Anwendung von (Perl) Regular Expressions für die Mustersuche in Strings

Andreas Deckert
Institute of Public Health
INF 324
69120 Heidelberg
a.deckert@uni-heidelberg.de

Heiko Zimmermann
Institute of Public Health
INF 324
69120 Heidelberg
h.zimmermann@uni-heidelberg.de

Zusammenfassung

SAS bietet verschiedene Funktionen wie z.B. SCAN, SUBSTR und INDEX zur Manipulation von Strings und Texten z.B. bei der Datenvalidierung und beim Ersetzen von Text-Teilen. Bei komplexen Problemstellungen wird der Code jedoch schnell umfangreich und unübersichtlich. Für Situations-flexible Algorithmen sind Schleifen erforderlich. Alternativ dazu können mit den Funktionen der (Perl) Regular Expressions selbst komplexe Problemstellungen mit wenigen Zeilen Code wesentlich effizienter gelöst werden. Ausgehend von einem einfachen Beispiel wird hier die Verwendung der (Perl) Regular Expressions veranschaulicht und deren Funktionsweise erläutert. Abschließend wird die Mächtigkeit dieses Werkzeugs an einer komplexen Problemstellung demonstriert.

Schlüsselwörter: Regular Expression, PRXPARSE, PRXSUBSTR, PRXCHANGE

1 Einleitung: (Perl) Regular Expressions

Regular Expressions (Reguläre Ausdrücke) können in SAS seit Version 6.2 verwendet werden. SAS-eigene Reguläre Ausdrücke wurden früher schon umfangreich bei der KSFE behandelt [1] und auch die Perl Regular Expressions waren schon einmal Thema von Tipps und Tricks [2]. Trotz ihrer vielen Vorteile sind sie jedoch nach wie vor kaum bekannt und viele Nutzer schrecken vor ihrer Verwendung zurück. Das Ziel dieses Artikels ist daher nicht, eine umfangreiche Einführung in Perl Regular Expressions zu geben, sondern stattdessen vor allem zur Anwendung der Regulären Ausdrücke anzuregen. Für eine detailliere Einführung sei auf [2], [3] verwiesen.

Der Begriff Regulärer Ausdruck stammt aus der Theoretischen Informatik und bezeichnet "eine Zeichenkette die der Beschreibung einer Menge bzw. Untermengen von Zeichenketten mit Hilfe bestimmter syntaktischer Regeln dient"[1]. Hiermit lassen sich Filter und Schablonen für Textbausteine erstellen, nach denen dann innerhalb eines Strings gesucht werden kann. Für den Aufbau Regulärer Ausdrücke steht eine Syntax aus (reservierten) Zeichen eines Alphabets zur Verfügung, die dann über die drei Operatoren Alternative, Verkettung und Wiederholung miteinander kombiniert werden können. Je

[1] Für eine allgemeine Einführung in Reguläre Ausdrücke siehe Wikipedia: http://de.wikipedia.org/wiki/Regulärer_Ausdruck [Zugriff 16.03.2011]

nach Anordnung und Kombination dieser Zeichen können damit verschiedene Anordnungen von Wörtern erfasst und verarbeitet werden.

Die Verwendung Regulärer Ausdrücke ist in SAS seit Version 6.2 in SAS integriert. Diese sogenannten SAS Regular Expressions haben eine eigene Syntax und sind in den RX-Funktionen hinterlegt. Beispielhaft seien hier die RX-Funktionen RXPARSE zur Erstellung eines Regulären Ausdrucks und RXCHANGE zum Ersetzen von Textteilen genannt. Seit Version 9 bietet SAS auch die Möglichkeit, sogenannte Perl Regular Expressions über die PRX-Funktionen zu verwenden, wodurch die Mächtigkeit für Textmanipulationen deutlich erweitert wurde. Die Syntax dieser Regulären Ausdrücke orientiert sich dabei an der Programmiersprache Perl. Auch hier gibt es mehrere Funktionen, die von SAS für den Umgang mit Perl Regular Expressions bereitgestellt werden, wie z.B. PRXPARSE (Erstellen des Regulären Ausdrucks), PRXSUBSTR (Ausgabe von Position und Länge eines gefundenen Ausdrucks in einem String) und PRXCHANGE (Austausch von Textteilen).

Im Folgenden soll nun anhand eines einfachen Beispiels die Verwendung von (Perl) Regular Expressions den Standard-SAS-Funktionen gegenübergestellt werden. Für die weiteren Betrachtungen werden dann die Perl Regular Expressons verwendet, da sie in ihrem Funktionsumfang und ihrer Syntax den SAS-Regular-Expressions überlegen sind.

