Rechnen mit der Wahrheit und If 0 Then Set

Stefan Beimel
Merz Pharmaceuticals GmbH
Eckenheimer Landstr. 100
60318 Frankfurt am Main
stefan.beimel@merz.de

Zusammenfassung

SAS speichert intern die Wahrheitswerte TRUE und FALSE als numerische Variable mit den Ausprägungen 1 bzw. 0 ab. Das kann man sich zunutze machen, um Code zu vereinfachen.
Das scheinbar sinnlose Konstrukt IF 0 THEN SET ... kann genutzt werden, um Variablen zu definieren, während ich an den Observations nicht das geringste Interesse habe.

Schlüsselwörter: Datenschritt, Boolesche Variablen, Kompilierungsphase, Ausführungsphase

1 Rechnen mit der Wahrheit

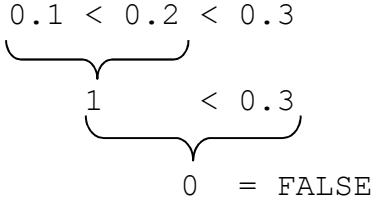
SAS kennt keine Booleschen Variablen, also Variablen, die die Wahrheitswerte TRUE oder FALSE darstellen. Die „Wahrheit“ wird über Zahlen definiert, wobei gilt:

- Missing values (auch spezielle fehlende Werte wie .A) und 0 sind ‘falsch’ (FALSE)
- Alle anderen Zahlen sind ‘wahr’ (TRUE).

Ergebnisse von Vergleichsoperationen werden als 1 (TRUE) bzw. 0 (FALSE) ausgegeben bzw. zwischengespeichert. Einige Beispiele sollen verdeutlichen, welche Auswirkungen dieses Verhalten hat, zumal es nicht immer intuitiv ist.

Der Ausdruck `0.1 < 0.2 < 0.3` wird in der Datenschritt-Programmierung als TRUE erkannt. Das entspricht der Intuition. Allerdings wird dieser Ausdruck vor der Bewertung intern umgewandelt in `(0.1 < 0.2) and (0.2 < 0.3).`

Der gleiche Ausdruck ist in der Makrosprache FALSE, da diese Umwandlung nicht stattfindet und folgendermaßen schrittweise von links abgearbeitet wird:



Einige andere, ähnliche Beispiele mit teilweise überraschendem Ergebnis:

```
(0.1 < 0.2) < 0.3       => FALSE
 0.1 < (0.2 < 0.3)      => TRUE
(1 < 2) < 3             => TRUE
```

Des Weiteren macht SAS keinen Unterschied zwischen dem Zuweisungsoperator und dem Gleichheitsoperator, beides wird durch das Gleichheitszeichen dargestellt.
Folgende seltsam anmutende Konstruktion ist daher möglich und syntaktisch völlig einwandfrei:

```
a = 1 = 2;
```

Zuerst wird der Vergleich angestellt: 1 = 2 ist falsch, deshalb ergibt dieser Ausdruck 0. Diese 0 wird a zugewiesen. Die Variable a hat also am Ende den Wert 0!

Dieses Verhalten kann man sich zunutze machen, um z. B. IF-Anweisungen anders darzustellen.

```
if age > 50 then age_group = 2;
            else age_group = 1;
```

kann geschrieben werden als

```
age_group = 2 * (age > 50) + 1 * (age <= 50);
```

Von der Richtigkeit dieser Konstruktion überzeugt man sich am besten am Beispiel:

```
Age=60   =>   age_group= 2 * 1 + 1 * 0 = 2
Age=40   =>   age_group= 2 * 0 + 1 * 1 = 1
```

So kann man Programme kürzer und mit weniger Anweisungen schreiben. Allerdings ist diese Art der Programmierung nicht selbsterklärend und sollte unbedingt kommentiert werden.

1.1 Beispiel: Berechnung des Alters

Das Alter eines Menschen in Jahren berechnet man intuitiv, indem man das derzeitige Jahr von Geburtsjahr abzieht. Danach überprüft man, ob derjenige in diesem Jahr schon Geburtstag hatte – wenn ja, wird noch 1 abgezogen.
SAS bietet leider keine Funktion, die das Alter exakt berechnet. Auch die Funktion YRDIF gibt nicht immer das richtige Resultat, siehe http://support.sas.com/ /kb/3/036.html.

