Zahnschema ohne Schmerzen (Eine Etüde in Zeichenketten)

Ralph Leonhardt
Versicherungskammer Bayern (VKB)
Warngauer Str. 30
81539 München
ralph.leonhardt@vkb.de

Zusammenfassung

Zur Überprüfung von Zahnbehandlungen muss der Versicherungsträger wissen, welcher Zahn behandelt wurde. Dies wird zwar im Zahnschema mittels eines Strings dokumentiert, aber korrektes Suchen in Strings ist alles andere als trivial. Um auch Sachbearbeitern dies zu ermöglichen, wird das Zahnschema in Indikatorvariablen umkodiert. Makrovariable Magie im Data Step und Zeichenkettenfunktionen machen es möglich.

Schlüsselwörter: Zeichenketten, Data Step, Makrovariablen, symget, symput

1 Worum es geht

Führt ein Zahnarzt eine Zahnbehandlung durch, so dokumentiert er die behandelten Zähne zur Abrechnung beim Versicherungsträger in einem Zahnschema. Um die Korrektheit der Abrechnung zu prüfen und eventuell auch Garantieleistungen zu erhalten (ja, Sie haben 3 Jahre Garantie auf Zahnersatz), muss das Zahnschema ausgewertet werden.
Bei Vorliegen einer einzelnen Rechnung ist das kein Problem, sollen jedoch die Leistungen mehrerer Jahre geprüft werden, da es nun mal Zahnbehandlungen über mehrere Jahre hinweg gibt, so müssen unter Umständen Tausende, ja sogar Millionen Abrechnungen verglichen werden.
Nun sind die Sachbearbeiter Spezialisten in der Abrechnung aber korrekte Abfragen von Zeichenketten, noch dazu für die Spezialitäten eines Zahnschemas, ist dann vielleicht doch in der Breite zuviel verlangt. Entsprechend steht man so vor der Aufgabe, ein Zahnschema in eine Folge von Indikatorvariablen umzukodieren, so daß diese dann mit Excel gefiltert oder mit einer Point&Click Software wie z.B. Sas Visual Analytics ausgewertet werden kann. Und nebenbei geht es auch um viel Geld für den Versicherten.

Den Prozess der Zerlegung einer Zeichenkette in bedeutungstragende Einheiten nennt man auf *compudenglisch parsen*, ein Programm, dass dies realisiert einen *Parser*. Im Folgenden wird zunächst die Mechanik des Zahnschemas erläutert, danach entwickeln wir eine Strategie zur Umsetzung dieser Aufgabe. Deren Programmierung in einem SAS-Makro wird im Anschluss daran im Detail dargelegt.

2 Zahnschema

Das FDI[1]- Zahnschema für Erwachsene sieht so aus: Es wird aus Patientensicht aufgeschrieben, wo also rechts steht ist Patienten-rechts und Zahnarzt-links. Jeder Zahn wird durch eine zweistellige Ziffernkombination beschrieben.

Das Gebiss eines Menschen ist zweiachsig symmetrisch aufgebaut, es ist sowohl frontal als auch horizontal symmetrisch. Es wird in vier Quadranten aufgeteilt, die von eins bis vier durchnummeriert sind. Die Zahnpositionen werden in jedem Quadranten, beginnend vom Frontzahn, von eins bis acht durchnummeriert. Diese Darstellung betont die Symmetrie des Gebisses und macht es intuitiver zu bezeichnen, andererseits wurde - so geht die Historie - die 9 und 0 zur Darstellung auf Lochkarten seinerzeit eingespart.

Für das Milchgebiss werden die Quadranten einfach von fünf bis acht weitergezählt. Aber, wie bereits bemerkt, hat das Milchgebiss in jedem Quadranten nur fünf Zähne.