2 Einführendes Beispiel

Angenommen ein Textstring enthalte ein Sonderzeichen, das entfernt werden soll, z.B. das $-Zeichen in "Dies ist ein Bei$spiel". Eine Möglichkeit dieses Zeichen zu entfernen, besteht in der Verwendung der Funktionen INDEX und SUBSTR:

```
DATA Beispiel;
     text = "Dies ist ein Bei$spiel";
     pos = INDEX(text,"$");
     l_total = LENGTH(text);
     l_part = l_total - pos;
     _text = SUBSTR(text,1,pos-1)||SUBSTR(text,pos+1,l_part);
RUN;
```

Dazu wird mit INDEX zunächst die Position des Zeichens in dem String bestimmt. Mit SUBSTR können nun die Textteile vor und nach dem Zeichen aus dem String herausgeschnitten werden und dann über eine Konkatenation wieder zusammengesetzt werden (entweder mit der Funktion CATS oder mit || in der Kurzform).

Der entsprechende Code mit SAS Regular Expressions sieht folgendermaßen aus:

```
DATA Beispiel;
     LENGTH _text $ 25;
     text = "Dies ist ein Bei$spiel";
     rx = RXPARSE("'$' to '");
```

```
     CALL RXCHANGE(rx,1,text,_text);
RUN;
```

Dabei wird mit RXPARSE der Reguläre Ausdruck definiert, der in diesem Fall eine Regel zur Ersetzung von $ durch eine leere Menge beschreibt, was gleichbedeutend ist mit dem Entfernen des Zeichens. Man kann sich die Vorgehensweise der Funktion CALL RXCHANGE nun wie folgt vorstellen: Die mit RXPARSE erstellte Schablone wird von links nach rechts über den Text geschoben und sobald der erste Teil der Schablone auf eine Entsprechung im String trifft, wird die Ersetzungsregel angewandt. Die Zahl 1 innerhalb von RXCHANGE bedeutet dabei, dass die Ersetzung nur einmal vorgenommen werden soll. Anstelle der 1 kann auch eine beliebige Zahl eingesetzt werden; -1 bedeutet hingegen, dass jedes $ in dem zugrundeliegenden String ersetzt werden soll. Hierbei wird schon ein erster Vorteil gegenüber dem ersten Code (s. oben) sichtbar: Will man mit den Prozeduren INDEX und SUBSTR ebenso eine beliebige Anzahl von $ innerhalb eines Strings ersetzen, kommt man nicht umhin, dafür z.B. eine DO-UNTIL-Schleife zu installieren, die so oft ausgeführt werden muss, bis mit INDEX keine Position mehr für ein $ gefunden wird.

Mit Perl Regular Expressions, also der Syntax der Programmiersprache Perl, sieht der Code mit der gleichen Funktionalität wie folgt aus:

```
DATA Beispiel;
     LENGTH _text $ 25;
     text = "Dies ist ein Bei$spiel";
     pattern = PRXPARSE('s/\$//');
     CALL PRXCHANGE(pattern,-1,text,_text);
RUN;
```

Hier werden nun anstelle der Funktionen RXPARSE und RXCHANGE die Funktionen PRXPARSE und PRXCHANGE verwendet. Die Definition des Regulären Ausdrucks sieht hier ein wenig kryptischer aus. Das erste "*s*" innerhalb dieser Definition steht für "Substitution" und zeigt an, dass das darauffolgende Suchmuster ersetzt werden soll. Zwischen den auf das "*s*" folgenden Schrägstrichen "/" befindet sich die Beschreibung des eigentlichen Suchmusters, in unserem Fall das $-Zeichen. Diesem ist hier ein Backslash ("\") vorangestellt, da das $-Zeichen innerhalb der Perl-Syntax ein reserviertes Zeichen mit einer bestimmten Funktion darstellt. Zwischen dem zweiten und dem letzten Schrägstrich befindet sich dann die Beschreibung der Ersetzung, in unserem Beispiel hier eine leere Menge.

Der Code lässt sich noch weiter kürzen indem die Musterbeschreibung direkt in PRXCHANGE integriert wird. Schon hier ist zu erkennen, mit wie wenig Code die Regulären Ausdrücke im Vergleich zu den üblichen SAS-Funktionen auskommen können:

```
DATA Beispiel;
     text = "Dies ist ein Bei$spiel";
     text = PRXCHANGE('s/\$//',-1,text);
```

```
RUN;
```

Mit diesem Code können auch Sätze wie "Dies i$$$st ein Beisp$$iel" problemlos und mit wenig Aufwand bereinigt werden.