Mit ‚konventionellem' SAS Code würde man programmieren:

```
BDate = '05Jun1967'd;
Today = today();

Age = year(Today) - year(BDate);

*** 1 abziehen, wenn in diesem Jahr noch kein Geburtstag war **;

if  month(Today) < month(BDate) or
    month(Today) = month(BDate) and day(Today) < day(BDate)

    then Age = Age - 1;
```

Dies hat den Nachteil, dass es, eingebaut in ein Makro, nicht wie eine einfache Funktion verwendet werden kann. Wandelt man die IF-Bedingung um und berechnet das Alter mit

```
Age = year(Today) - year(BDate) -
      (month(Today) < month(BDate) or
       month(Today) = month(BDate) and day(Today) < day(BDate)
      );
```

dann kann das Makro in einer einfachen Zuweisung benutzt werden, also:

```
Age = %Age(BDate, Today)
```

1.2 Beispiel: Gruppenmerkmale

Häufig benötigt man in der Statistik Kennzahlen, die pro Gruppe berechnet werden müssen, z. B. für Mittelwertsdifferenzen zweier Gruppen (hier die rote und blaue Gruppe):

$$\text{MeanDiff}_{\text{blau-rot}} = \text{Summe}_{\text{blau}} / \text{Anzahl}_{\text{blau}} - \text{Summe}_{\text{rot}} / \text{Anzahl}_{\text{rot}}$$

Die Datenstruktur ist dabei oft vertikal, d. h. die Gruppenzugehörigkeit ist in einer eigenen Variablen gespeichert (siehe Datensatz VALUES in Abbildung 1).
Mit der Prozedur SQL und der 0-1-Logik gelingt es nun in einem Schritt, die Anzahlen und Summen in eine Beobachtung zu bringen:

```
proc sql;
  create table sum as
  select
      sum(group='rot' )          as n_rot
     ,sum(group='blau')          as n_blau
     ,sum(value*(group='rot' ))  as sum_rot
     ,sum(value*(group='blau'))  as sum_blau
  from values
  ;
```

Die Screenshots verdeutlichen diese ‚interne' Entwicklung des Ausgabedatensatzes:

VIEWTABLE: Work.Values

	group	value	(group='rot')	(group='blau')	value * (group='rot')	value * (group='blau')
1	rot	0.4	1	0	0.4	0
2	rot	0.7	1	0	0.7	0
3	rot	2.1	1	0	2.1	0
4	rot	1.9	1	0	1.9	0
5	rot	0.6	1	0	0.6	0
6	blau	0.3	0	1	0	0.3
7	blau	0.8	0	1	0	0.8
8	blau	2	0	1	0	2
9	blau	0.9	0	1	0	0.9
10	blau	1.6	0	1	0	1.6
11	blau	0.5	0	1	0	0.5

VIEWTABLE: Work.Sum

	n_rot	n_blau	sum_rot	sum_blau
1	5	6	5.7	6.1

Abbildung 1: Datensatz VALUES und Ausgabendatensatz SUM

1.3 Beispiel: Gruppeneigenschaften

Aufgabe sei es, diejenigen Beobachtungen aller Gruppen zu behalten, bei denen wenigstens eine Beobachtung eine bestimmte Eigenschaft hat.

Zum Beispiel sollen beim Datensatz VALUES (s. Abb. 1) alle Gruppen behalten werden, bei denen die Variable VALUE in irgendeiner Beobachtung größer als 2 ist. Das ist bei der roten Gruppe der Fall.

```
proc sql;
   create table values2 as
   select *
   from values
   group by group
   having max(value > 2.0);
```

VIEWTABLE: Work.Values

	group	value	value > 2	max(value > 2)
1	rot	0.4	0	1
2	rot	0.7	0	1
3	rot	2.1	1	1
4	rot	1.9	0	1
5	rot	0.6	0	1
6	blau	0.3	0	0
7	blau	0.8	0	0
8	blau	2	0	0
9	blau	0.9	0	0
10	blau	1.6	0	0
11	blau	0.5	0	0

2 IF 0 THEN SET: Der Unterschied zwischen Kompilierung und Ausführung

In einigen Vorträgen auf der KSFE, z. B. [1], wurde bereits auf den Unterschied zwischen „Compilation" (Kompilierungsphase) und „Execution" (Ausführungsphase) eingegangen. Dieses Kapitel zeigt ein Beispiel, wie man sich diesen Unterschied zunutze machen kann.

