PharmaSUG 2013 - Paper CC23

Extending the PRX Functions with PROC FCMP

Kunal Agnihotri, PPD, Inc., Morrisville, NC
Kenneth W. Borowiak, PPD, Inc., Morrisville, NC

ABSTRACT

Beginning with SAS® Version 9, users have been afforded the ability to perform complex string matching, replacement, and extraction with Perl regular expressions via the PRX functions and call routines. We present two user-written functions with PROC FCMP which extend the functionality of PRX. The first function (UPRXMATCH) augments the PRXMATCH function with two arguments, one which allows the users to control which position in the string to begin the match and the other to control whether the beginning or ending location of the string is returned. The second function (UPRXPOSN) eases the use of PRXPOSN by allowing the user to extract a capture buffer from a source in single call, as opposed to a sequence of calls to PRXPARSE, PRXMATCH and PRXPOSN.

INTRODUCTION

Beginning with SAS Version 9, users have been afforded the ability to perform complex string matching, replacement, and extraction with Perl regular expressions via the PRX functions and call routines. We present two user-written functions with PROC FCMP which extend the functionality of PRX. The **F**unction **C**o**MP**iler procedure has been available since Version 9.0, but had a limited scope. In SAS V9.2, user-written functions and call routines were available anywhere where system-defined functions are used. The extensions to PRX functions were implemented using PROC FCMP over creating a macro for multiple reasons; the primary of these being that functions created using PROC FCMP are usable in a variety of places including: PROC SQL, WHERE statements, data set options, and in the macro facility with %SYSFUNC.

The first user-written function UPRXMATCH is an extension of the PRXMATCH function by augmenting it with two additional arguments. The first argument allows the users to control which position in the string to begin the match and the other to control whether the beginning or ending location of the string is returned. The second user-written function UPRXPOSN consolidates the multi-step requirements for extracting matched characters using PRXPOSN into a single call by eliminating the explicit calls to PRXPARSE and PRXMATCH.

Some familiarity with regular expressions (aka regex) and the PRX functions is assumed. Those unfamiliar with them should refer to Borowiak [2008]. Secosky [2007] and Adams [2010] are excellent references on PROC FCMP.

UPRXMATCH

The PRXMATCH function finds a match for the regex starting from the first position of the source string and returns the position of the source where the match is found. What if the user wants to do a regex match from a given point in the source string? Or what if the user is interested in knowing the ending position of the match in the source string? The user-written UPRXMATCH created with PROC FCMP can help the user achieve this by adding two additional arguments to the syntax of PRXMATCH. This user-written function mimics the REGEXP_INSTR function available to Oracle users, which is discussed in Agnihotri and Borowiak [2012]. After adding the two arguments, this is how the syntax of the new function looks like:

UPRXMATCH (pattern , source string , starting position , end position)

- pattern - Pattern to be matched. It can not accept a compiled regular expression from a previous call to PRXPARSE.
- source string - Source variable or string where the pattern is to be searched.
- starting position - Specifies what position to begin the search in the source string.

- end position - Controls whether it either returns the beginning or ending position of the match. A value of 1 or higher returns where the match ends in a given source string.

The full code for the definition of the UPRXMATCH function can be found in Appendix 1, but the workings of the algorithm is as follows:

- If the third parameter has a value of 0 or 1 and the fourth parameter is a missing value or 0 then return the value of executing PRXMATCH using the first two parameters.
- If the third argument of the function is greater than 1 then create a temporary variable SOURCE2 that substrings the source variable beginning at the position specified in the third argument.
    - If the fourth argument is missing or 0 then return the value of executing PRXMATCH using the first argument and the temporary variable SOURCE2.
- If the fourth argument is greater than one then augment the regular expression in the first argument in a capture buffer with a pair of parenthesis (). The PRXPOSN function is then used to extract the first capture buffer and length of the capture buffer is found. The length is added to the starting position of the match from the PRXMATCH function and the result is returned.