Die Syntax der Musterbeschreibung innerhalb des Regulären Ausdrucks ist sicherlich gewöhnungsbedürftig und wirkt auf den ersten Blick eher verwirrend. Hat man sich allerdings ein paar der häufigsten Zeichen eingeprägt, kann man relativ zügig Ausdrücke für entsprechende Muster zusammenstellen und anwenden. Die wichtigsten Zeichen und ihre Bedeutung sind in der folgenden Tabelle zusammengestellt:

Tabelle 1: Wichtige Zeichen für die Musterbeschreibung

Muster	Bedeutung
/	Beginn und Ende der Regular Expression
\w	Abgleichen von Buchstaben
\d	Abgleichen von Ziffern
[Tel]	Es wird explizit nach "Tel" gesucht
?	Falls nicht oder 1x vorhanden
+	Falls 1x oder beliebig oft vorhanden
*	Falls nicht oder beliebig oft vorhanden
(*Muster*)	Gruppierung und Speicherung
$1	Abruf des ersten Speichereintrages
s/*Muster*/*Ausgabe*/	Einsetzen bzw. Ersetzen von Textteilen

Wie oben schon angedeutet, steht der Schrägstrich "/" jeweils für den Beginn und das Ende eines regulären Ausdrucks. Ein Backslash "\" gefolgt von einem "*w*" bedeutet, dass das gesuchte Muster irgendeinen Buchstaben enthalten muss. Dagegen wird bei der Verwendung von "*d*" nach Ziffern gesucht. Sollen bestimmte Buchstaben oder Ziffern gesucht werden, müssen diese explizit in rechteckigen Klammern "[ ]" eingeschlossen angegeben werden. Falls einem Suchmuster z.B. bestehend aus "*w*" ein "*?*" folgt, bedeutet dass, dass nach keinem oder genau einem Buchstaben gesucht wird. Steht statt dem Fragezeichen ein "+" nach "*w*", sucht die Schablone nach mindestens einem oder beliebig vielen aneinander gereihten Buchstaben. Bei einem Stern wird nach keinem oder beliebig vielen Buchstaben gesucht. Wenn ein Suchmuster bestehend aus einer Folge von Zeichen in zwei Klammern gesetzt wird, dann wird der Inhalt des Strings der mit diesem Suchmuster übereinstimmt in den Speicher geschrieben. Die Speichereinträge können bei Bedarf wieder mit "*$1*" bis "*$n*" entsprechend der Reihenfolge ihres Eintrags aus dem Speicher ausgelesen werden (s. auch im Beispiel unten). Für das Ersetzen von Texten wird vor das Suchmuster und dem ersten Schrägstrich ein "*s*" gesetzt

und nach dem zweiten Schrägstrich "/" dann in ähnlicher Weise die Ausgabe bzw. der Ersatz für die im String gefundenen Teile definiert.

3 Komplexes Beispiel

Angenommen es existiere eine Tabelle A bei der in einer Variablen *Name* sowohl eine beliebige Anzahl an Vornamen als auch der Nachname von Personen in der Reihenfolge "Nachname, Vorname1 Vorname2 ..." enthalten sind (siehe Abbildung 1), wobei die Zwischenräume mit beliebig vielen Leerzeichen gefüllt sein können. Die Aufgabe besteht nun darin, diese Variable in eine weitere Variable zu überführen, in der nur der erste Vorname gefolgt von dem Nachnamen enthalten sind[2]. Dies ließe sich noch mit relativ wenig Aufwand mit den üblichen SAS-Funktionen verwirklichen. Ungleich schwieriger wird es allerdings, wenn die Variable in Tabelle A neben den Vor- und Nachnamen auch noch beliebige Namenszusätze enthält. Die folgende Abbildung illustriert die Problemstellung und das geforderte Ergebnis:

