PharmaSUG2011 – Paper CC21

Permutation via Recursive SAS® Macro

Jian Dai, Clinovo, Sunnyvale, CA

ABSTRACT

In this paper, we will demonstrate how to implement a recursive algorithm with SAS macro that generates the permutations of a list of distinguished elements. We will also present two recursive utility macros used to manipulate lists. One of them implements For-Each loop; the other returns the complement of an element in a given list.

INTRODUCTION

Recursion is at the heart of computer science; however, why recursion in SAS? For starters, define a *family tree* relation as an SQL table

```
Create Table FAMILYTREE (
        Father          num,
        Mother          num,
        Child           num
);
```

Given any individual in this relation, how to query the earliest forefather? Check out this recursive macro:

```
%macro ForeFather(Id);
        %local Tmp;
        proc sql noprint;
                select Father into :Tmp from FAMILYTREE where Child = &Id;
        quit;
        %if &sqlobs %then %do;
                %ForeFather(&Tmp)
        %end;
        %else %do;
                %global FFather;
                %let FFather=&Id;
        %end;
%mend;
```

SAS Macro supports fully fledged recursive invocation [1-5]; however, this feature is not duly appreciated (for example, homemade runtime stack is coded as a workaround to implement recursion [6, 7]).

The purpose of this paper is threefold. First to demonstrate the elegancy and terseness of recursion in the context of SAS Macro, second to increase awareness and promote the application of recursive macros in everyday tasks and, last but not least, to explore the power, richness and potential of SAS Macro.

To achieve this goal, we will use in this paper a well-understood example: permutation of a list [8]. This choice is based on three criteria: i) The algorithm is well-formulated such that the focus is on the SAS implementation; ii) The algorithm is patently typical so the gist of recursion is efficiently conveyed; iii) The algorithm is not overwhelmingly sophisticated such that the methodology is able to be grasped with any level of programming experience.

INVITATION OF RECURSION AND PERMUTATION

Recursion within the domain of computation is the invocation of a function, a subroutine or a macro by itself. Recursion is an efficient problem-solving method that breaks down a complicated problem into several easier problems; at certain point during the course of reduction, the initial problem is retrogressed to simple problems with existing solutions.

For starters, the factorial of a positive integer n, which is mathematically written as n!, is recursively defined as n times the factorial of (n-1)! if n is larger than 1, else 1. The following code is the SAS Macro implementation of this well-known example:

```
%macro Factorial(n);
       %* Assume &n=1,2,3,4,... ;
       %if &n=1 %then 1;
       %else %eval(&n*%Factorial(%eval(&n-1)));
%mend;
```

A permutation of n distinct objects is an assignment of order of these objects. For example, all permutations of numbers 1, 2, 3 can be represented as

1 2 3,
1 3 2,
2 1 3,
2 3 1,
3 1 2,
3 2 1.

From the combinatorial point of view, factorial of n is the number of all permutations composed of these n objects. In light of this interpretation, the algorithm to compute n! is interpreted as to factorize n! to the counting of the distinct ways to choose one from n objects, which equals n, times the counting of permutations of the rest (n-1) objects, that is (n-1)!. So it is quite natural to move from there to a full-blown generalization that generates these permutations.

In the following section, we describe the detailed algorithm for permutations, and the recursive SAS macro implementation.

TOP-DOWN DESIGN OF THE ALGORITHM

In this paper, a list is defined as a series of words delimited by white spaces. A word is understood as a string containing only alphanumeric characters (the rigid definition of a word is neither relevant nor important though). i.e. we require all words used in this paper to be "atomic"; in other words, we do not consider quoted string as a word. As we are dealing with permutation, we assume each word in the list is unique.

Let L be a list of distinct words and x be any of the words in L. All permutations of L can be recursively defined as:

i) If L is null, i.e. no word contained, then the permutation of L is also null;
ii) Else the permutations of L are the concatenations of each word x and the permutations of the complement of x in L.

Complement is a set-theory term. The complement of x in L is the resulting list by removing x from L.

In terms of pseudo code, the above algorithm is expressed as:

```
%PermutationOf &List:=
       %If &List^= %Then %Do;
              %ForEach &word In &List
                     &word %PermutationOf (%ComplementOfWord &word In &List);
       %end;
```

In practice, to implement this algorithm

a) A buffer parameter is needed to store the already-selected order.
b) The termination condition that the list L is null must be explicitly written out so that when each permutation is set by the termination of recursion, the macro can take some action to that permutation.
c) Two sub-macros %ForEach and %ComplementOfWord are to be defined.