Es ist üblich bei Behandlungen, die mehrere nebeineinander stehende Zähne betrifft, zwischen den ersten und letzten behandelten Zahn einen Bindestrich zu schreiben. Leerzeichen sind irrelevant. Eine bestimmte Reihenfolge der Zahnstellungen ist nicht vorgegeben, d.h. 23-25 und 25-23 sind gleichwertige Darstellungen. Damit haben wir im Grunde den Zeichenvorrat für einen Zahnschemaparser vollständig beschrieben als: `{12345678,-}`. Zu beachten wäre, dass ein gültiges Zahnschema stets mit einer Ziffer aus `{12345678}` beginnt.
Beispiel: `18-14,11-23,31,32`

In diesem Beispiel wären die Zähne eines Erwachsenengebisses behandelt, betroffen sind die Zähne 18, 17, 16, 15, 14, 11, 21, 22, 23, 31, 32.

3 Strategie

Nach Absprache mit dem Fachbereich wurde vereinbart, für jeden Zahn des Zahnschemas Indikatorvariablen zu definieren, die mit ’X‘ markiert sind, wenn der Zahn betroffen ist, sonst aber die leere Zeichenkette ‘ ‘ enthält.

Der Sinn dieser Anforderung ist, dass man bei der Prüfung einen einzelnen Zahn für alle Behandlungen selektieren kann. So kann auf mehrfache Behandlungen eines oder mehrerer Zähne geprüft werden. Idealerweise sind auch daraus aggregierte Variablen ableitbar, wie z.B. Quadranten.

Beispiel: `18-14,11-23,31,32`
wird zu (Zeilenumbrüche dienen hier nur der Darstellung im Artikel):

[1] Fédération Dentaire Internationale - Zahnärzteweltverband

Z18	Z17	Z16	Z15	Z14	Z13	Z12	Z11
X	X	X	X	X			X
Z21	Z22	Z23	Z24	Z25	Z26	Z27	Z28
X	X	X					
Z38	Z37	Z36	Z35	Z34	Z33	Z32	Z31
						X	X

4 Umsetzung

Einige kurze Bemerkungen bevor ich an die Umsetzung gehe: Viele Zwischenergebnisse lege ich in Variablen ab, die ich später aber nicht lösche. Die Motivation zu diesem, zugegebenermaßen recht verschwenderischen Vorgehen, ist zweierlei. Zum ersten erleichtert es die Fehlersuche, da jeden Monat mehrere hundertausend neue Datensätze einlaufen, die möglicherweise neuartige Fehlerkonditionen erzeugen. Zum zweiten präventive Faulheit. Wer weiß was man später zur Weiterentwicklung benötigt.

In der Darstellung bespreche ich nicht jede Zeile des Makros, sondern nur die entscheidenden Ideen. Im Anhang A Das Makro %zahnparser“ ist der gesamte Quelltext des Makros zu finden.

4.1 Makrodefinition - das Gesicht zur Welt

Der Zahnparser ist Teil eines größeren Makros und kapselt im Sinne der strukturierten Programmierung die Funktionalität des Zahnschemaparsers.

Die Funktionalität soll durch die Makrodefinition möglichst gut wiedergespiegelt werden und Seiteneffekte (d.h. nicht offensichtliche Änderungen der zugrunde liegenden Tabelle) sollen möglichst minimiert oder vermieden werden.

Um das Zahnschema anzuwenden, wird der Tabellenname benötigt, in der die Zahnschemavariable steht, sowie der Name der Zahnschemavariablen. Sinnigerweise nenne ich diese Variablen dann `table` und `variable`. Diese sind Positionsparameter.

Eine gewisse Variabilität soll durch die Angabe möglicher Alternativen zum Trennzeichen `&delim` ‚,‘ ermöglicht werden, ebenso für den Listentrenner `&ldel`‘-‘. Weiterhin ist es sinnvoll, einen Schalter zu haben, der beim *debuggen* hilft und ggf. Zwischener-

gebnisse beibehält, anstatt diese zu löschen. Hier sollten aber nach Möglichkeit sinnvolle *defaults* angegeben werden. Daher sind benannte Parameter mit entsprechenden Zuweisungen. Beim Aufruf im Programm kann man dann auf diese verzichten

```
%macro zahnparser(table, variable, delim=",", ldel="-" , debug= );
…
%mend zahnparser;
```

Der Aufruf erfolgt dann mit
`%zahnparser(zahngesamt, zahnschema).`

4.2 Vorverarbeitung

Vor der eigentlichen Bearbeitung nehme ich eine gewisse Normalisierung der Zeichenketten, die das Zahnschema enthalten, vor. Mit der Funktion
`compress(&variable," " )` werden hierfür alle Leerzeichen entfernt.

