Paper CT12

Understanding regular expression and its application in SAS using PRX (PERL regEx) functions

Jagadish Katam, Parexel International, Uxbridge, United Kingdom

ABSTRACT

SAS already has a powerful set of string functions which are sufficient to carry out pattern matches and text mining. But sometimes regular expressions are more efficient to deal with complicated string manipulation tasks and for reading highly unstructured data streams. For example, you may have different text used for reported adverse events in a data file and you want to extract all of the AEs which are reported by a specific investigator. Once a pattern is recognized, we can identify the position of the pattern, extract a substring, or substitute a string. Also, we can use regular expressions for many day to day utilities for example, performing error/warning quick checks and running many files in one go. Perl regular expressions greatly enhance the power of the SAS language and is worth exploring.

INTRODUCTION

A regular expression (regEx for short) is a special text string for describing a search pattern. Regular expressions are extremely useful in extracting information from any text by searching for one or more matches of a specific search pattern. Fields of application range from validation of data, replacing text and extracting a substring from a string. In this paper we will cover different PRX functions in SAS that use the Perl regular expressions which are more efficient compared with regular SAS search functions like INDEXW, FINDW or string extraction functions SCAN, SUBSTR.

RegEx consist of letters, numbers, metacharacters, and special characters which form patterns. For SAS to properly interpret these patterns, all regEx values must be encapsulated by delimiter pairs identified by the forward slash, /, throughout the text (refer to the examples in this paper). They act as the container for our patterns. So, all regEx patterns that we create will look something like this: /pattern/. If you would like to use any other delimiter, then forward slash / could be replaced by # as a delimiter. In this paper we can see the use of # as delimiter in example 7.

PERL REGULAR EXPRESSION BASICS

Regular expressions are a pattern language which provides fast tools for parsing large amounts of text. Regular expressions are composed of characters and special characters that are called metacharacters.

GENERAL CONSTRUCTS

Metacharacter	Description
( )	indicates grouping.
non-metacharacter	matches a character.
{ } [ ] ( ) ^ $. | * + ? \	characters with a special function; precede them with \ if you want to match literally
\	overrides the next metacharacter.

BASIC PERL METACHARACTERS

The following table lists the metacharacters that you can use to match patterns in Perl regular expressions.

Metacharacter	Description
\b	matches a word boundary (the position between a word and a space): "er\b" matches the "er" in "never" "er\b" does not match the "er" in "verb"
\d	matches a digit character that is equivalent to [0−9].
\D	matches a non-digit character that is equivalent to [^0−9].
\s	matches any white space character, including space, tab, form feed, and so on, and is equivalent to [\f\n\r\t\v].
\S	matches any character that is not a white space character and is equivalent to

	[^\f\n\r\t\v].
\w	matches any “word” character, i.e. alphanumeric or underscore
\W	matches any non-word character or non-alphanumeric character and excludes the underscore.

REPETITION FACTORS

Perl regular expressions support repetition factors. A repetition factor matches a preceding subexpression zero or more times as it can when using a specific starting location.

Metacharacter	Description
*	matches the preceding subexpression zero or more times: zo* matches "z" and "zoo" * is equivalent to {0,}
+	matches the preceding subexpression one or more times: "zo+" matches "zo" and "zoo" "zo+" does not match "z" + is equivalent to {1,}
?	matches the preceding subexpression zero or one time: "do(es)?" matches the "do" in "do" or "does" ? is equivalent to {0,1}
{n}	matches preceding subexpression exactly n times.
{n,}	matches preceding subexpression at least n times.
{n,m}	m and n are non-negative integers, where n<=m. They match at least n and at most m times: "o{1,3}" matches the first three o's in "fooooood" "o{0,1}" is equivalent to "o?" You cannot put a space between the comma and the numbers.

CLASS GROUPINGS

By placing part of a regular expression inside round brackets or parentheses, you can group that part of the regular expression together. This allows you to apply a quantifier to the entire group or to restrict alternation to part of the regEx.