As a top-down design, we detailed below the algorithms for the two macros described above:

For Each

For-Each loop is a versatile generalization of the more mundane for-loop over a sequence of numbers: For-Each loops through a list of words. The algorithm is a typical tail-recursion: If the list is not null then

i) Execute the designated task for the first element;
ii) Execute the designated task for the rest elements.

Written in pseudo code:

```
%ForEach &Item In &List Do &Something :=
  %if &List^= %then %do;
    Do &Something to %TheFirstWordOf &List;
    %ForEach &Item In %ExceptTheFirstWord_SubListOf &List,Do &Something
  %end;
```

which implies that two submacros are to be defined: %TheFirstWordOf and %ExceptTheFirstWord_SubListOf. Actually, they are the most important two list manipulation utilities [9].

Complement

The complement of a word in a list is the sub-list of the original list that excludes that word. The algorithm is also a tail-recursion: If the list is not null then

i) If the first word is to be excluded then return the complement in the sub-list but the first word;
ii) If the first word is not to be excluded then return the concatenation of the first word and the complement in the sub-list but the first word.

Written in pseudo code:

```
%ComplementOfWord &w In &List :=
  %if &List^= %then %do;
    %if %TheFirstWordOf &List Is &w %then
       %ComplementOfWord &w In %ExceptTheFirstWord_SubListOf &List;
    %else
       %TheFirstWordOf &List
       %ComplementOfWord &w In %ExceptTheFirstWord_SubListOf &List;
  %end;
```

In fact, if the list contains only distinct words then the first part of the algorithm can be simplified as

i') If the first word is to be excluded then return the sub-list but the first word;

MACRO CODE

Now we detail the implementation of above-described algorithms.

First we write the macro to return the first word in a list by calling the macro function SCAN:

```
%macro TheFirstWordOf(List=);
       %scan(&List,1)
%mend;
```

The sublist of a list without the first word is returned from the following macro by calling the macro function SUBSTR:

```
%macro ExceptTheFirstWord_SubListOf(List=);
       %local L; %let L=%length(%TheFirstWordOf(List=&List));
       %if &L<%length(&List) %then %left(%substr(&List,%eval(1+&L)));
%mend;
```

With these two utility macros in hand, we can easily write down ForEach macro:

```
%macro ForEach(ItemRef,In=,Do=%nrstr(/* Nothing */));
       %if %left(%trim(&In))^= %then %do;
              %let &ItemRef=%TheFirstWordOf(List=&In);
              %unquote(&Do);

       %ForEach(&ItemRef,In=%ExceptTheFirstWord_SubListOf(List=&In),Do=&Do)
       %end;
%mend;
```

It is noteworthy that the parameter ItemRef is a so-called soft-reference to another macro variable, i.e. the value of this parameter is interpreted as the name of another macro variable. The value of ItemRef serves as the loop variable. The value of parameter &In is a list through which &ItemRef loops. Moreover, parameter &DO contains a segment of macro code, which in principle contains the loop variable.

With TheFirstWordOf and ExceptTheFirstWord_SubListOf we can easily write down the code for ComplementOfWord:

```
%macro ComplementOfWord(w,In=);
```

```
	%local _1stWord;
	%if &In^= %then %do;
		%let _1stWord=%TheFirstWordOf(List=&In);
		%if &_1stWord=&w %then
			%ComplementOfWord(&w,
				In=%ExceptTheFirstWord_SubListOf(List=&In));
		%else &_1stWord %ComplementOfWord(&w,
			In=%ExceptTheFirstWord_SubListOf(List=&In));
	%end;
%mend;
```

Now our main macro:

```
%macro PermutationOf(
	List,
	SelectedPath=,
	Execute=%nrstr(%put &SelectedPath;)
);
	%If &List= %Then
		%unquote(&Execute) ;
	%Else %do;
		%ForEach(word, In=&List,
			Do=%nrstr(
				%PermutationOf(
					%ComplementOfWord(&word,In=&List),
					SelectedPath=&SelectedPath &word,
					Execute=&Execute
				)
			)
		)
	%end;
%mend;
```

Here the parameter &SelectedPath is the buffer parameter we mentioned in the algorithm design. Each time the recursion terminates, macro code segment &Execute is executed by unquoting.

