use. However for more complex logics, things become more tricky and difficult to implement. And also the basic operators may generate unexpected results when they deal with variables with missing values.

Let's review how the basic logical operators work. We have three basic logical operators: AND, OR, and NOT. The following table (Table 1) lists their operation characteristics:

Variables		Logical Operators*		
a	**b**	**a AND b**	**a OR b**	**NOT a**
.	.	0	0	1
.	0	0	0	1
.	1	0	1	1
0	.	0	0	1
0	0	0	0	1
0	1	0	1	1
1	.	0	1	0
1	0	0	1	0
1	1	1	1	0
* In logical expression, SAS treats missing or 0 as false and all other value as true.				

Table 1: Basic logical operators and results

From Table 1, we can see that 0 or missing value is treated as false in a logic expression, and 1 or other value is treated as true. In this paper, we only discuss 0, 1, and missing values. Any other values should be converted to one of these three values before putting into a logic expression.

LOGICAL EXPRESSION AND ISSUES

In clinical settings, oftentimes missing value should be treated as missing instead of false. For example, we need to create a "Group" variable from 2 binary variables – "Male" and "Diabetes". We need to identify male subjects with diabetes as one group, the rest subjects as another group. For group 1, it is easy to determine. If a subject is a male with diabetes, then Group = 1. However for group 2, it is a little bit tricky given the presence of missing values of 2 binary variables - "Male" and "Diabetes". In the following, I give 2 wrong solutions and one correct solution, and result comparison table (Table 2).

Solution 1:

```
DATA Demo1;
    SET Baseline;
    IF Male AND Diabetes THEN Group=1;
    ELSE IF Missing(Male) OR Missing(Diabetes) THEN Group=.;
    ELSE Group=2;
RUN;
```

Solution 2:

```
DATA Demo2;
    SET Baseline;
    IF Male AND Diabetes THEN Group=1;
    ELSE IF (NOT Male) OR (NOT Diabetes) THEN Group=2;
RUN;
```

Solution 3 (correct solution):

```
DATA Demo3;
    SET Baseline;
    IF Male AND Diabetes THEN Group=1;
    ELSE IF (Male EQ 0) OR (Diabetes EQ 0) THEN Group=2;
RUN;
```

Variables		Solution 1	Solution 2	Solution 3
Male	**Diabetes**	**Group**	**Group**	**Group**
.	.	.	2	.
.	0	.	2	2
.	1	.	2	.
0	.	.	2	2
0	0	2	2	2
0	1	2	2	2
1	.	.	2	.
1	0	2	2	2
1	1	1	1	1

Table 2: Results Comparison between 3 Solutions

For solution 1, a few cases are missed to be identified as group 2. This is because one of these two binary variables is missing. But in some cases even one variable is missing, we can still determine which group the subject should belong to. For the subject with “Male=.” and “Diabetes=0”, this subject is definitely not in Group 1 (“Male” and “Diabetes”) no matter what the missing value would be for variable “Male”. So this subject should belong to group 2. In another case for subject with “Male=.” and Diabetes=1”, the “Group” should be missing because the results depend on value of the variable “Male”.

For solution 2, all subjects are grouped into two groups. This is also not what we wanted. The issue is related to the NOT logical operator. As we mentioned, all logical operators treated missing value as false. Using NOT logical is not appropriate in this case even though the SAS code is very close to what the correct solution is.

For solution 3, we are able to give the correct solution with the help of comparison operator “EQ” instead of using the logical operator “NOT” as in solution 2.

From the example, we can see that a very simple logic task can get us into a tricky situation. If we are given more variables or more complex logics, things are getting a lot worse. Even a well experience programmer may feel that it is not easy to handle.

To address this common issue among all levels of SAS programmers, a very simple implementation using SAS User-defined Function is proposed. It is simple, flexible, error-prove, and very effective.

