PharmaSUG 2015 – QT29

Sensitivity Training for PRXers

Kenneth W. Borowiak, PPD, Morrisville, NC

ABSTRACT

Any SAS® user who intends to use the Perl style regular expressions through the PRX family of functions and call routines should be required to go through sensitivity training. Is this because those who use PRX are mean and rude? Nay, but the regular expressions they write are case sensitive by default. This paper discusses the various ways to flip the case sensitivity switch for the entire or part of the regular expression, which can aid in making it more readable and succinct.

Keywords: case sensitivity, alternation, character classes, modifiers, mode modified span

INTRODUCTION

Any SAS® user who intends to use the Perl style regular expressions through the PRX family of functions and call routines should be required to go through sensitivity training. Is this because those who use PRX are mean and rude? Nay, but the regular expressions they write are case sensitive by default. Before proceeding with an example, it will be assumed that the reader has at least some previous exposure to regular expressions[1]. Now consider the two queries below in Figure 1 against the CHARACTERS data set for the abbreviation for 'mister' at the beginning of the free-text captured field NAME.

Figure 1
Case Sensitivity of Regular Expressions

Though the two regexen below look similar …

```
data characters ;
 input name $40. ;
datalines ;
MR Bigglesworth
Mini-mr bigglesworth
Mr. Austin D. Powers
dr evil
mr bIgglesWorTH
;
```

```
proc print data=characters ;
 where prxmatch('/^MR/', name) ;
 run ;
```

```
Obs        name
 1     MR Bigglesworth
```

… they generate different results

```
proc print data=characters ;
 where prxmatch('/^Mr/', name);
 run ;
```

```
Obs        name
 3     Mr. Austin D. Powers
```

While the regular expressions `^MR` and `^Mr` specified as the first arguments in the PRXMATCH functions above look very similar, they generate different results. Both regexen (plural geek-speak for 'regular expressions') look for an upper case 'M' at the beginning of the field, as required by the anchor `^`. However, the first expression attempts to match an upper case 'R' in the second byte of the field while the second expression matches a lower case 'r'. In addition, neither of the regexen match the fifth observation 'mr bIgglesWorTH' because of the lower case 'm' at the beginning of the field. This example demonstrates a very important concept: regular expressions are case sensitive by default. This paper discusses the various ways to flip the case-sensitivity switch for the entire or part of the regular expression, which can aid in making it more readable and succinct.

[1] Cassell [2007] and Borowiak [2008] provide nice introductions to the PRX functions and regular expressions.

CHARACTER CLASSES

One way to deal with case-sensitivity when needed is to use a *character class*. You specify a character class with a pair of brackets (i.e. [ ]) and it allows any single character specified within to be matched. Suppose we are interested in displaying observations that begin with 'M'[2] followed by either a lower or uppercase 'r'. Then we could write the regex as displayed in Figure 2.

Figure 2
Using a Character Class to Address Case-Sensitivity

```
/* Match 'M' followed by by 'R' or 'r' at the beginning of the field */

                    proc print data=characters ;
                      where prxmatch('/^M[Rr]/', name) ;
                      run ;

                Obs          name
                 1     MR Bigglesworth
                 3     Mr. Austin D. Powers
```

This character class contains the elements 'R' and 'r'

When crafting a regex that needs to be case sensitive except at few locations with limited surrogates, character classes are a viable way to proceed.

ALTERNATION

Another way to deal with case-sensitivity when needed is to use *alternation*, which is specified with '|' and is analogous to the OR logical operator. Suppose we are again interested in displaying observations that begin with 'M' followed by either a lower or uppercase 'r'. Then we could write the regex as displayed in Figure 3A.