Note that PROC FCMP does not allow optional arguments without first declaring them outside of the user-written function call, such as in an array in a DATA step. Since one of the motivating factors of creating these user-written functions was to use them outside of a DATA step, all four of the arguments in UPRXMATCH are required.

Display 1 illustrates the use of UPRXMATCH function in a PROC SQL step. The newly created variable MATCH1 returns the ending position of the match of a letter `A` through `E` at the beginning of the NAME variable followed a vowel using a case-insensitive search. The return of the ending position is dictated by using a value of 1 in the fourth argument. The variable MATCH2 uses the PRXMATCH function to find a similar match, but it only returns the starting position of the match. Output is shown in Table 1.

Display 1: Return the End Position of a Match

```
proc sql ;
  create table class1 as
  select   *
         , uprxmatch ( 'm/^[A-E][aeiou]/io' , name, 0 , 1) as match1
       , prxmatch ( 'm/^[A-E][aeiou]/io' , name) as match2
  from     sashelp.class( obs=5 )
  ;
quit ;
```

Table 1: Return the End Position of a Match

Name	match1	match2
Alfred	0	0
Alice	0	0
Barbara	2	1
Carol	2	1
Henry	0	0

Let us look at another example where the third argument uses a number greater than 1. This time it will be used in a DATA step to perform a case-insensitive search for the letter **R**. The newly created variable MATCH1 matches the letter **R** in the variable NAME but begins the search at position 3 in the string. The value of the fourth argument is 1 which returns where the regex match ends in the source variable. PRXMATCH creates the variable MATCH2 which again *only* returns the starting position of the match and does not have the feature of defining which position to initiate the search.

Display 2: Begin the Pattern Match Search at the Third Byte of the Variable

```
data _null_;
    set sashelp.class ;
    match1 = uprxmatch( '/R/io', name, 3, 1 ) ;
       match2 =  prxmatch( '/R/io', name ) ;
run;
```

Partial output is shown below:

Table 2: Begin the Pattern Match Search at the Third Byte of the Variable

Name	match1	match2
Alfred	5	4
Alice	0	0
Barbara	4	3
Carol	4	3
Henry	5	4
Jeffrey	6	5
Mary	4	3
Philip	0	0
Robert	6	1
Ronald	0	1

The output above shows the difference in the UPRXMATCH and PRXMATCH calls. An example to differentiate the results of the UPRXMATCH function versus PRXMATCH is shown at the observation where name equals **Robert** in the above output. This name has two R s , one at the first position and the other at the fifth. PRXMATCH finds a match at the very first position, whereas UPRXMATCH ignores the first **R** in the string as it was instructed by the third argument to initiate the search only at position 3. Hence it gives out a value of 6 which is where the second **R** in the string *ends*. If the value of the STARTING POSITION argument were to be 0 or 1 in the above example call, then the value of MATCH1 would have been 2.

UPRXPOSN

The PRXPOSN function gives the user the ability to extract a capture buffer used in a prior call to PRXMATCH. The vexing aspect of this function is that a prior call to PRXMATCH is a prerequisite and that the first argument to PRXPOSN has to be the same as used in the call to PRXMATCH. In order to sync these two function calls they both need to use a compiled result of a call to PRXPARSE. The multi-step approach to extracting a capture buffer is demonstrated in Figure 3, where the second vowel in the variable NAME is sought.

Display 3: Multi-Step Approach to Extracting a Capture Buffer

```
data foo ;
    set sashelp.class ( obs=6 ) ;
    length vowel2 $1 ;

    /* Compile the regex */
    re1 = prxparse( 'm/^[^aeiou]*([aeiou])[^aeiou]*([aeiou])/io' ) ;

    /* Search for a match */
    start_pos = prxmatch( re1, name ) ;

    /* Extract the second capture buffer */
    vowel2 = prxposn( re1, 2, name ) ;
run ;
```

Output is shown below.

