Besserer Abgleich mit regulären Ausdrücken

Daniel Schulte
viadee Unternehmensberatung AG
Anton-Bruchausen-Straße 8
48147 Münster
daniel.schulte@viadee.de

Zusammenfassung

Reguläre Ausdrücke lesen sich oft wie eine fremde Sprache und sind auf den ersten Blick meist verwirrend. Mit dem Werkzeug lassen sich aber hervorragend Muster formulieren, mit denen sich Textvergleiche deutlich besser umsetzen lassen. Statt verschachtelter Substring, Index und if-then-else Konstrukte genügt meist ein regulärer Ausdruck.
Neben einem Überblick über die Syntax und Funktionen in SAS werden in dem Beitrag auch Praxisbeispiele und Fallstricke gezeigt.

Schlüsselwörter: Pattern, Abgleich, reguläre Ausdrücke, SAS Base

1 Ausgangslage

Ob bei der Datenaufbereitung und der Vereinheitlichung von Inhalten im Rahmen von ETL Verarbeitungen oder beim Abgleich von Informationen bei Abfragen: Die Formatierung von Inhalten erzeugt früher oder später die Notwendigkeit, Inhalte nach Mustern oder Teilinformationen zu gliedern, filtern oder abzuleiten.
Oft sind dann verschachtelte Ketten von substr oder index Funktionen zu finden. Diese lassen sich aber oft auf schlankere Weise durch reguläre Ausdrücke ersetzen.

2 Überblick reguläre Ausdrücke

Die Beschreibung von Mustern, die für das menschliche Auge recht trivial erscheinen, ist eine oft auftretende Fragestellung, die nicht mehr ganz so trivial in Abfragen implementiert werden kann. Nehmen wir als einfaches Beispiel eine Telefonnummer. Welche der folgenden Nummern ist eine gültige Telefonnummer:

- 0251-77777/0
- (0251) 77777 - 0
- +49 251 / 7 7 7 7 7 0
- 777770

Auf den ersten Blick sehen alle valide aus. Sogar die letzte, wenn man sich im Ortsnetz Münster (Vorwahl 0251) aufhält. Die Prüfung auf eine korrekte Telefonnummer in einem Datenstrom (egal ob in einem Formular oder in einem ETL Prozess) zu formulieren, artet im ungünstigsten Fall in einer Folge von substr und index Funktionen aus.

Genau hier setzen reguläre Ausdrücke an. Mit einer auf den ersten Blick kryptischen Syntax können auch komplexe Muster abgebildet werden. Der einfachste Ansatz wäre: „sind es Ziffern?“. Die Abfrage nach Ziffern (englisch „digit“) erfolgt mit dem Element `\d`. Da es sich aber um eine unbekannte Anzahl an Ziffern handelt, muss man hier noch eine Menge beschreiben.

- ? 0 oder 1 Auftreten des vorherigen Elements
- * 0 oder beliebig viele Wiederholungen des vorherigen Elements
- + 1 oder beliebig viele Wiederholungen des vorherigen Elements

Da Telefonnummern aus einer Vorwahl und einer Durchwahl bestehen sollen, sind mind. 5 Ziffern wünschenswert. Die Quantifizierung dafür sähe so aus: `{5,}`
Da es aber auch nach oben ein Limit geben soll, kann man den zweiten Parameter in den geschweiften Klammern nutzen, um ein Maximum von z.B. 20 zu nutzen: `{5,20}`
Für einen Treffer mit dem Ausdruck /`\d{5,20}`/ ist es nun notwendig, dass zwischen 5 und 20 Ziffern aufeinander folgen. Sind diese durch Trennzeichen wie - / oder Leerzeichen gegliedert, beispielsweise 0251 – 777 77 / 0, würde hier kein Treffer erzielt.

