Paper PP16

Perl Regular Expressions and Complex Searches in Data

Romain Gruber, IQVIA, Strasbourg, France

ABSTRACT

Perl Regular Expressions are tools for pattern matching and string manipulation. They are essentially sequences of characters that form a pattern, useful for various tasks including searching, replacing, or splitting text.
These expressions are particularly useful when dealing with data where exact values aren't known, but a specific pattern might be present. Perl Regular Expressions allow us to search for and manipulate such patterns.

While SAS® provides numerous functions for string manipulation, Perl Regular Expressions offer several benefits. For standard string processing tasks, such as searching for a specific substring, filtering alphabetical or numeric characters, or replacing one word with another, they can be more concise than conventional SAS functions. Moreover, they facilitate pattern matching and replacement, which may not be achievable with standard SAS functions.

Perl Regular Expressions adhere to a specific syntax, using metacharacters to define patterns. Although they might seem complicated to interpret initially, their usage becomes easier with experience.
You can utilize Perl Regular Expressions in SAS with PRX functions, notably PRXPARSE, PRXMATCH, PRXCHANGE, and PRXPOSN.

INTRODUCTION

A regular expression can include literal characters and special characters called metacharacters. Literal characters match the character exactly, while metacharacters have special meanings.

In SAS, you can use Perl Regular Expressions using PRX functions such as PRXPARSE, PRXMATCH, PRXCHANGE and PRXPOSN.

PRXPARSE compiles a Perl Regular Expression that can then be used for pattern matching.

PRXMATCH function checks if a Perl regular expression pattern matches a specific string. The function returns the position at which the pattern is found.

PRXCHANGE function is used to search and replace patterns in a string. The function returns a new string where the specified changes have been made.

PRXPOSN returns a character string that contains the value for a capture buffer.

METACHARACTERS

The following tables covers some metacharacters that can be used in a Perl Regular Expression.

Metacharacter	Description
\d	matches a digit character that is equivalent to [0-9].
\l	specifies that the next character is lowercase.
\s	matches any white space character, including space, tab, form feed, and so on
\w	matches any word character or alphanumeric character, including the underscore.
\	marks the next character as either a special character, a literal, a back reference, or an octal escape
|	specifies the or condition when you compare alphanumeric strings

^	matches the position at the beginning of the input string.
$	matches the position at the end of the input string.
*	matches the preceding subexpression zero or more times
+	matches the preceding subexpression one or more times
?	matches the preceding subexpression zero or one time
{n,m}	m and n are non-negative integers, where n<=m. They match at least n and at most m times
[...]	specifies a character set that matches any one of the enclosed characters
[^...]	specifies a negative character set that matches any character that is not enclosed
[a-z]	specifies a range of characters that matches any character in the range
(?=...)	specifies a zero-width, positive, look-ahead assertion. In the expression regex1 (?=regex2), a match is found if both regex1 and regex2 match.

PERL REGULAR EXPRESSIONS AND SAS PRX FUNCTIONS

EXAMPLE 1

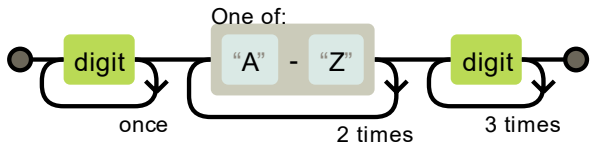
In this first example, we consider a text variable which may contain a date in DDMMMYYYY (date9.) format that needs to be extracted and converted to a numeric variable.

	TEXTVAR
1	Date is 12NOV1990.
2	No date.

Using usual SAS function this might be difficult if the position of the date in the data is not known and if this is a free text variable.
However using PRXMATCH functions this can be done easily.

```
data phuse_ex1;
    set phuse_ex1;
    format vardt date9.;
    match=prxmatch("/\d{2}[A-Z]{3}\d{4}/",upcase(textvar));
    if match^=0 then do;
        vardt=input(substr(textvar,match,9),date9.);
    end;
run;
```

In this example, \d{2}[A-Z]{3}\d{4} specifies a Perl Regular Expression corresponding to a pattern of 2 digits, followed by 3 capital letters, followed by 4 digits.
PRXMATCH functions will search for this pattern and return the position of it if it is found in the variable or 0 if not.



The previous code will produce the following result:

	TEXTVAR	VARDT
1	Date is 12NOV1990.	12NOV1990
2	No date.	

EXAMPLE 2

In this example we consider a dataset with a character variable which contain some test results (typically laboratory results). Results can be of various types.