USER-DEFINED FUNCTION (UDF) FOR LOGIC EXPRESSION

LOGICAL OPERATOR UDFS IMPLEMENTATION

Use SAS Function Compiler procedure to create the User-defined functions for the logical operators. For the UDFs, we need to treat missing value as missing. And the results depend on the situations listing in the following table (Table 3):

Parameters		UDF	
a	**b**	**a AND b**	**a OR b**
.	.	.	.
.	0	0	.
.	1	.	1
0	.	0	.
0	0	0	0
0	1	0	1
1	.	.	1
1	0	0	1
1	1	1	1

Table 3: UDF definitions and Results

The UDF implementations based on Table 3 are given below:

```
PROC FCMP OUTLIB=sasuser.myUDF.logic;

    FUNCTION logicand(a,b);
           IF a=. AND b=. THEN logicand=.;
           IF a=. AND b=0 THEN logicand=0;
           IF a=. AND b=1 THEN logicand=.;
           IF a=0 AND b=. THEN logicand=0;
           IF a=0 AND b=0 THEN logicand=0;
           IF a=0 AND b=1 THEN logicand=0;
           IF a=1 AND b=. THEN logicand=.;
           IF a=1 AND b=0 THEN logicand=0;
           IF a=1 AND b=1 THEN logicand=1;
           RETURN(logicand);
    ENDSUB;
    FUNCTION logicor(a,b);
           IF a=. AND b=. THEN logicor=.;
           IF a=. AND b=0 THEN logicor=.;
           IF a=. AND b=1 THEN logicor=1;
           IF a=0 AND b=. THEN logicor=.;
           IF a=0 AND b=0 THEN logicor=0;
           IF a=0 AND b=1 THEN logicor=1;
           IF a=1 AND b=. THEN logicor=1;
           IF a=1 AND b=0 THEN logicor=1;
           IF a=1 AND b=1 THEN logicor=1;
           RETURN (logicor);
    ENDSUB;
run;
```

SOLUTION FOR THE EXAMPLE USING UDF

Now we have defined the UDFs for the logical operators and are ready to use them to solve our earlier simple example.

```
options cmplib=sasuser.myUDF; /* Locate the User-define UDFs */
DATA DemoUDF;
    SET Baseline;
```

```
    Group=2-logicand(Male,Diabetes);
RUN;
```

One may doubt that the UDF just lists all possible outcomes, and it is not very convincing for simplifying a logic expression. In the following section, I will give a more complex example. I will compare the implementations between using regular logical operators and using the UDFs. Then you will see more clearly the power of using UDFs to solve logic problems.

COMPARISON BETWEEN TWO METHODS

The following example is from a pacemaker device to determining the physiologic settings. The logic need to be implemented is as following:

(Pacing Mode is AAIR) OR ((Pacing Mode is DDDR) AND ((AV Search = ON) OR (AV min > PR)))

Pacing Mode AAIR, Pacing Mode DDDR, AV Search = ON, and AV min > PR are four binary variables with values (0, 1, or missing).

SAS code used for the both methods:

```
DATA DemoCMP;
    SET CRM;

    /* Use UDF */
    UDF_Logic=logicor(AAIR, logicand(DDDR, logicor(AVSearch, AVGTPR)));

    /* Use regular logic expression */
    if AAIR=1 or (DDDR=1 and (AVSearch=1 or AVGTPR=1)) then Reg_Logic=1;
    else if AAIR=0 and (DDDR=0 or (AVSearch=0 and AVGTPR=0)) then Reg_Logic=0;
RUN;
```

For the UDF method, it is very straight-forward. You can use the UDFs as regular functions, and just follow the given logic specifications to write the SAS code. This is can be easily implemented by all levels of programmers.