Man kann nun die Suche erweitern, dass dann eben nicht nur Ziffern, sondern auch diese Trennzeichen mit erfasst werden und damit Bestandteil einer gültigen Telefonnummer sein sollen. Diese Zusammenfassung der relevanten Zeichen erfolgt in eckigen Klammern. Da Schrägstriche und minus Zeichen auch Steuerzeichen in einem regulären Ausdruck sind, müssen diese durch ein Backslash maskiert werden.

```
[\d\-\/ ]{5,20}
```

Die Quantifizierung erfolgt nun auf alle in den eckigen Klammern eingefassten Elemente.

Da Telefonnummern ohne Vorwahlen nur bedingt hilfreich sind, macht es Sinn, diese als Notwendigkeit für eine gültige Telefonnummer mit zu erfassen. Dabei soll auch die häufig anzutreffende Einfassung in runden Klammern, die übrigens auch ein Steuerzeichen für reguläre Ausdrücke sind, berücksichtigt werden: (0251). Vorwahlen starten mit einer 0 und können dann 2 bis 5 Ziffern beinhalten.

- `\(?` Eine optionale Runde Klammer
- `0` Eine führende 0
- `\d{2,5}` Mind. 2/ max. 5 Ziffern
- `\)?` Eine optionale Runde Klammer

Zusammengesetzt ist dies `\(?0\d{2,5}\)?` für die Vorwahl und komplett sieht der Ausdruck wie folgt aus: `\(?0\d{2,5}\)?[\d\-\/ ]{5,20}`
Was noch fehlt ist die Erkennung von internationalen Vorwahlen wie +49 oder 0049.

- `(` Start einer Gruppe
- `\+` Ein + Zeichen
- `|` Oder

- `00` Zwei Nullen
- `)` Ende einer Gruppe
- `\d{1,3}` Mind. 1/ max. 3 Ziffern

Zusammengesetzt also: `(\+|00)\d{1,3}`

Komplett ergibt sich damit – mit diesem „einfachen“ Regelwerk - folgender Ausdruck
`/((\+|00)\d{1,3}|\(?0\d{2,5}\)?)[\d\-\/ ]{5,20}/`
Hier wurde zusätzlich noch eine Klammerung und ODER-Verknüpfung hinzugefügt.
Schauen wir uns die einzelnen Bestandteile des Ausdrucks an:

`/`	Durch den Schrägstrich wird der reguläre Ausdruck gestartet bzw. beendet.
(…)	Durch die Klammerung können Bereiche gruppiert werden, hier zur Bestimmung der Vorwahl
`\d`	Beschreibt Ziffern (0 … 9)
`|`	Oder
`\`	Maskierung von Schlüsselzeichen wie Klammern oder Schrägstrichen
`?`	Das Vorherige Element ist optional
`{1,3}`	Quantifizierung des vorherigen Elements Hier: 1 bis 3 mal
`[…]`	Eines der enthaltenen Zeichen
`+`	1 oder mehr des vorherigen Elements
`(\+|00)\d{1,3}`	Ein Pluszeichen oder 00 gefolgt von 1-3 Ziffern
`\(?0\d{2,5}\)?`	Eine in optionalen Klammern geführte 0 mit 2 bis 5 Ziffern
`[\d\-\/ ]{5,20}`	5 bis 20 - Ziffern - - (Minuszeichen) - / (Schrägstrich) - Leerzeichen

Hier kann man sehen, dass das Herunterbrechen eines Ausdrucks dem Verständnis deutlich hilft.

Um mit regulären Ausdrücken zu experimentieren gibt es z.B. die Webseite https://regexr.com/ .Hier findet sich neben einer guten Referenz auch die Möglichkeit, mit Ausdrücken live zu experimentieren und sich Modifikationen am Ausdruck direkt in eigenen Textelementen zu visualisieren.

Abbildung 1: Beispiel unter https://regexr.com/

Auf der Seite gibt es darüber hinaus noch einen „Explain“ Modus, der ähnlich der obigen Tabelle den Ausdruck auseinander nimmt.