Leider ist die Zeichenerkennung beim Einscannen der Rechnungen nicht perfekt, so dass vor der eigentlichen Zerlegung der Zeichenkette eine Überprüfung des verwendeten Zeichenvorrats erforderlich ist. Das erlaubt, die Fehler abzufangen. Andernfalls würde der Prozess abbrechen.

Anstatt nun jedes Zeichen einzeln mit verschachtelten Schleifen zu testen, bietet SAS hier eine Abkürzung mit der Funktion `verify(&variable,"012345678-,")` an. Dabei überprüft `verify`, ob die Variable `&variable` nur die Zeichen des zweiten Arguments,`"012345678-,"`, enthält. Ist dies nicht der Fall, wird die Stelle der Zeichenkettenvariablen ausgegeben, an der ein abweichendes Zeichen steht, ansonsten eine 0 (Null). Die Zeichenkette muss in Hochkommas eingeschlossen werden.

Im Makro gibt es zwei unterschiedliche Tests: Der erste Test überprüft die ersten zwei Stellen Variablen. Es dürfen dort nur Ziffern von 1 bis 8 stehen:

```
_verify_start=verify(
          substr(trim(&variable),1,2),'12345678');
```

Wird hier kein Fehler ausgeworfen, so testen wir den gesamten String:

```
_verify=verify(trim(&variable),'123456789-,');
```

In den Variablen `_verify` bzw. `_verify_start` wird das Testergebnis abgelegt. Deren Inhalt wird dann ein eine Fehlervariable transformiert:

```
if _verify_start>0 then do;
     zfehler=1;
end;
```

Man beachte die `trim` - Funktion, die führende und folgende Leerzeichen entfernt. Bei der Anwendung von Stringfunktionen auf Variablen ist das immer eine gute Idee!

Anhand der Ergebnisse wird eine fehlerhafte Zeichenkette durch ein Leerzeichen ersetzt:

```
data _tpattern_1;
set _tpattern;
if zfehler >0  then &variable='';
run;
```

4.3 Das Variableninterface im Datastep

Bekanntlich kann in einem Datastep nicht direkt auf Makrovariaben zugegriffen werden. Hierfür benötigt man die Funktionen `symput, symget` welche in einem Datastep Werte in die Symboltabelle schreiben können bzw. diese lesen.

`symput` ist dabei zweiparametrig: der erste Parameter gibt den Wert an, der geschrieben werden soll, der zweite ist ein String, der den Namen der Variablen angibt, in die der Wert des ersten Arguments geschrieben wird. Der Variablenname muss in Hochkommas eingeschlossen werden.

Derartige Konstrukte werden häufig in einem Nullstep `data _NULL_` durchgeführt, in diesem Fall jedoch machen wir alles in einem Data Step.

4.4 Symput, symget, Makroschleifen

Nachdem die Vortests auf den korrekten Zeichenvorrat des Strings durchlaufen sind, erstelle ich als Muster die Tabelle `zahnschema` mit den entsprechenden Variablen in dem `retain` unter extensiver Nutzung der automatischen Bereichsdefinition von SAS. So stelle ich sicher, dass die Zahnvariablen auch alle nebeneinander in der gewünschten Reihenfolge stehen.

Danach erzeuge ich die Makrovariablen in der Symboltabelle des SAS Programms. Der Grund für diesen Schritt ist, dass ich sicher gehen will, dass das spätere Auslesen der Variablen mittels `symget` nicht fehlschlägt und zu einem Programmabbruch führt.