For the regular logic expression method, depending on your experiences and logical thinking skill, different programmers may give different answers. The given SAS code above by far is the simplest solution that I can get with a paid price that I worked at background using the Set Operations. For *Reg_logic = 1*, it is straight-forward. The following are the derivations for *Reg_logic = 0*:

- ⇨ $Reg\_Logic = 0$
- ⇨ $AAIR \cup (DDDR \cap (AVSearch \cup AVGTPR)) = 0$
- ⇨ $\overline{AAIR \cup (DDDR \cap (AVSearch \cup AVGTPR))} = 1$
- ⇨ $\overline{AAIR} \cap \overline{(DDDR \cap (AVSearch \cup AVGTPR))} = 1$
- ⇨ $\overline{AAIR} = 1$ and $\overline{(DDDR \cap (AVSearch \cup AVGTPR))} = 1$
- ⇨ $\boldsymbol{AAIR = 0}$
  AND
  $\overline{(DDDR \cap (AVSearch \cup AVGTPR))} = 1$
  - ⇨ $\overline{DDDR} \cup \overline{(AVSearch \cup AVGTPR)} = 1$
  - ⇨ $\overline{DDDR} = 1$ or $\overline{(AVSearch \cup AVGTPR)} = 1$
  - ⇨ $\boldsymbol{DDDR = 0}$
    OR
    - ⇨ $\overline{(AVSearch \cup AVGTPR)} = 1$
    - ⇨ $\overline{AVSearch} \cap \overline{AVGTPR} = 1$
    - ⇨ $\overline{AVSearch} = 1$ and $\overline{AVGTPR} = 1$
    - ⇨ $\boldsymbol{AVSearch = 0}$
      AND
      $\boldsymbol{AVGTPR = 0}$

In summary of above derivations, the logic expression should be:

AAIR=0 AND (DDDR=0 OR (AVSearch=0 AND AVGTPR=0))

The following table (Display 1) lists partial dataset and output:

	Pacing Mode - AAIR (Yes/No/Missing - 1/0/.)	Pacing Mode - DDDR (Yes/No/Missing - 1/0/.)	AV Search = ON (Yes/No/Missing - 1/0/.)	AV min > PR (Yes/No/Missing - 1/0/.)	UDF_Logic	Reg_Logic
1	0	0	.	1	0	0
2	.	.	0	.	.	.
3	0	1	1	1	1	1
4	0	0	1	0	0	0
5	.	0	1	0	.	.
6	0	0	1	0	0	0
7	1	0	1	.	1	1
8	1	0	0	0	1	1
9	.	1	0	.	.	.
10	0	1	.	0	.	.
11	0	1	.	.	.	.
12	.	0	1	1	.	.
13	0	0	0	0	0	0
14	1	.	.	.	1	1
15	.	1	0	.	.	.
16	0	0	.	1	0	0
17	0	.	0	0	0	0
18	1	.	.	0	1	1
19	.	.	0	.	.	.
20	.	.	1	1	.	.
21	0	1	0	0	0	0
22	0	.	0	0	0	0
23	1	1	0	0	1	1
24	0	0	1	0	0	0
25	0	1	1	0	1	1
26	.	1	.	.	.	.
27	0	1	0	1	1	1
28	1	0	0	.	1	1
29	0	.	.	0	.	.
30	1	.	1	1	1	1
31	.	1	1	0	1	1
32	0	.	.	.	.	.
33	.	1	0	.	.	.
34	1	0	0	1	1	1
35	.	1	0	0	.	.
36	1	1	0	0	1	1
37	0	1	.	1	1	1
38	1	.	0	1	1	1
39	.	1	0	0	.	.
40	.	.	1	1	.	.

Display 1: Partial dataset and output for the physiologic settings of the pacemaker device

CONCLUSION

The SAS language has provided SAS programmers a way to define a User-defined Function. Using UDFs for SAS logic programming has demonstrated a lot of advantages over other methods. The solution is ideally simple, easy to implement, error-prove, and easy to maintain.

REFERENCES

"Base SAS(R) 9.2 Procedures Guide." SAS 9.2 Documentation. Copyright © 2013 SAS Institute Inc. Available at http://support.sas.com/documentation/cdl/en/proc/61895/HTML/default/viewer.htm#a002890483.htm

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Name: Songtao Jiang
Enterprise: Boston Scientific Corporation
Address: 50 Boston Scientific Way
City, State ZIP: Marlborough, MA 01752
Work Phone: 508-683-4432
Fax: 508-683-5642
E-mail: JIANGS@BSCI.com
Web: http://www.bostonscientific.com/us/index.html