Metacharacter	Description
[...]	specifies a character set that matches any one of the enclosed characters: “[abc]” matches the “a” in “plain”
[^...]	specifies a character set that matches any character that is not enclosed within [...]: “[^abc]” matches the “p” in “plain”
[a-z]	specifies a range of characters that matches any character in the range: “[a-z]” matches any lowercase alphabetic character in the range “a” through “z”
[^a-z]	specifies a range of characters that does not match any character in the range: "[^a-z]" matches any character that is not in the range "a" through "z"
[0-9]	specifies a range of digits that matches any digit in the range: “[0-9]” matches any digit in the range 0 through 9
[^0-9]	specifies a range of digits that matches any digit in the range: "[^0-9]" does not matches any digit that is in the range 0 through 9

SAS PERL REGULAR EXPRESSION FUNCTIONS AND THEIR SYNTAX

PRX FUNCTIONS

PRXPARSE, PRXMATCH, PRXCHANGE, CALL PRXSUBSTR, PRXNEXT

SYNTAX
PRXPARSE (perl-regular-expression)
PRXMATCH (regular-expression-id | perl-regular-expression, source)
PRXCHANGE (perl-regular-expression | regular-expression-id, times, source)
CALL PRXSUBSTR (regular-expression-id, source, position <, length>)
CALL PRXNEXT (regular-expression-id, start, stop, source, position, length)

ARGUMENTS
regular-expression-id specifies a numeric variable with a value that is the identification number that is returned by the PRXPARSE function.

perl-regular-expression specifies a character constant, variable, or expression with a value that is a Perl regular expression.

source specifies a character constant, variable, or expression that you want to search.

times is a numeric constant, variable, or expression that specifies the number of times to search for a match and replace a matching pattern. If the value of times is –1, then matching patterns continue to be replaced until the end of source is reached.

start is a numeric variable that specifies the position at which to start the pattern matching in source. If the match is successful, CALL PRXNEXT returns a value of position + MAX(1, length). If the match is not successful, the value of start is not changed.

stop is a numeric constant, variable, or expression that specifies the last character to use in source. If stop is -1, then the last character is the last non-blank character in source.

source specifies a character constant, variable, or expression that you want to search.

position is a numeric variable with a returned value that is the position in source at which the pattern begins. If no match is found, it returns zero.

length is a numeric variable with a returned value that is the length of the string that is matched by the pattern. If no match is found, it returns zero.

PRXPARSE and PRXMATCH

The PRXPARSE function returns a pattern identifier number or pattern that is used by other Perl functions and CALL routines to match patterns. If an error occurs in parsing the regular expression, then SAS returns a missing value. PRXPARSE uses metacharacters in constructing a Perl regular expression.

The PRXMATCH functions searches for a pattern match and returns the position at which the pattern is found.

Sample data:

```
data conmed;
input subject$ cmdecod:$100.;
cards;
001 CARBAMAZEPINE
001 DOXORUBICIN
001 CARBAZINE
001 VINCRISTINE
002 PILSICAINIDE
002 PHENOBARBITAL
002 BUTALBITAL
002 RIFAP
run;
```

Some of the approaches to use the PRXPARSE with PRXMATCH

1) With (IF _N_ =1) condition to retain the PATTERN variable and then passing the PATTERN_ID into PRXPARSE. Metacharacter /i ignores the case and 'm' tag at the beginning of the search string tells PRXMATCH that it is doing a matching operation.

Example 1:

```
data want;
 set conmed;
  retain pattern_id;
  if _n_=1 then
  pattern_id ='m/CARBAMAZEPINE|PHENOBARBITAL|BUTALBITAL|RIFAP/i';
  pattern=prxparse(pattern_id);
  position=prxmatch(pattern_id,cmdecod);
run;
```

Output:

SUBJECT	CMDECOD	PATTERN	PATTERN_ID	POSITION
001	CARBAMAZEPINE	m/CARBAMAZEPINE|PHENOBARBITAL|BUTALBITAL|RIFAP/i	1	1
001	DOXORUBICIN	m/CARBAMAZEPINE|PHENOBARBITAL|BUTALBITAL|RIFAP/i	2	0
001	CARBAZINE	m/CARBAMAZEPINE|PHENOBARBITAL|BUTALBITAL|RIFAP/i	3	0
001	VINCRISTINE	m/CARBAMAZEPINE|PHENOBARBITAL|BUTALBITAL|RIFAP/i	4	0
002	PILSICAINIDE	m/CARBAMAZEPINE|PHENOBARBITAL|BUTALBITAL|RIFAP/i	5	0
002	PHENOBARBITAL	m/CARBAMAZEPINE|PHENOBARBITAL|BUTALBITAL|RIFAP/i	6	1
002	BUTALBITAL	m/CARBAMAZEPINE|PHENOBARBITAL|BUTALBITAL|RIFAP/i	7	1
002	RIFAP	m/CARBAMAZEPINE|PHENOBARBITAL|BUTALBITAL|RIFAP/i	8	1

2) With (IF _N_ =1) condition without creating the PATTERN variable and using PRXPARSE.