Figure 3A
Using Alternation to Address Case-Sensitivity

```
/* Match 'M' followed by by 'R' or 'r' at the beginning of the field */

                    proc print data=characters ;
                      where prxmatch('/^M(R|r)/', name) ;
                      run ;

                Obs          name
                 1     MR Bigglesworth
                 3     Mr. Austin D. Powers
```

Parens cause the submatch to be captured in a temporary variable

The regex matches the first and third observations from the CHARACTERS data set, as desired. Unfortunately, using parenthesis ( ) to group the alternatives causes a side effect in this simple pattern matching exercise. The parens cause the characters matched within to be 'captured' and held in a temporary variable, which could be extracted later using the PRXPOSN

[2] The ^ has a different meaning inside of a character class than it does outside of one. See Borowiak [2008] for more details on the use of ^ as a metacharacter.

function[3] or call routine. Since we are not interested in pulling out the matched characters in this example, there is no need to expend the resources needed to maintain the variable. To use the parens for grouping only, we can use non-capturing parens specified with (?:), as demonstrated below in Figure 3B.

Figure 3B
Using Alternation to Address Case-Sensitivity

```
/* Match 'M' followed by by 'R' or 'r' at the beginning of the field */

                    proc print data=characters ;
                      where prxmatch('/^M(?:R|r)/', name) ;
                      run ;

                Obs          name
                 1     MR Bigglesworth
                 3     Mr. Austin D. Powers
```

Make the parens non-capturing by using (?:)

As with character classes, using alternation in a regular expression that needs to be case sensitive except at a limited number of locations is practical.

i MODIFIER

Let us now consider the task of matching records with the abbreviation for 'mister' at the beginning of the field while ignoring any differences in case. While we could write the regex using alternation or character classes, there is another option. Using the i modifier at the end of the regex makes the pattern search case-insensitive, as demonstrated below in Figure 4A.

Figure 4A
Case Insensitive Modifier i

```
/* Match 'M' followed by by 'R' or 'r' at the beginning of the field */

                    proc print data=characters ;
                      where prxmatch('/^Mr/i', name) ;
                      run ;

                Obs          name
                  1     MR Bigglesworth
                  3      Mr. Austin D. Powers
                  5     mr bIgglesWorTH
```

The i modifier is specified after the closing delimiter to make the search case insensitive

When the case-insensitive modifier is specified at the end of the regex it applies to the entire pattern. This changes the default behavior of the Perl style regular expressions, which are case-sensitive. So an *isomorphic*[4] regex could have been written as shown on in Figure 4B.

[3] PROC PRINT does not provide a means to extract the submatch and create a new variable. In this example where we are just displaying results, you could extract the submatch and display with a single PROC SQL step.

[4] Objects that are isomorphic are structurally identical if you choose to ignore finer-grained differences that may arise from how they are defined.

Figure 4B
Case Insensitive Modifier i

```
/* Match 'M' followed by by 'R' or 'r' at the beginning of the field */

                    proc print data=characters ;
                      where prxmatch('/^mR/i', name) ;
                      run ;
```

Obs	name
1	MR Bigglesworth
3	Mr. Austin D. Powers
5	mr bIgglesWorTH

This regex is isomorphic to the one in Figure 4A

Making the regex case-insensitive with the i modifier is more convenient, succinct, and arguably more readable relative to alternation and character classes when the pattern to match is long. Consider the example below in Figure 4C where a case-insensitive search for the word 'Bigglesworth' is desired.

Figure 4C
Case Insensitive Modifier i

```
proc print data=characters ;
 where prxmatch('/\bBigglesworth\b/i', name) ;
 run ;
```

Obs	name
1	MR Bigglesworth
2	Mini-mr bigglesworth
5	mr bIgglesWorTH

```
proc print data=characters ;
 where  prxmatch('/\b[Bb][Ii][Gg]{2}[lL][eE][Ss][Ww][oO][Rr][tT][Hh]\b/',name);
 run ;
```

You can see that needing to create a character class for every non-repeating alpha character (disregarding case) can be rather tedious.

i MODE MODIFIED SPAN

The case when the entire pattern to be matched is to be case-insensitive has been examined. Let us now turn our attention to the case where only part of the expression is required to be case insensitive. Suppose we are interested in displaying observations that begin with a lower case 'b' at the beginning of a word followed by the string 'igglesworth', where the characters can be upper or lower case. The i modifier cannot help us here because it negates the ability to detect the lower case 'b' at the beginning of the word. This is where a *mode modified span* is useful. A case-insensitive mode modified span is specified by (?i:) within the delimiters of the regex and makes the subexpression within the parenthesis case-insensitive, as demonstrated below in Figure 5A.