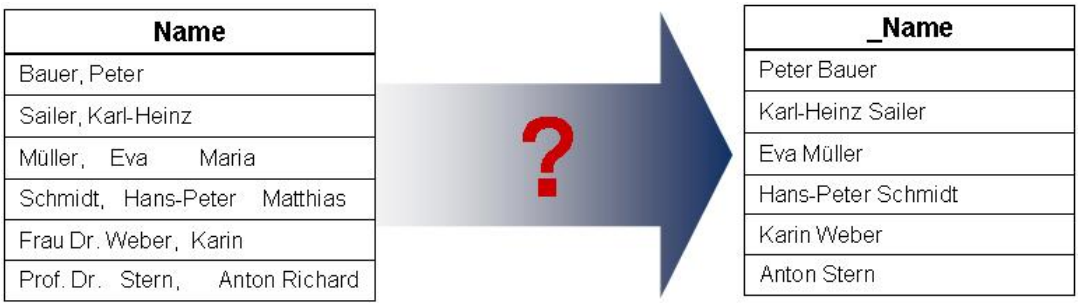

Abbildung 1: Problemstellung

Mit Perl Regular Expressions lässt sich das Problem mit wenigen Zeilen Code lösen:

```
DATA Ergebnis; SET Namen;
     pattern = PRXPARSE("s/(\w+), +(\w+-?\w+)/$2 $1/");
     CALL PRXSUBSTR(pattern, Name, Position, Laenge);
     match = SUBSTR(Name,Position,Laenge);
     _Name = PRXCHANGE(pattern,-1,match);
RUN;
```

Die schwierigste Aufgabe besteht dabei darin, sich genau zu überlegen, wie das entsprechende Muster zu den gesuchten Textbausteinen aussehen muss, damit diese eindeutig vom Rest des Strings unterschieden werden können. Schauen wir uns die Beschreibung des Musters innerhalb von PRXPARSE nun genauer an:

1) Das "*s*" am Anfang bedeutet, dass eine Ersetzung vorgenommen werden soll.

[2] Das Aufsplitten in zwei getrennte Variable für Vor- und Nachname sollte dann kein größeres Problem darstellen.

2) Jeder Nachname in Tabelle A beginnt mit einem Buchstaben (\w, s. oben):
   `s/(\w` ***S***chmidt, Hans-Peter Matthias
3) Dem ersten Buchstaben können beliebig viele weitere folgen:
4) `s/(\w+` ***Schmidt***, Hans-Peter Matthias
5) Durch die runden Klammern um diesen ersten Ausdruck wird ein gefundener Nachname in den Speicher an die Stelle $1 geschrieben:
   `s/(\w+)` → $1: Schmidt
6) Jedem Nachnamen folgt in der Tabelle ein Komma. Damit können die Nachnamen z.B. von den davorstehenden Namenszusätzen unterschieden werden:
   `s/(\w+),` ***Schmidt,*** Hans-Peter Matthias
7) Nach dem Komma kann eines oder beliebig viele Leerzeichen enthalten sein:
   `s/(\w+), +` ***Schmidt,___***Hans-Peter Matthias
8) Anschließend muss wieder ein Buchstabe bzw. beliebig viele Buchstaben folgen:
   `s/(\w+), +(\w+` ***Schmidt,___Hans***-Peter Matthias
9) Der erste Vorname kann (muss aber nicht) aus einem Namen mit Bindestrich bestehen (daher hier ein Fragezeichen):
   `s/(\w+), +(\w+-?` ***Schmidt,___Hans-***Peter Matthias
10) Nach einem Bindestrich (bzw. auch ohne Bindestrich) können wieder beliebig viele Buchstaben folgen:
    `s/(\w+), +(\w+-?\w+` ***Schmidt,___Hans-Peter*** Matthias
11) Der erste Vorname ist damit ebenfalls erfasst und wird durch die Klammern in den Speicher an Stelle $2 geschrieben:
    `s/(\w+), +(\w+-?\w+)` → $2: Hans-Peter
12) Mit dem nächsten Schrägstrich endet der Reguläre Ausdruck. Damit hat die Suchschablone nur dann Erfolg, wenn der Nachname gefolgt von einem Komma und dem ersten Vornamen genau entsprechend diesem Muster gefunden wurden. Sämtliche weitere Vornamen und auch die Namenszusätze werden durch das Suchmuster nicht abgedeckt.
13) Nach dem Schrägstrich wird nun definiert, durch was der gefundene Textteil des Strings ersetzt werden soll, in unserem Beispiel durch die verdrehte Reihenfolge der vorher in den Speicher geschriebenen Nach- und Vornamen:
    `s/(\w+), +(\w+-?\w+)/$2 $1/` → Ausgabe: Hans-Peter Schmidt

Damit ist das Suchmuster eindeutig definiert. Im obigen Code werden zunächst über PRXSUBSTR die Position und die Länge des Suchmusters im String ausgegeben (dazu verwendet PRXSUBSTR nur das Suchmuster aus PRXPARSE ohne den Ersetzungsteil). Dann wird mit einem normalen SUBSTR ein gefundenes Muster (also Nachname, Vorname) aus dem String herausgeschnitten und an eine neue Variable übergeben. Auf diese neue Variable wird dann noch einmal mit PRXCHANGE das definierte Suchmuster angewendet (was dann natürlich immer zu Treffern führt) und dann die abschließende Ersetzung vorgenommen.