Example 2:

```
data want;
 set conmed;
  retain pattern_id;
  if _n_=1 then
  pattern_id=prxparse('m/CARBAMAZEPINE|PHENOBARBITAL|BUTALBITAL|RIFAP/i');
  position=prxmatch(pattern_id,cmdecod);
run;
```

Output:

SUBJECT	CMDECOD	PATTERN_ID	POSITION
001	CARBAMAZEPINE	1	1
001	DOXORUBICIN	1	0
001	CARBAZINE	1	0
001	VINCRISTINE	1	0
002	PILSICAINIDE	1	0
002	PHENOBARBITAL	1	1
002	BUTALBITAL	1	1
002	RIFAP	1	1

3) Without (IF _N_ =1) condition and using the PRXPARSE. If Perl regular expression is a constant or if it uses the /o option, the Perl regular expression is compiled only once. Successive calls to PRXPARSE will not cause a recompile but will return the regular-expression-id for the regular expression that was already compiled. This behavior simplifies the code by avoiding the use of an initialization block (IF _N_ =1) to initialize Perl regular expressions.

Example 3:

```
data want;
 set conmed;
 pattern_id=prxparse('m/CARBAMAZEPINE|PHENOBARBITAL|BUTALBITAL|RIFAP/oi');
 position=prxmatch(pattern_id,cmdecod);
run;
```

Output:

SUBJECT	CMDECOD	PATTERN_ID	POSITION
001	CARBAMAZEPINE	1	1
001	DOXORUBICIN	1	0
001	CARBAZINE	1	0
001	VINCRISTINE	1	0
002	PILSICAINIDE	1	0
002	PHENOBARBITAL	1	1
002	BUTALBITAL	1	1
002	RIFAP	1	1

4) The same result can be achieved without using PRXPARSE and PATTERN_ID by directly using the PRXMATCH as below

Example 4:

```
data want;
 Set conmed;
 position=prxmatch('m/CARBAMAZEPINE|PHENOBARBITAL|BUTALBITAL|RIFAP/oi',cmdecod);
run;
```

Output:

SUBJECT	CMDECOD	POSITION
001	CARBAMAZEPINE	1
001	DOXORUBICIN	0
001	CARBAZINE	0
001	VINCRISTINE	0
002	PILSICAINIDE	0
002	PHENOBARBITAL	1
002	BUTALBITAL	1
002	RIFAP	1

COMPARISON OF PRXMATCH WITH INDEXW AND FINDW

The advantages of using the PRXMATCH when compared with regular SAS functions INDEXW and FINDW are: When searching for multiple words like CARBAMAZEPINE, PHENOBARBITAL, BUTALBITAL or RIFAP then we must use INDEXW for each word like below and moreover, it is case sensitive and cannot make case insensitive but searches the complete word.

```
if indexw(cmdecod,'CARBAMAZEPINE') or indexw(cmdecod,'PHENOBARBITAL') or
indexw(cmdecod,'BUTALBITAL') or indexw(cmdecod,'RIFAP') then flag=1;
```

Like INDEXW we can use FINDW for every word, however FINDW has an advantage over INDEXW i.e., it has an option modifier 'i' to ignore the case. But still it needs to be repeated for each word like below.

```
if findw(cmdecod,'CARBAMAZEPINE',' ','I') or findw(cmdecod,'PHENOBARBITAL',' ','i') or
findw(cmdecod,'BUTALBITAL',' ','i') or findw(cmdecod,'RIFAP',' ','i') then flag=1;
```

Comparing the SAS functions INDEXW and FINDW with PRXMATCH, a multiple line code could be reduced to a single line. In PRXMATCH, the multiple words could be separated by '|' pipe symbol, 'i' make it case insensitive, 'o' will retain the pattern to all the lines for search. Similar code of PRXMATCH could be applied in performing error/warning checks on log files where we can place any ERROR, WARNING, or any NOTE in the PRXMATCH separated by 'I'.

Example 5:

```
data want;
 set conmed;
 if prxmatch('m/CARBAMAZEPINE|PHENOBARBITAL|BUTALBITAL|RIFAP/oi',cmdecod) then flag=1;
run;
```

We can also search for digits using the PRXMATCH, in the below example we can search the string with dose

information and flag it. The metacharacter '\d' helps in recognizing any digit in the string.