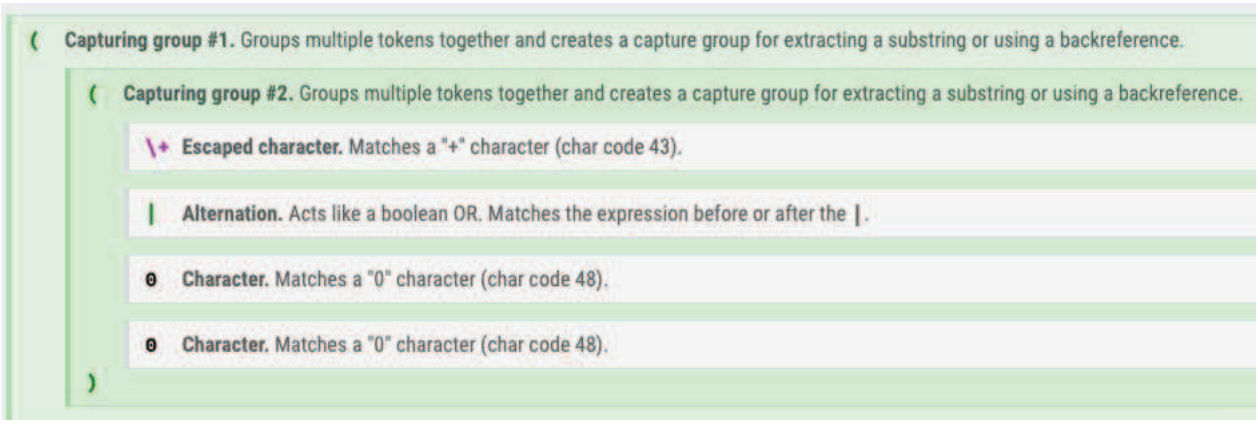

Abbildung 2: Erläuterungen unter https://regexr.com/

2 Reguläre Ausdrücke in SAS

In SAS kann über Funktionen auf reguläre Ausdrücke zugegriffen werden. Einige Funktionen benötigen statt des Ausdrucks eine ID. Diese kann mittels der Funktion PXRPARSE erstellt werden:

```
PRX_ID=prxparse('/Regulärer Ausdruck/');
```

2.1 PRXMATCH

Diese Funktion kann genutzt werden, um eine Übereinstimmung und Position in einer Variablen oder Zeichenkette zu prüfen.

Syntax: `PRXMATCH (Ausdrucks_ID | Ausdruck , zu durchsuchendes Element)`
Ausgabe der Funktion ist ein Nummerischer Wert:

- 0 wenn zu dem Ausdruck kein Treffer gefunden wurde
- eine Zahl > 0, welche die Position des Treffers angibt

Die Funktion bietet sich vor allem für Abfragen nach bestimmten Inhalten im DataStep oder PROC SQL an.

Beispiel:

```
data telefon;
input tel $20.;
datalines;
Tel: 0251-77777/0
(0251) 77777 - 0
+49 251 / 7 7 7 7 7 0
777770
;
run;

data Ausgabe;
set telefon;
   position=
      prxmatch('/((\+|00)\d{1,3}|\(?0\d{2,5}\)?)[\d\-\/ ]{5,20}/'
               , tel);
run;
```

	tel	position
1	Tel: 0251-77777/0	6
2	(0251) 77777 - 0	1
3	+49 251 / 7 7 7 7 7	1
4	777770	0

Abbildung 3: Ausgabe EG

2.2 PRXCHANGE

Wenn wir unser Beispiel mit den Telefonnummern wieder etwas vereinfachen, wird die Arbeitsweise der PRXCHANGE Funktion deutlich:

- +49 251 777770
- +49 251 77777142

Die Nummern liegen ordentlich formatiert als Telefonnummer vor. Nun sollen Ländervorwahl, Ortsvorwahl und Durchwahl in einzelnen Variablen abgelegt werden. Reguläre Ausdrücke können durch runde Klammern (…) Gruppen erzeugen, die dann mit der PRXCHANGE Funktion aufgegriffen werden können.
Der reguläre Ausdruck zur Beschreibung der obigen Telefonnummern sieht wie folgt aus:

