Erstellen flexibler Makros am Beispiel einer SQL-Routine

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

Zusammenfassung

SAS-Makros eignen sich u.a. besonders dafür, häufig wiederkehrende komplexe Prozesse zu kapseln und damit die erforderliche Funktionalität für verschiedene Variationen der Daten wiederholt zur Verfügung zu stellen, ohne dass die komplette Syntax jedes Mal neu programmiert werden muss. Der Anwender muss dann lediglich die Schnittstelle des Makros bedienen. Oftmals sind solche Makros jedoch nicht komplett situations-flexibel programmiert, d.h. eine kleine Abweichung z.B. in den Anforderungen an die Funktionalität einer Prozedur innerhalb des Makros kann vom Makro nicht abgedeckt werden. Anhand eines einführenden Beispiels werden hier die Prinzipien der bedingten Makro-Programmierung vorgestellt. Anschließend wird eine vom Autor auf der KSFE 2011 vorgestellte Matching-Methode für Fall-Kontroll-Zuweisungen in epidemiologischen Studien in einem Makro derart verallgemeinert, dass damit verschiedenste Matching-Situationen abgedeckt werden können. Kernstück ist hierbei eine SQL-Routine. Das Makro erzeugt dabei vor der Kompilierung situationsabhängig die notwendigen Code-Zeilen innerhalb des SELECT- und WHERE-Statements selbst.

Schlüsselwörter: Matching, Makro-Programmierung, PROC SQL, Makro-Bedingung

1 Einführendes Beispiel

Häufig wird dieselbe Prozedur mehrmals z.B. innerhalb einer Auswertung einer großen klinischen Studie verwendet, wobei sich die einzelnen Prozedur-Aufrufe dann nur im verwendeten Datensatz, kleinen Änderungen bei den Variablen und den Optionen unterscheiden. Als Beispiel sei hier die Prozedur PROC MEANS aufgeführt, die innerhalb einer Auswertung in verschiedenen Variationen verwendet wird:

```
PROC MEANS DATA=tab1 MAX SUM;
   VAR var1;
   OUTPUT OUT=test SUM=sum1 MIN=min1 MAX=max1;
RUN;
PROC MEANS DATA=tab2 MIN MAX MEAN MEDIAN;
   VAR var1 var2;
   OUTPUT OUT=test SUM=sum1;
RUN;
```

Diese Prozedur kann nun zur wiederholten Verwendung relativ einfach in ein Makro überführt werden:

```
%MACRO mean(Tab=,VarList=,Stats=);
    PROC MEANS DATA=&Tab. &Stats.;
        VAR &VarList.;
        OUTPUT OUT=test;
    RUN;
%MEND mean;
```

Ein Makro-Aufruf könnte nun folgendermaßen aussehen:

```
%mean(Tab = Tab1,VarList = var1 var2, Stats = sum min max)
```

Vergleicht man nun aber den Inhalt (und Output) des Makros genauer mit den obigen Prozeduren, so sieht man, dass das Makro nur bis zu einem bestimmten Grad die gleiche Funktionalität aufweist wie die Prozeduren. Die Angabe der Statistiken nach OUTPUT OUT kann im Makro nicht wie oben erfolgen, da der Code selbst direkt von der Anzahl der Statistiken in der Eingabevariablen "Stats" abhängt. Hier muss also mehr Programmieraufwand betrieben werden, um das Makro so flexibel zu gestalten, dass es die volle Funktionalität wie die einzelnen Prozeduren für verschiedene Situationen bereitstellen kann.

2 Wiederholung Makro-Grundlagen

Für das bessere Verständnis der nachfolgenden Programmierschritte werden hier kurz einige wichtige Grundlagen der Makro-Programmierung wiederholt.
Makros können quasi-objektorientiert programmiert werden in dem Sinne, dass eine Kapselung des Codes erfolgt und die Interaktion mit dem Anwender nur über die Schnittstelle erfolgt. Die Funktionsweise des Makros selbst muss dem Anwender für die korrekte Anwendung eines Makros nicht bekannt sein. Innerhalb der Makros können sowohl Data-Steps als auch Prozeduren ausgeführt werden. Außerdem erlauben Makros die bedingte Verarbeitung von SAS-Code (s. unten).

2.1 Makro-Variablen

Innerhalb eines Makros können eigene Makro-Variablen (sowohl global als auch lokal) mit %LET definiert werden. Diese Makro-Variablen werden dann in **Text-Format** in der Symboltabelle angelegt:

`%LET MakroVar1 = 10;` → Eintrag in der Symboltabelle: *MakroVar1* | *10*

Um Makro-Variablen aus der Symboltabelle auszulesen, wird mit einem & auf diese referenziert. Um dabei die in Text-Format vorliegende Makro-Variable als Text zu übergeben, wird die Makro-Variable in Anführungszeichen gesetzt:

```
Text = "&MakroVar1.";
```

Um eine Makro-Variable als Zahl zu übergeben, wird sie ohne Anführungszeichen einer numerischen Variablen gleichgesetzt:

Zahl = &MakroVar1.;
Es empfiehlt sich, Makro-Variablen generell mit einem Punkt als Delimiter zu beenden. Für manche Situationen ist das sogar unbedingt notwendig, aber es hat auch den Vorteil, dass die (meisten) Makro-Variablen dadurch im Code in farblich hervorgehoben werden und so besser erkennbar sind.