Sample data:

```
data conmed;
input subject$ cmdecod&:$100.;
cards;
001 RIFAP 5 mg
001 OXCARBAZEPINE 1000 mg
002 CARBAMAZEPINE
002 PHENYTOIN 62.5 mg
run;
```

Example 6:

```
data want;
 set conmed;
 if prxmatch('m/\d/o',cmdecod) then flag=1;
run;
```

Output:

SUBJECT	CMDECOD	FLAG
001	RIFAP 5 mg	1
001	OXCARBAZEPINE 1000 mg	1
002	CARBAMAZEPINE	.
002	PHENYTOIN 62.5 mg	1

PRXCHANGE

Performs a pattern matching replacement. Let's see how we can use the PRXCHANGE to replace the UN and UNK with '--' and '---'.

s – is a tag in prxchange that represents substitution
( ) - indicates grouping
\s – matches a single space
\d - matches any digit
* - matches preceding subexpression one or more times

(UN\s) is group 1 which identifies the text UN ending with a space, (UNK\s) is group 2 which identifies the text UNK ending with a space and (\d*) is group 3 which identifies any digit repeated one or more times. '-- --- $3' $3 represents (\d*) before which if we keep '-- ---' the text before $3 group will be replaced with '-- ---'.

Example 7:

```
data have;
input date &$100.;
updated_date=prxchange('s#(UN\s)(UNK\s)(\d*)#-- --- $3#i',-1,date);
cards;
UN UNK 2018
run;
```

Output:

DATE	UPDATED_DATE
UN UNK 2018	-- --- 2018

CALL PRXSUBSTR

The CALL PRXSUBSTR routine searches the variable source with the pattern from PRXPARSE, returns the position of the start of the string, and if specified, returns the length of the string that is matched. By default, when a pattern matches more than one character that begins at a specific position, CALL PRXSUBSTR selects the longest match.

To extract dose information (DOSE) from the CMDECOD we can use PRXPARSE to get the pattern ID for use in CALL PRXSUBSTR and later use the SUBSTR function. The variables i.e., with starting position (start) and length of the dose information (length) are created by CALL PRXSUBSTR. Here start and length are the variables used in SUBSTR to extract the dose information.

\d – matches any digit
[ ] – matches any character or metacharacter within [ ] example [\.|\d|\s] matches a dot or digit or single space
* - matches preceding subexpression one or more times
\w – matches a single letter, when used like \w[mg] – matches a letter with 'm' or 'g'

Sample data:

```
data conmed;
input subject$ cmdecod&:$100.;
cards;
001 RIFAP 5 mg
001 OXCARBAZEPINE 1000 mg
002 CARBAMAZEPINE
002 PHENYTOIN 62.5 mg
run;
```

Example 8:

```
data want;
 set conmed;
  id=prxparse('/\d[\.|\d|\s]*\w[mg]*/oi');
  call prxsubstr(id,cmdecod,start,length);
  if start>0 then dose=substr(cmdecod,start,length);
run;
```

Output:

SUBJECT	CMDECOD	ID	START	LENGTH	DOSE
001	RIFAP 5 mg	1	7	4	5 mg
001	OXCARBAZEPINE 1000 mg	1	15	7	1000 mg
002	CARBAMAZEPINE	1	0	0	
002	PHENYTOIN 62.5 mg	1	11	7	62.5 mg

CALL PRXNEXT

Returns the position and length of a substring that matches a pattern and iterates over multiple matches within one string.

Sample data:

```
data conmed;
input subject$ resupply&$300.;
cards;
001 10 vials for this patient send 13/12/09
002 04/2/12 - 240 mg 1/2/12 - 240 mg 15/2/12 - 240 mg 22/2/12 - 240 mg
003 1st Re-supply - 1/02/2013(medicine) 2nd Re-supply - 1/07/2013(medicine)
004 1st re-supply - 03-FEB-2015 - medicine - 11 vials 2nd re-supply 11-Feb-2015 medicine - 1 vial
005 1st re-supply - 03/JAN/2016 - medicine - 11 vials 2nd re-supply 11/JAN/2016 medicine - 1 vial
006 1st re-supply - 03/MAR/16 - medicine - 11 vials 2nd re-supply 11/MAR/16 medicine - 1 vial
007 resupplied as medicine on 9 Jan 10- same dose
run;
```

Example 9:

```
data want;
 set conmed;
  start=1;
  stop=length(resupply);
  pattern_id=prxparse('/\d{1,2}[\/\-]\d{1,2}[\/\-]\d{2,4}|
                       \d{1,2}[(\-)?(\s)?\,\/]\w{3,}[(\-)?(\s)?\/]\d{2,4}\b/');
  call prxnext(pattern_id,start,stop,resupply,position,length);
   do while(position>0);
   resupply_date=substr(resupply,position,length);
   call prxnext(pattern_id,start,stop,resupply,position,length);
  output;
 end;
run;
```

The following regular expression \d{1,2}[\/\-]\d{1,2}[\/\-]\d{2,4} matches the dates with patterns 4/26/2013 or 28/10/09 and dates with patterns like 14-Mar-2016 or 9 Dec 10 or 11/Feb/2016 or 03/FEB/16 are matched by \d{1,2}[(\-)?(\s)?\,\/]\w{3,}[(\-)?(\s)?\/]\d{2,4}. Both these regular expressions are separated by '|' pipe symbol in PRXPARSE and pattern search is performed on RESUPPLY variable. On execution of the above code we get the below dataset where each date in any pattern mentioned above repeated one or more times in the RESUPPLY variable is output as a separated record in RESUPPLY_DATE variable. The regular expression with CALL PRXNEXT has simplified the identification of multiple resupply dates in different patterns.