(\+\d{1,3}) (\d{1,5}) (\d{1,12})
Auf der Seite https://regexr.com/ wird das unter den Tools (Details) wie folgt dargestellt:

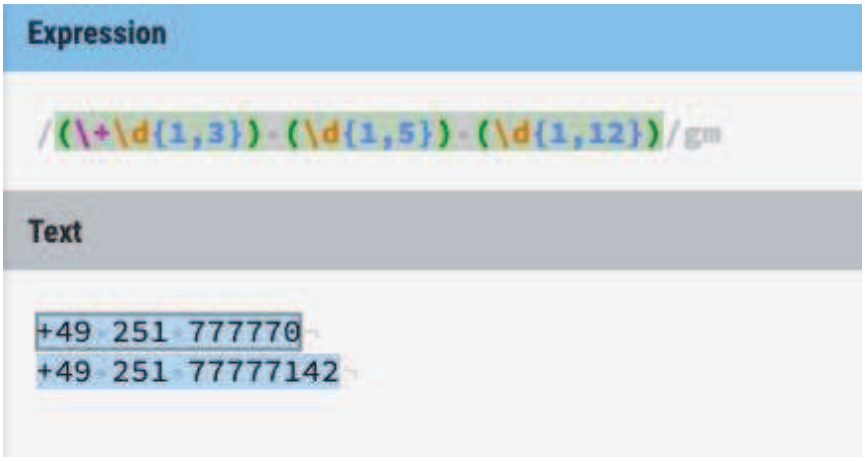

Abbildung 4: Beispiel unter https://regexr.com/

Abbildung 5: Erläuterungen unter https://regexr.com/

Diese Gruppen können dann für eine Ersetzung oder Extraktion genutzt werden.
Syntax: `PRXCHANGE (Ausdrucks_ID | Ausdruck , Anzahl der Vorgänge, zu durchsuchendes Element)`

Die Ausgabe ist eine neue Zeichenkette.

Der reguläre Ausdruck ist nun jedoch nicht mehr nur /…/ sondern s/… / …./
Beginnend mit einem s wird zwischen den ersten beiden Schrägstrichen weiterhin das Suchmuster zu hinterlegt, zwischen dem zweiten und dritten Schrägstrich wird das Ausgabemuster hinterlegt. In dem Ausgabemuster kann auf die Gruppen mit $1 … $ n zugegriffen werden. Um in unserem Beispiel die Ländervorwahl zu extrahieren, ist nur die erste Gruppe notwendig:

```
land=prxchange('s/(\+\d{1,3}) (\d{1,5}) (\d{1,12})/$1/',1 , tel);
```

In dem Ausgabemuster kann dann auch weiterer Text hinzugefügt werden, wie in diesem Beispiel die führende Null für die Ortsvorwahl
Beispiel:

```
Data telefon2;
input tel $20.;
datalines;
+49 251 777770
+49 251 77777142
;
run;

data Ausgabe;
set telefon2;
   land=prxchange('s/(\+\d{1,3}) (\d{1,5}) (\d{1,12})/$1/',1 , tel);
   ort=prxchange('s/(\+\d{1,3}) (\d{1,5}) (\d{1,12})/0$2/',1 , tel);
   durchwahl=prxchange('s/(\+\d{1,3}) (\d{1,5}) (\d{1,12})/$3/',1
                       ,tel);
run;
```

	tel	land	ort	durchwahl
1	+49 251 777770	+49	0251	777770
2	+49 251 77777...	+49	0251	77777142

Abbildung 6: Ausgabe EG

Mittels dieser Funktion können aber auch z.B. Namen unterschiedlicher Notation an- oder abgeglichen werden. Die Varianten Nachname, Vorname in Vorname Nachname umzuformen bedarf lediglich der Zeile:

```
Neuer_Name=prxchange('s/(\w+), (\w+)/$2 $1/',1 , Name);
```

3 Fallstricke

In der Anwendung regulärer Ausdrücke kommt es auf eine hohe Präzision an. Ein falsches Zeichen kann dafür sorgen, dass der Ausdruck eine andere Bedeutung bekommt. Beim Lesen der obigen Beispiele fallen jedem sicher noch diverse Variationen ein, die dabei nicht berücksichtigt wurden. Die Erweiterung der Ausdrücke auf solche Variationen kann diese schnell zu komplex und nicht mehr nachvollziehbar machen. Daher muss hier ein gesunder Mittelweg gefunden werden!