In der Kompilierungsphase wird der Program Data Vector (PDV) erzeugt. Hierzu werden, vereinfacht gesagt, alle Variablen erzeugt und ihre Eigenschaften wie Name, Datentyp, Länge und Label festgelegt in der Reihenfolge ihres Bekanntwerdens im Datastep. In der Ausführungsphase werden die Daten der Eingabedatensätze gelesen und alle ausführbaren Anweisungen abgearbeitet.

Die Aufgabe, deren Lösung im Folgenden gezeigt wird, besteht darin, Metadaten eines Datensatzes zu benutzen. Die Beobachtungen interessieren nicht. Die eigentlichen Daten werden aus anderen Quellen gelesen.

Der Datensatz VARIABLEN soll für die kommenden Beispiele die Metadaten enthalten:

```
data Variablen;
    length brthdat $8;
    label brthdat = 'Date of Birth';
    ...
run;
```

2.1 Einlesen der Daten aus einem zweiten Datensatz

Zunächst soll der Fall betrachtet werden, wenn die Daten bereits in einem Datensatz vorhanden sind:

```
data Werte;
    x = 8; y =  7; output;
    x = 1; y = 12; output;
run;
```

1. Idee (schon ziemlich gut)

```
data Neu;

    set Variablen(where=(0))
        Werte;

run;

NOTE: There were 0 observations read from the data set WORK.
VARIABLEN.
      WHERE 0 /* an obviously FALSE WHERE clause */ ;
NOTE: There were 2 observations read from the data set WORK.WERTE.
NOTE: The data set WORK.NEU has 2 observations and 3 variables.
```

Beide Datensätze werden während der Ausführungsphase gelesen, und nur die Observations des zweiten Datensatzes werden in den Ausgabedatensatz geschrieben. Damit ist die Aufgabe erfüllt. Noch eleganter ist es, wenn der Datensatz VARIABLEN nur während der Kompilierungsphase genutzt wird, um den PDV zu erstellen, und während der Ausführungsphase ignoriert wird:

2. Idee (noch besser):

```
data Neu;

    if 0 then set Variablen;
    set Werte;

run;

NOTE: There were 2 observations read from the data set WORK.WERTE.
NOTE: The data set WORK.NEU has 2 observations and 3 variables.
```

Im Log kann man erkennen, dass der Datensatz VARIABLEN während der Ausführungsphase nicht gelesen wird.

2.2 Einlesen der Daten im Datastep

Nun soll der Fall betrachtet werden, wenn die Beobachtungen innerhalb eines neuen Datenschritts erzeugt werden, z. B. mittels einer Schleife:

```
data Werte;
    ...
    do brthdat = '01Jan1960'd to '31Dec2020'd;
       output;
    end;
run;
```

1. Idee (die schon ziemlich gute Idee von oben ist jetzt nicht mehr gut):

```
data Werte;
    set Variablen(where=(0));

    do brthdat = '01Jan1960'd to '31Dec2020'd;
         output;
    end;

run;
```

```
NOTE: There were 0 observations read from the data set WORK.
VARIABLEN.
      WHERE 0 /* an obviously FALSE WHERE clause */ ;
NOTE: The data set WORK.WERTE has 0 observations and 1 variables.
```

Der Ausgabedatensatz enthält keine Beobachtungen! Der Datenschritt wird nicht ausgeführt, weil SET keine Beobachtungen liefert.

2. Idee (wieder besser):

```
data Werte;
    if 0 then set Variablen;

    do brthdat = '01Jan1960'd to '31Dec2020'd;
         output;
    end;

run;

NOTE: The data set WORK.WERTE has 22281 observations and 1
variables.
```

Das ist die erwartete Anzahl der Beobachtungen, und die Variable BRTHDAT hat die Attribute aus dem Datensatz VARIABLEN geerbt.

Literatur

[1] Gigic, B., Deckert, A.: SAS Backstage. Proceedings der 16. KSFE Dresden. Shaker-Verlag, 2012, S. 103-111.