Output:

SUBJECT	RESUPPLY	START	STOP	PATTERN_ID	POSITION	LENGTH	RESUPPLY_DATE
001	10 vials for this patient send 13/12/09	40	39	1	0	0	13/12/09
002	04/2/12 - 240 mg 1/2/12 - 240 mg 15/2/12 - 240 mg 22/2/12 - 240 mg	24	66	1	18	6	04/2/12
002	04/2/12 - 240 mg 1/2/12 - 240 mg 15/2/12 - 240 mg 22/2/12 - 240 mg	41	66	1	34	7	1/2/12
002	04/2/12 - 240 mg 1/2/12 - 240 mg 15/2/12 - 240 mg 22/2/12 - 240 mg	58	66	1	51	7	15/2/12
002	04/2/12 - 240 mg 1/2/12 - 240 mg 15/2/12 - 240 mg 22/2/12 - 240 mg	58	66	1	0	0	22/2/12
003	1st Re-supply - 1/02/2013(medicine) 2nd Re-supply - 1/07/2013(medicine)	62	71	1	53	9	1/02/2013
003	1st Re-supply - 1/02/2013(medicine) 2nd Re-supply - 1/07/2013(medicine)	62	71	1	0	0	1/07/2013
004	1st re-supply - 03-FEB-2015 - medicine - 11 vials 2nd re-supply 11-Feb-2015 medicine - 1 vial	76	93	1	65	11	03-FEB-2015
004	1st re-supply - 03-FEB-2015 - medicine - 11 vials 2nd re-supply 11-Feb-2015 medicine - 1 vial	76	93	1	0	0	11-Feb-2015
005	1st re-supply - 03/JAN/2016 - medicine - 11 vials 2nd re-supply 11/JAN/2016 medicine - 1 vial	76	93	1	65	11	03/JAN/2016
005	1st re-supply - 03/JAN/2016 - medicine - 11 vials 2nd re-supply 11/JAN/2016 medicine - 1 vial	76	93	1	0	0	11/JAN/2016
006	1st re-supply - 03/MAR/16 - medicine - 11 vials 2nd re-supply 11/MAR/16 medicine - 1 vial	72	89	1	63	9	03/MAR/16
006	1st re-supply - 03/MAR/16 - medicine - 11 vials 2nd re-supply 11/MAR/16 medicine - 1 vial	72	89	1	0	0	11/MAR/16
007	resupplied as medicine on 9 Jan 10- same dose	35	45	1	0	0	9 Jan 10

CONCLUSION

In this paper we tried to understand the Perl regular expressions and how we can use them through PRX functions in SAS. The examples provided here would help the programmers understand how they can be used in the day to day programming as an alternate to regular SAS functions like INDEXW, FINDW, SUBSTR etc., The metacharacters listed in this paper are only those which are used in the examples, there are a lot of other metacharacters which are available online. To get more information related to metacharacters please refer to some of the resources provided in the reference section of this paper. We covered PRXMATCH, PRXCHANGE and CALL SUBSTR with some examples. We also saw the advantage of call routine CALL PRXNEXT in its robustness to fetch different patterns of dates from the complex text data. These functions along with other PRX functions could be explored further on a wide variety of data and hope this paper has succeeded to some extent in the quest of efficient functions to handle data.

REFERENCES

Google search **of** SAS documentation website for metacharacters and PRX functions**.**
Referred regular expressions website for different metacharacters: https://www.regular-expressions.info/
Windham, K. Matthew. 2014. Introduction to Regular Expressions in SAS®. Cary, NC: SAS Institute Inc.

ACKNOWLEDGMENTS

I would like to thank Claire Kerswill, Bharat Buchupalli, Tejendra Parikh and Jyosthna Kanna for proofreading and their comments on my draft paper.

CONTACT INFORMATION

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

Jagadish Katam
Parexel International
The Quays, 101-105 Oxford Rd
Uxbridge / UB8 1LZ
Work Phone: +44 742 441 6637
Email: Jagadish.katam@parexel.com