Die in den Beispielen genutzte direkte Verwendung der Ausdrücke ist in realen Umgebungen eher unangebracht. Alleine in dem Beispiel zur Splittung der Telefonnummer ist der Ausdruck dreifach redundant genannt. Die Verwendung von gleichen Ausdrücken macht die PRXPARSE Funktion recht einfach, da dann immer nur noch mit der ID gearbeitet werden muss. Alternativ bietet es sich an, häufig genutzte Ausdrücke in Macrovariablen auszulagern:

```
%let prx_tel=(\+\d{1,3}) (\d{1,5}) (\d{1,12});
data Ausgabe;
set telefon2;
   land=prxchange("s/&prx_tel./$1/",1 , tel);
   ort=prxchange("s/&prx_tel./0$2/",1 , tel);
   durchwahl=prxchange("s/&prx_tel./$3/",1 , tel);
run;
```

Werden diese über eine gemeinsame Bibliothek verwaltet, ergeben sich daraus auch Synergien! In dem Beispiel für die Aufteilung der Telefonnummer werden drei Gruppen auf der obersten Ebene verwendet. Wird mit den Klammern nun eine Verschachtelung genutzt, wird die Zählweise schnell unübersichtlich. Die Zählweise geht sequenziell der Reihe der geöffneten Klammern durch.

(Gruppe 1(Gruppe 2))(Gruppe 3(Gruppe 4))

Beim Einfügen einer neuen Gruppe kann dann die Zählweise durcheinandergewürfelt werden

Darüber hinaus sind aber auch umgebungsspezifische Eigenheiten zu berücksichtigen. Unter Windows / Unix ist es z.B. legitim

```
[A-z]
```

als Formulierung für das Alphabet in Groß- und Kleinbuchstaben zu verwenden. Unter zOS führt dieser Ausdruck allerdings zu:

```
ERROR: Invalid Ý¨ range "A-z" before HERE mark in regex m/[A-z <<
HERE ]*/
```

Für einen fehlerfreien Lauf muss hier

```
[A-Za-z]
```

genutzt werden. Um möglichst flexibel mit dem erstellten Code zu sein, bietet es sich an, dies im Hinterkopf zu behalten um nicht bei der Überführung von Code auf den Host von solchen Meldungen überrascht zu werden.

Des Weiteren wird bei der ersten Notation auch der ASCII Bereich 91-121 (u.a. ^ _) mit erfasst und kann auch unter Windows / Unix so zu ungewollten Nebenwirkungen führen. Wichtig auch: Sonderzeichen wie z.B. é ê Ö werden weder von `[A-Za-z]` noch `\w` mit erfasst!

Bei der Anwendung in Abfragen sind darüber hinaus auch mehr Ressourcen für die Ausführung einzuplanen. Bei der Implementierung hat es sich als gute Praxis herausgestellt, reguläre ausdrücke in ETL-Stecken zu verwenden, um verschachtelte if- then-else oder substr / index kompakter zu implementieren. Bei Abfragen sollte es auf das nötigste reduziert werden, da Laufzeiten hier deutlich nach oben gegangen sind.

Um die Laufzeit bei komplexeren Abfragen zu reduzieren, bietet es sich ebenfalls an, in der Initialisierung von einem Datastep den Ausdruck zu parsen:

```
if _N_ = 1 then
   do;
      retain prx_id;
      prx_id = prxparse("/Regulärer Ausdruck/");
   end;
```

Literatur

[1] SAS Programming Documentation (https://go.documentation.sas.com/?cdcId=pgmsascdc&cdcVersion=9.4_3.4&docsetId=pgmsashome&docsetTarget=home.htm&locale=en)