4 Anwendungsmöglichkeiten

Die Anwendung der Perl Regular Expressions im Falle des obigen komplexen Beispiels zeigt wie flexibel einsetzbar und gleichzeitig wie wenig Code-intensiv die PRX-Funktionen sind. Damit eignen sich Perl Regular Expressions beispielsweise sehr gut für die Datenvalidierung. Ein weiteres Einsatzgebiet für Perl Regular Expressions ist generell das Suchen und Ersetzen von Zeichenfolgen innerhalb von Strings. Bei der Bearbeitung von Freitexten können Perl Regular Expressions verwendet werden, um relevante Informationen aus den Freitexten zu extrahieren. Außerdem eignen sie sich, um lange ungeordnete Text-Variablen in Einzelteile zu zerlegen und zu bearbeiten. Dies sei abschließend an einem weiteren Beispiel kurz gezeigt.

Problemstellung: Von einer Homepage im Internet wird mittels copy & paste eine umfangreiche Adressliste in ein Text-File übernommen. Diese Liste besteht aus einer ungeordneten Abfolge von Adress-Teilen wie Anschrift, Straße und Telefonnummern. Nach dem Import in SAS hat die Tabelle das folgende Aussehen:

Tabelle 1: Importiertes Text-File mit einer Abfolge von Adress-Beständen

text
Landratsamt Altötting
Telefon 08671-502-665
Gesundheitsamt Amberg
Hockermühlstr. 53
Tel 08621-39-399
Landratsamt Hof
Fax.: 09281-16873
Tel.: 09281-16-0
Gesundheitsamt Hamburg
21029 Hamburg
Tel.: 042891-2216
Fax.: 042891-2200
Stadt Ingolstadt
Telefon: 0841-3051-460
85049 Ingolstadt

In der Tabelle stehen nun verschiedene Institutionen untereinander wobei die Einträge für jede Institution in unterschiedlicher Reihenfolge vorliegen. Angenommen es sollen aus dieser Tabelle alle Telefonnummern extrahiert und in ein einheitliches Format der Form "Tel.: 0XXXX-XXXXX..." überführt und dabei nur die Telefonnummern mit Vorwahlen beginnend mit 08 oder 09 berücksichtigt werden, so lässt sich dies mit Perl Regular Expressions wie folgt bewerkstelligen:

```
DATA Ergebnis; SET Liste;
     _Telefon = "s/[Tel]\w+\.*:* +([0][8-9]\d\d\d-\d+)/Tel.: $1/";
     pattern = PRXPARSE(_Telefon);
     IF PRXMATCH(pattern,text) THEN DO;
         Ergebnis = PRXCHANGE(pattern,-1,text);
     END;
RUN;
```

Das Ergebnis ist in der folgenden Tabelle dem ursprünglichen Eintrag gegenüber gestellt:

text	**Ergebnis**
Landratsamt Altötting	
Telefon 08671-502-665	Tel.: 08671-502-665
Gesundheitsamt Amberg	
Hockermühlstr. 53	
Tel 08621-39-399	Tel.: 08621-39-399
Landratsamt Hof	
Fax.: 09281-16873	
Tel.: 09281-16-0	Tel.: 09281-16-0
Gesundheitsamt Hamburg	
21029 Hamburg	
Tel.: 042891-2216	
Fax.: 042891-2200	
Stadt Ingolstadt	
Tel.: 0841-3051-460	Tel.: 0841-3051-460
85049 Ingolstadt	

Literatur

[1] W. Herff, et.al. (2003): Tipps & Tricks für einen leichteren Umgang mit der BASE SAS Software. C. Becker, H. Redlich (Hrsg.), Proceedings der 7. Konferenz der SAS®-Anwender in Forschung und Entwicklung (KSFE)Shaker-Verlag, Aachen

[2] M. Kappler, et.al. (2005): Tipps & Tricks für einen leichteren Umgang mit der BASE SAS Software. Proceedings der 9. Konferenz der SAS®-Anwender in Forschung und Entwicklung
http://de.saswiki.org/images/a/ae/9.KSFE-2005-Kappler-Tipps-und-Tricks-für-den-leichteren-Umgang-mit-der-SAS-Software.pdf [Zugriff 16.03.2011]

[3] R. Cody, et.al. (2004): An Introduction to Perl Regular Expressions in SAS9. SUGI 29 Tutorials, Paper 265-29, Princeton, New Jersey
http://www2.sas.com/proceedings/sugi29/265-29.pdf [Zugriff 16.03.2011]