Table 3: Multi-Step Approach to Extracting a Capture Buffer

Name	re	start_pos	vowel2
Alfred	1	1	e
Alice	1	1	i
Barbara	1	1	a
Carol	1	1	o
Henry	1	0	
James	1	1	a

The user-written function UPRXPOSN bundles the three steps into one convenient step. The function definition is presented in Display 4. The UPRXPOSN function takes three arguments, just as in PRXPOSN, but the main difference is that the uncompiled regular expression is entered directly in the first argument.

UPRXPOSN (pattern , buffer, source string)

- pattern - Pattern to be matched. It can not accept a compiled regular expression from a previous call to PRXPARSE.
- buffer - Number of the capture buffer to return.
- source string - Source variable or string where the pattern is to be searched.

Display 4: Definition of the UPRXPOSN Function

```
proc fcmp outlib=work.funcs.uprx;

 /* Define the function which results in a character field */
 /* The function takes three arguments, where the 1st and 3rd are character */

    function uprxposn( re $, buffer, source $ ) $   ;
            /* Compile the regex */
           ___re=prxparse( re ) ;
           /* Peform the match using the compiled regex */
             ___match=prxmatch( ___re, source ) ;
             /* Extract the capture buffer */
           length ___buffer $2000 ;
           ___buffer=prxposn( ___re, buffer, source ) ;
           return( ___buffer ) ;
    endsub ;
run;
```

Display 5: Using the UPRXPOSN Function with the Macro Facility

The example in Display 5 below demonstrates using the UPRXPOSN function to succinctly extract a capture into a macro variable[1]. Similar to the example in Figure 3, the second vowel will be extracted from the string Alfred .

```
%let vowel2=
  %sysfunc( uprxposn(m/^[^aeiou]*([aeiou])[^aeiou]*([aeiou])/i, 2, Alfred) ) ;

%put Second Vowel-&vowel. ;

Second Vowel-e
```

CONCLUSION

The efficiency of the PRX functions to match or extract regular expressions can be improved by using the PROC FCMP procedure. The UPRXMATCH function discussed above gives the user more control over the pattern match by not only defining the location to begin the search but also to give out where the search ends in the given source. The UPRXPOSN function makes full use of PROC FCMP by bringing together functionalities from PRXMATCH, PRXPOSN and PRXPARSE into one unique function thereby eliminating multiple calls of the PRX functions to extract a given pattern from a source string. These user defined functions work seamlessly across DATA steps, PROCs or in SAS macros. The user is encouraged to experiment with the logic presented in this paper and apply it to another PRX function - PRXCHANGE (Borowiak [2012]) and enjoy better control over pattern substitution.

[1] If running this example in SAS V9.2 then the following HotFix needs to be applied: http://support.sas.com/kb/40/737.html

REFERENCES

Adams, John H. (2010) "The new SAS 9.2 FCMP Procedure, what functions are in your future?". Proceedings of the Pharmaceutical Industry SAS Users Group Conference, USA.
www.lexjansen.com/pharmasug/2010/ad/ad02.pdf

Agnihotri, Kunal and Kenneth W. Borowiak (2012) "A SAS Users Guide to Regular Expressions When the Data Resides in Oracle". Proceedings of the Twentieth Annual Southeast SAS Users Group Conference, USA.
http://analytics.ncsu.edu/sesug/2012/PO-07.pdf

Borowiak, Kenneth W. (2012), "PRXCHANGE: Accept No Substitutions". Proceedings of the Twentieth Annual Southeast SAS Users Group Conference, USA.
http://analytics.ncsu.edu/sesug/2012/CT-03.pdf

Borowiak, Kenneth W. (2008), "PRX Functions and Call Routines: There is Hardly Anything Regular About Them!". Proceedings of the Twenty First Annual Northeast SAS Users Group Conference, USA.
http://www.nesug.org/proceedings/nesug08/bb/bb11.pdf

Friedl, Jeffrey E.F., *Mastering Regular Expressions 3rd Edition*

Secosky, Jason (2007), "User-Written DATA Step Functions". SAS Global Forum 2007
http://www2.sas.com/proceedings/forum2007/008-2007.pdf

ACKNOWLEDGEMENTS

The authors would like to thank Jenni Borowiak and Jim Worley for their insightful comments on this paper and Kim Sturgen for her encouragement and support for submitting the paper.