Die Variablen werden dabei über verschachtelte Schleifenkonstruktionen als leere Zeichenkette vorbelegt. Man beachte bitte die Verwendung der Hochkommata. Einfache Hochkommata würden die Interpolation der Variablen mit den Werten verhindern!

```
data zahnschema ;
set _tpattern_1;
retain z11-z18 z21-z28 z31-z38 z41-z48 z51-z55 z61-z65 z71-z75 z81-
z85 '';
/* Definition*/
%do j=1 %to 8;
     %if &j<5 %then %do;
         %do i=1  %to 8;
```

```
            call symput('z'||"&j&i", '');
          %end;
     %end;
     %else %do;
          %do k=1  %to 5;
               call symput('z'||"&j&k", '');
          %end;
     %end;
%end;
```

Und natürlich sollte man beachten, dass ja eine Krankenversicherung von der Wiege bis zur Bahre einsteht. Der Zahnschemaparser muss also sowohl das Erwachsenen- als auch das Kinderzahnschema verstehen. Nach diesem Teil des Datasteps enthält also der Datensatz die benötigten Variablen und in der SAS-Symboltabelle sind die notwendigen Makrovariable als leere Zeichenkette vorhanden.

4.5 An die Arbeit!

Die *tools of trade* für die jetzt erfolgende Verarbeitung des Zahnschemas sind die Stringfunktione `scan` und `index`

Mit

```
i=1;
do while (scan(&variable,i, &delim) ne '');
…
end;
```

wird die eigentliche Verarbeitung des Strings eröffnet. Dabei wird die Stringvariable auf den Trenner `&delim` solange untersucht, bis diese leer ist. Und solange dies nicht der Fall ist wird die Schleife durchlaufen.
Die Iteratorvariable i dient dazu, die Elemente des Zahnschemastrings, die durch die Funktion `scan` gefunden wurden, durchzunummerieren. Beim Hochzählen wird immer das nächste Teilstück nach `&delim` analysiert.

Als erstes, wenn also das Zahnschema verarbeitet werden soll, prüfe ich ob hier eine Liste der Form `11-23` vorliegt. Und die liegt per definitionem genau dann vor wenn ein `&ldel` in der Variable gefunden wird:

```
if find(scan(&variable,i, &delim), &ldel.)>0 then do;
…
end;
```

Der so gefundene Listenteilstring des Zahnschemas wird am Listentrenner `&ldel` zerlegt in den Anfang `&slist` und das Ende `&elist`. Da dies aber Makrovariablen in einem Datastep sind, müssen diese mit `call symput` in die Symboltabelle geschrieben werden. Da bleiben sie aber nicht lange, sondern werden sofort als `l` und `r`, für links und rechts in Variablen geschrieben. Zur Sicherheit überprüfen wir nochmal ob

das auch zwei Ziffern sind, wenn nicht verabschieden wir uns mit `leave,` werfen jedoch in die Variable `zfehler` den Fehler 2 aus.

Und jetzt müssen schmerzhafte Fallunterscheidungen gemacht werden, abhängig davon, ob die Zähne im gleichen Quadranten liegen - also ob die erste Ziffer des Zahnschemas links von `&ldel` gleich der ersten Ziffer rechts von `&ldel` ist.

Jetzt macht auch das Einlesen der Listenteile in den Datensatz Sinn, denn man kann einfach mal arithmetisch vorgehen um den Quadranten zu bestimmen:

```
q1=int(l/10);
q2=int(r/10);
quad=q1-q2;
```

Wenn Listenanfang und Listenende im selben Quadranten liegen, ist `quad` 0. Jetzt muss ich noch die Reihenfolge herausfinden, in der die Listen angegeben sind. In Abhängigkeit davon startet die Schleife zur Variablenbelegung mit der linken `l` oder rechten `r` Listenbegrenzung:

```
if quad=0 then do;
     /* Jede Reihenfolge ist möglich*/
     if l-r <0 then do;
          diff=0; zpos=l-r;
               do pos=l to r;
                    call symput('z'||compress(pos,' '), 'X');
               end;
     end;
     else do;
          diff=0; zpos=l-r;
               do pos=r to l;
                    call symput('z'||compress(pos,' '), 'X');
               end;
     end;
end;
```

Wenn Listenanfang und Listenende im gleichen Quadranten liegen, kann man gleich mit der Zuweisung beginnen, einfach indem man durchzählt. Ist er es nicht, wird es noch mal hart.