Figure 5A
Partial Case-Insensitivity with a Mode Modified Span

```
proc print data=characters ;
  where prxmatch('/\bb(?i:ig{2}LESwoRTh)\b/', name) ;
  run ;
```

```
 Obs            name
 2     Mini-mr bigglesworth
 5     mr bIgglesWorTH
```

Note that when the i modifier is omitted after the closing delimiter in the expression the expression is case sensitive. This allows for the beginning lower case 'b' to be matched. The case-insensitive mode modifying span then overrides the case sensitive mode and allows the subexpression in the parens to be a case-insensitive match. The parens used in a mode modified span are non-capturing[5].

You can also create a case-sensitive mode modified span when the i modifier is specified, as shown below in Figure 5B.

Figure 5B
Partial Case-Insensitivity with a Mode Modified Span

```
proc print data=characters ;
 where prxmatch('/\b(?-i:b)ig{2}LESwoRTh\b/i', name) ;
 run ;
```

```
 Obs            name
 2     Mini-mr bigglesworth
 5     mr bIgglesWorTH
```

In this example, the mode modifying span is used to turn case sensitivity on in order to match the leading lower case 'b'. The syntax for a case-sensitive mode modifying span is to add a dash before the i in the non-capturing buffer.

[5] You can think of a non-capturing buffer (?:) as a mode modifying span where the mode is unchanged.

CONCLUSION

The Perl style regular expression that available to the SAS user through the PRX functions and call routine are an invaluable tool for pattern matching, which may not require the alpha characters to match exactly. Regular expressions are case-sensitive, but the default setting can be turned off by the specifying the i modifier. Under certain conditions, the i modifier can be more convenient, succinct and readable vis-à-vis character classes and alternation. When you need part of the regex to be case-(?:in)?sensitive, modifying spans are handy.

You may have noticed there was hardly a mention of the relative efficiency amongst the various methods to handle case-insensitivity in regular expressions. The primary goal of the paper was to focus to introduce the techniques. However, regardless of the method you employ to handle case-insensitivity, you are asking the regex engine to work harder and each of these techniques incurs a certain amount of overhead and resources. The internal workings of the regex engine are well beyond the scope of the paper. If you are interested in efficiency then you should consider benchmarking the various techniques against sample or hypothetical data.

REFERENCES

Borowiak, Kenneth W. (2008), "PRX Functions and Call Routines: There Is Hardly Anything Regular About Them", *NorthEast SAS Users Group Proceedings*
http://www.lexjansen.com/nesug/nesug08/bb/bb11.pdf

Cassell, David L. (2007), "The Basics of the PRX Functions", *SAS Global Forum Proceedings*
http://www2.sas.com/proceedings/forum2007/223-2007.pdf

SAS OnLineDoc®

Friedl, Jeffrey E.F. (2006), *Mastering Regular Expressions 3rd Edition* : O'Reily Media Inc.

ACKNOWLEDGEMENTS

The author would like to thank Toby Dunn and Pramod Dokuri for their insightful comments in reviewing this paper.

DISCLAIMER

The contents of this paper are the work of the author and do not necessarily represent the opinions, recommendations, or practices of PPD.

CONTACT INFORMATION

Your comments and questions are valued and encouraged.

Ken Borowiak
PPD
3900 Paramount Parkway
Morrisville NC 27560

Ken.Borowiak@ppdi.com
Ken.Borowiak@gmail.com