SAMPLE INVOCATION

- Do nothing:
  ```
  %PermutationOf(1 2 3 4,Execute=%nrstr())
  ```
- Do the default (output to Log file):
  ```
  %PermutationOf(1 2 3 4)
  ```
- Do the default plus count the number of permutation:
  ```
  %let c=0;
  %PermutationOf(1 2 3 4,Execute=%nrstr(%let c=%eval(&c+1);%put &c:
&SelectedPath;))
  ```
- Output the permutations to a SAS dataset for n=5:
  ```
  data Perm5;
  	%PermutationOf(1 2 3 4 5,Execute=%nrstr(
  	perm="&Path &L";output;))
  run;
  ```
- Output the permutations to a SAS dataset for n=8:
  ```
  data Perm8;
  	%PermutationOf(1 2 3 4 5 6 7 8,Execute=%nrstr(
  	perm="&Path &L";output;))
  run;
  ```

The excerpt from the log to create dataset Perm8 is given below, from which you can see the O(N!) nature of this algorithm.

```
NOTE: The data set WORK.PERM8 has 40320 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time           21:24.93
      cpu time            21:06.15
```

CONCLUSION

We have shown recursive program can be written in SAS macro as in any modern programming language that supports recursion. Although this paper is not a comprehensive exploration of recursive algorithm by using SAS Macro, we demonstrate the power of recursive macros using permutation which is a detailed and well-documented example. Besides, For-Each loop is also implemented via recursive macros.

Note that the permutation is a special case of the tree structure. In the future we will explore the application of SAS Macro to address more general non-linear data structures with the weapon of recursion under our disposal. Also note that i) starting from Version 9.2, recursive function can be defined by user via PROC FCMP [10]; ii) for permutation per se, there exist at least two solutions in SAS: PROC PLAN and CALL ALLPERM; iii) another Macro solution to permutation can be found in a SAS technote [11]; iv) the subtlety of using %NRSTR and %UNQUOTE is carefully studied in Whitlock's SUGI article [12].

REFERENCES

1 John H. Adams, The power of recursive SAS® macros - How can a simple macro do so much? http://www2.sas.com/proceedings/sugi28/087-28.pdf
2 John Zheng, Recursive SAS Macro: A Fun Application http://support.sas.com/resources/papers/proceedings10/206-2010.pdf
3 Houliang Li, The Pegboard Game: A Recursive SAS® Macro Solution http://www.nesug.org/proceedings/nesug06/ap/ap03.pdf
4 Jim Fang, DrSearch: Recursive SAS® Macro solution for understanding complete program dependencies in large SAS® reporting system (LSRS) http://www.lexjansen.com/pharmasug/2007/cc/cc03.pdf
5 Ian Whitlock, Recursion in SAS Macro http://www.sugme.org/meeting_files/20100402/whitlock_recursion.pdf
6 William E. Benjamin, Jr., A Pseudo-Recursive SAS® Macro ftp://ftp.sas.com/techsup/download/observations/obswww18/obswww18.pdf
7 Stephen Rhoades, Recursion? SAS? Let's Fake It! http://www.nesug.org/proceedings/nesug01/ps/ps8013.pdf
8 William Ford and William Topp, Data structure with C++, 1996 Prentice Hall
9 CAR and CDR http://en.wikipedia.org/wiki/CAR_and_CDR
10 The FCMP Procedure http://support.sas.com/documentation/cdl/en/proc/61895/HTML/default/viewer.htm#a002890483.htm
11 TS-498: Generating Combinations and Permutations http://support.sas.com/techsup/technote/ts498.html
12 Ian Whitlock, A Serious Look Macro Quoting http://www2.sas.com/proceedings/sugi28/011-28.pdf

ACKNOWLEDGEMENTS

I appreciate the review and feedback on the manuscript from Sophie McCallum, Gulan Zhang and Yu Shu.

CONTACT INFORMATION

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

Jian Dai
Clinovo
1208 E. Arques Avenue, #114
Sunnyvale, CA 95085
E-mail: jian@clinovo.com
Web: http://www.clinovo.com/