	LBORRES
1	POSITIVE
2	4.32
3	4.3.2
4	<3.2
5	12
6	7u

Numeric results need to be mapped to a numeric variable. This could be done using usual SAS functions, but this should be programmed very carefully as some special cases need to be considered. For instance, result “4.3.2”, despite containing only digits and “.”, is not a numeric result.
However, this can be done using the PRXMATCH function.

```
data phuse_ex2;
    set phuse_ex2;
    format lbstresn 8.2;
    if prxmatch("/^\d+\.?\d*$/",strip(lborres))
    then lbstresn=input(lborres,best.);
run;
```

^\d+\.?\d*$** specifies a Perl Regular Expression corresponding to a pattern at least one digit, followed zero or one time by “.”, followed zero or more times by a digit. The expression is enclosed between **^** and **$, meaning the variable starts and ends with this pattern and does not contain anything else.

The previous code will produce the following result:

	LBORRES	LBSTRESN
1	POSITIVE	
2	4.32	4.32
3	4.3.2	
4	<3.2	
5	12	12.00
6	7u	

Numeric results are correctly mapped to variable LBSTRESN, and specific cases like “4.3.2” or “<3.2” are not considered as numeric results.

EXAMPLE 3

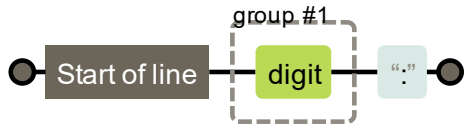
In this example we consider a character variable containing a time. When the time is <12:00 then the variable may be in the H:MM format instead of HH:MM format.

	TIME
1	6:50
2	11:25

To create a character variable in HH:MM format, usual SAS functions could be used, for instance by checking the length of the character variable, if there is a “0” at the beginning of the variable, and so on. Using PRXCHANGE function makes it more compact.

```
data phuse_ex3;
    set phuse_ex3;
    new_time=prxchange("s/^(\d{1}):/0$1:/",-1,time);
run;
```

In the previous expression, **s/** specifies a substitution expression, **^(\d{1}):** specifies a pattern at the beginning of the variable with one digit, considered to be group 1, followed by “:”, and **0$1:** specifies that the previous pattern should be replaced by a 0, followed by the first group (i.e. the digit), followed by “:”.



The previous code will produce the following result:

	TIME	NEW_TIME
1	6:50	06:50
2	11:25	11:25

“0” has been correctly added only for time <12:00: as the first group in the expression is **\d{1}**, only cases where there is only one digit at the start are affected.

EXAMPLE 4

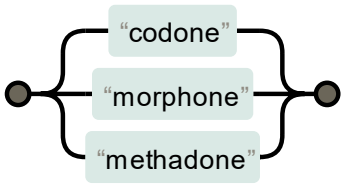
In this example we consider a character variable with either prespecified values or free text.

	CMDECOD
1	PRESINOL
2	OXYCODONE HYDROCHLORIDE
3	DOPAMIDE
4	ROLATUSS WITH HYDROCODONE
5	LEVOMETHADONE

If the variable contains the terms “codone”, “morphone” or “methadone” then the record should be kept, otherwise it should not be kept.
Using usual SAS functions this could be done for instance by using three calls to function INDEX. Using PRXMATCH makes it more compact.

```
data phuse_ex4;
    set phuse_ex4;
    where prxmatch('/codone|morphone|methadone/i',cmdecod);
run;
```

| means “OR”, meaning PRXMATCH will search for the strings “codone” or “morphone” or “methadone”. **/i** indicates that case should be ignored.



The previous code will produce the following result:

	CMDECOD
1	OXYCODONE HYDROCHLORIDE
2	ROLATUSS WITH HYDROCODONE
3	LEVOMETHADONE

CONCLUSION

Perl Regular Expressions in conjunction with SAS PRX functions form a useful toolset for conducting intricate searches within data and replacing patterns. They enable string manipulations that would otherwise be unfeasible or time-consuming. Moreover, they can offer a more concise solution for simpler string manipulations, which are typically performed using standard SAS functions.

However, Perl Regular Expressions and PRX functions can be challenging to interpret, particularly for those unfamiliar with them. This necessitates the need for comprehensive code commenting to explain the specific Perl Regular Expressions being used. In some instances, to enhance code maintainability, it might be preferable to use standard SAS functions.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:
Author Name: Romain Gruber
Company: IQVIA
Email: romain.gruber@iqvia.com
Website: https://www.iqvia.com/