Um hier die Zahnstellen durchiterieren zu können, muss für den jeweiligen Quadranten der Frontzahn künstlich erzeugt werden. Selbstverständlich beginnt dieser Teil mit `else do;`.

```
/* es wird immer vom Frontzahn aus gezählt*/
     else do;
     /* Definieren der Frontzähne*/
     /* es wird immer vom Frontzahn aus gezählt*/
          q1base=q1*10+1;
          q2base=q2*10+1;
```

Von diesem wird dann bis zur jeweils angegebenen Zahnstelle hochgezählt.

```
     if l-q1base <0 then do;
                    diff=1; zpos=l-q1base;
                    do pos=l to q1base;
                         call symput('z'||compress(pos,' '), 'X');
                    end;
               end;
```

Zwei Techniken kennen wir bereits: es wird noch einmal wird die Reihenfolge der Zahnpositionen verglichen, diesmal aber mit `qbase`, damit korrekt hochgezählt werden kann. Die zweite ist das Schreiben in die Makrovariablen. Dies wiederholt sich dann entsprechend für den anderen Quadranten. Das ist zwar etwas länglich, aber im Grunde völlig analog der bisher verwendeten Logik. Am Ende nicht vergessen den Iterator `i` hochzuzählen, damit sich die durch `scan` erzeugte Liste auch weiter durchlaufen wird.

4.6 Abholen der Ergebnisse

Bis hierher diente die ganze Kunst nur dem korrekten Schreiben in die am Anfang des Data Steps verwendeten Makrovariablen. Jetzt bleibt uns nur noch, diese abzuholen und in die Variablen des Datensatzes zu schreiben. Das passiert völlig erwartbar:

```
%do j=1 %to 8;
     %*put &j;
     %if &j<5 %then %do;
         %do i=1  %to 8;
          /*  %put Zweig1 &j &i; */
            z&j&i= symget('z'||"&j&i");
          %end;
     %end;
     %else %do;
          %do i=1  %to 5;
           /* %put Zweig2 &j &i; */
            z&j&i= symget('z'||"&j&i");
          %end;
     %end;
%end;
```

Auch hier wieder der SAS-typische Doppelindex `&j&i`. Bleibt nur noch `run;` zu tippen.

4.7 Aufräumen

Am Ende werden einige Hilfstabellen aufgeräumt. Über die Variable `&debug` wird dies gesteuert. Hier geht sicher noch etwas mehr. Aber es ist an dieser Stelle gut genug.

5 Zusammenfassung

SAS bietet ein reichhaltiges Inventar an Zeichenkettenfunktionen an. Es lohnt sich, sich ein wenig Zeit zu Nachforschen zu nehmen. Die Umsetzung der Zeichenkette für das Zahnschema in Variablen ist nicht ganz trivial. Welche Komplexität man jedoch in einem Data Step kapseln kann, ist immer wieder überraschend. Das Pärchen `symput/symget` erlaubt es dabei, Variablen Inhalte via Makrovariablen aufzubewahren und wie ein Kaninchen aus der Tasche zu zaubern. Und ein wenig Magie benötigen wir alle in unserem Arbeitsalltag.