DISCLAIMER

The content of this paper are the works of the authors and do not necessarily represent the opinions, recommendations, or practices of PPD, Inc.

CONTACT INFORMATION

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

Name: Kunal Agnihotri
Enterprise: PPD, Inc.,
Address: 3900 Paramount Parkway
Morrisville NC 27560
Email: **kunal.agnihotri@ppdi.com**
agnihotrikunal@gmail.com

Name: Ken Borowiak
Enterprise: PPD, Inc.,
Address: 3900 Paramount Parkway
Morrisville NC 27560
Email: **ken.borowiak@ppdi.com**
ken.borowiak@gmail.com

APPENDIX 1 - Definition of UPRXMATCH

```
options cmplib = work.funcs ;

proc fcmp outlib=work.funcs.uprx ;

  /* Define the function which results in a numeric field */
  /* The function takes four arguments, where the 1st and 2nd args are character */
  /* If the 4th argument is an integer greater than 0 then the end position of the
match
    is returned. If the argument is 0 then return the beginning position of the
match. */

   function uprxmatch( __x_re $, source $, start, end_pos )  ;

   length ___re1 $1000 source2 $32767 ;

    if start in ( 0, 1 ) and end_pos in ( ., 0 ) then do ;  /* comma needed in the
IN operator */
      ___re=prxparse( __x_re  ) ;
      ___match=prxmatch( ___re, source ) ;
     return( ___match ) ;
     end ;

   else if start>1 and end_pos in ( ., 0 ) then do ;
       source2=substr(source,start) ; /* If you don't give it a new name it will
                                overwrite the SOURCE var in the output data set */
         ___match=prxmatch( prxparse(__x_re ), source2 ) ;
        if ___match>0 then ___match=___match++start-1 ;
       return( ___match ) ;
   end ;

  else if start in ( 0, 1 ) and end_pos not in ( ., 0 ) then do ;
    * Put an open parens before the reg ex to create a new, first capture buffer ;
                    ___re1=strip( prxchange( "s/^m?\//m\/(/io", -1, __x_re  ) );
    * Put a close parens before the regex to end the new,first capture buffer ;
                   ___re2=strip( prxchange( "s/(?<!\/)(?=\/[iox]*\s*$)/)/io", -1,
___re1 ) ) ;
                   ___re=prxparse( ___re2 ) ;
                  ___match=prxmatch( ___re, source ) ;
                  first_capture_buffer=prxposn( ___re, 1, source ) ;
     ___end_match=___match+length( first_capture_buffer)-1 ;
     return( ___end_match ) ;
      end ;

  else if start not in ( 0, 1 ) and end_pos not in ( ., 0 ) then do ;
    * Put an open parens before the reg ex to create a new, first capture buffer ;
           ___re1=strip( prxchange( "s/^m?\//m\/(/io", -1, __x_re  ) );
           ___re2=strip( prxchange( "s/(?<!\/)(?=\/[iox]*\s*$)/)/io", -1,
___re1 ) ) ;
                  ___re=prxparse( ___re2 ) ;
                        source2=substr(source,start) ;
                  ___match=prxmatch( ___re, source2 ) ;
                  first_capture_buffer=prxposn( ___re, 1, source ) ;
   if ___match>0 then ___end_match=___match+length( first_capture_buffer)+start-1 ;
        else ___end_match=0 ;
     return( ___end_match ) ;
      end ;

 endsub ;

 run ;
```