Anhang A Das Makro %zahnparser

```
%macro zahnparser(table, variable, delim=",", ldel="-" , debug= );

/* Testen auf Korrektheit Zahnschema*/

%put NOTE: Zahnschema Verifikation;

data _tpattern;
set &table;
zfehler=0;
&variable=compress(&variable,' ');
if &variable ne '' then do;
     /* sind die ersten beiden zeichen zahlen*/
    if length(trim(&variable))>1 then do;

     _verify_start=verify(substr(trim(&variable),1,2),'12345678');
          /* kommen noch andere zeichen als zahlen komma oder binde-
strich vor*/
          if _verify_start>0 then do;
               zfehler=1;
          end;
          else do;
               _verify=verify(trim(&variable),'123456789-,');
               if _verify then zfehler=1;
          end;
     end;
     else zfehler=1;
end;
run;

/* Vorbereiten Verarbeitung */
%put NOTE: Pos Zahn ergänzen;
data _tpattern_1;
set _tpattern;
if zfehler >0  then &variable='';
run;
```

```
data zahnschema ;
set _tpattern_1;
retain z11-z18 z21-z28 z31-z38 z41-z48 z51-z55 z61-z65 z71-z75 z81-
z85 '';
/* Definition*/
%do j=1 %to 8;
     %if &j<5 %then %do;
         %do i=1  %to 8;
            call symput('z'||"&j&i", '');
          %end;
     %end;
     %else %do;
          %do k=1  %to 5;
               call symput('z'||"&j&k", '');
          %end;
     %end;
%end;

/* Verarbeitung Zahnschemaliste */

%put NOTE: Parsen Zahnschema;
i=1;
do while (scan(&variable,i, &delim) ne '');
     if find(scan(&variable,i, &delim), &ldel.)>0 then do;
          call symput('slist',scan(scan(&variable,i, &delim),1,
&ldel));
          call symput('elist',scan(scan(&variable,i, &delim),2,
&ldel));
          l=symget('slist');
          r=symget('elist');
          if verify(trim(l),'123456789') or veri-
fy(trim(r),'123456789')  then  do;
               zfehler=2;
               leave;
          end;
          /* Ermittlung ob gleicher Quadrant vorliegt */
          q1=int(l/10);
          q2=int(r/10);
          quad=q1-q2;
          /* Gleicher Quadrant*/
          if quad=0 then do;
          /* Jede Reiehenfolge ist möglich*/
               if l-r <0 then do;
                    diff=0; zpos=l-r;
                    do pos=l to r;
                         call symput('z'||compress(pos,' '), 'X');
                    end;
               end;
               else do;
                    diff=0; zpos=l-r;
                    do pos=r to l;
                         call symput('z'||compress(pos,' '), 'X');
```

```
                    end;
               end;
          end;
          /* unterschiedliche Quadranten in der Liste */
          else do;
          /* Definieren der Frontzähne*/
          /* es wird immer vom Frontzahn aus gezählt*/
               q1base=q1*10+1;
               q2base=q2*10+1;
          /* erster Quadrant*/
               if l-q1base <0 then do;
                    diff=1; zpos=l-q1base;
                    do pos=l to q1base;
                         call symput('z'||compress(pos,' '), 'X');
                    end;
               end;
               else do;
                    diff=1; zpos=l-q1base;
                    do pos=q1base to l;
                         call symput('z'||compress(pos,' '), 'X');
                    end;
               end;
          /* zweiter Quadrant*/
               if r-q2base <0 then do;
                    diff=1; zpos=l-q2base;
                    do pos=l to q2base;
                         call symput('z'||compress(pos,' '), 'X');
                    end;
               end;
               else do;
                    diff=1; zpos=l-q2base;
                    do pos=q2base to r;
                         call symput('z'||compress(pos,' '), 'X');
                    end;
               end;
          end;
     end;
     else do;
          call symput('z'||scan(&variable,i, &delim), 'X');
     end;
     i=i+1;
end;

/* Abholen der Ergebnisse*/
%do j=1 %to 8;
     %*put &j;
     %if &j<5 %then %do;
         %do i=1  %to 8;
          /*  %put Zweig1 &j &i; */
            z&j&i= symget('z'||"&j&i");
          %end;
     %end;
```

```
     %else %do;
          %do i=1  %to 5;
           /* %put Zweig2 &j &i; */
            z&j&i= symget('z'||"&j&i");
          %end;
     %end;
%end;
run;

%if &debug ne %then %do;
%end;
%else %do;
%put Aufräumen;
proc datasets NOLIST NOWARN ;
    delete _tpattern _tpattern_1 ;
quit ;
%end;

%mend zahnparser;
```