2.2 Kombination von Makro-Variablen

Makro-Variablen können kombiniert werden. Dazu werden z.B. zwei Makro-Variablen hintereinander geschrieben:
`Person = "&Name.&Nr.";` → aufgelöst z.B. *Müller10*
Bei der Verwendung von Makro-Variablen für Bibliotheksangaben innerhalb eines Data-Steps ist die Verwendung des Delimiters gefolgt von einem weiteren Punkt zwingend erforderlich, da SAS den ersten Punkt grundsätzlich als Makro-Variablen-Delimiter interpretiert:
`data &lib..&Datei.;` → aufgelöst z.B. *data work.test*

2.3 Indirekte Referenzen

Indirekte Referenzen werden verwendet, um Makro-Variablen über mehrere Ebenen aufzulösen. Es existiere beispielsweise eine Liste von Namen die in den Makro-Variablen Name1 bis Name9 abgelegt sind und nun soll mittels eines Zählers auf die Inhalte der einzelnen Namen-Variablen zugegriffen werden (der Einfachheit halber wird hier die Makro-Variable Nr mit %LET gesetzt):

```
%LET Name1 = Paul; %LET Name2 = Claudia; ...
%LET Nr = 2;
```

`Person = “&&Name&Nr.";` → aufgelöst im ersten Schritt: `Person = “&Name2";`
→ aufgelöst im zweiten Schritt: `Person = "Claudia";`
Bei indirekten Referenzen werden zunächst die vorhandenen & von links nach rechts aufgelöst, d.h. && wird zu & aufgelöst und *&Name* zu *Name*. Der Teil der indirekten Referenz dem dann kein & mehr voransteht, wird jeweils durch seinen Eintrag in der Symboltabelle ersetzt. Vorsicht: Bei indirekten Referenzen darf kein Delimiter zwischen die einzelnen Makro-Variablen gesetzt werden!

2.4 Übergabe von Parametern an das Makro

Es empfiehlt sich, mit Schlüsselwortparametern zu arbeiten, d.h. Im Aufruf des Makros die Übergabeparameter explizit zu benennen. Dadurch muss nicht auf die richtige Reihenfolge der Parameter geachtet werden und man verhindert die falsche Zuweisung von Parameter-Werten. Zudem ist es so möglich, vorbelegte Default-Parameter wegzulassen.

```
%MACRO Makroname(Tab=, Var=, Stat=);
...
```

```
%MEND Makroname;
%Makroname(Var=Name, Stat=mean, Tab=Daten)
```

2.5 Makro-Funktionen

Innerhalb von Makros können verschiedene Funktionen verwendet werden. Diese vordefinierten Funktionen beginnen jeweils mit einem %-Zeichen. Im Folgenden sind einige beispielhaft aufgeführt:

`%EVAL(...)`: Einbinden arithmetischer / logischer Operationen
`%SYSFUNC(...)`: Ausführen von SAS-Funktionen innerhalb eines Makros
`%UPCASE(...)`: Kleinbuchstaben → Großbuchstaben
`%SUBSTR(...)`: Extrahieren von Textteilen
usw.

2.6 Makro-Bedingungen

In Makros können Bedingungen für die Verarbeitung von Code / Daten benutzt werden, die durch Makro-Variable gesteuert werden. Einige Beispiele für Bedingungen sind in Tabelle 1 aufgeführt.

Tabelle 1: Beispiele für Bedingungen innerhalb eines Makros

`%IF Bedingung %THEN %DO;`
`  ...;`
`%END; %ELSE %DO;`
`  ...;`
`%END;`
	`%DO z = 1 %TO 10;`
	`  ...;`
	`%END;`
		`%DO %WHILE(Bedingung);`
		`  ...;`
		`%END;`

Der Unterschied zwischen Bedingungen ohne %-Zeichen und Bedingungen mit %-Zeichen innerhalb eines Data-Steps in einem Makro besteht darin, dass Bedingungen ohne %-Zeichen sich direkt auf den verwendeten Datensatz beziehen und von diesem gesteuert werden (also wie außerhalb eines Makros), während Bedingungen mit %-Zeichen von Makro-Variablen außerhalb des Data-Step gesteuert werden:

```
%MACRO test(a=);
    DATA test;
        IF a THEN b=c;  ← Steuerung durch die Werte der Variablen im Datensatz
        ELSE b=d;
        %IF &Var. %THEN %DO;  ← Steuerung durch die Werte der Makro-Variable
            b=c;
        %END; %ELSE %DO;
            b=d;
        %END;
    RUN;
%MEND test;
```

2.7 Programmverarbeitung

Bei der Verarbeitung von SAS-Code wird der Code im Input-Stack zunächst durch einen Scanner in einzelne Wörter zerlegt. Im Allgemeinen werden diese Wörter dann an den Compiler[1] weitergegeben. Dieser prüft die Syntax und führt den erhaltenen Code-Abschnitt aus, sobald eine sogenannte Schrittgrenze erreicht ist (z.B. ein "Run" nach einem Data-Step). Falls der Scanner allerdings auf ein %- oder &-Zeichen stößt, wird das darauf folgende Wort nicht an den Compiler übergeben, sondern an den sogenannten Makro-Prozessor (s. Abbildung 1). Dieser Prozessor kann zum einen Makro-Anweisungen oder -Befehle im Input-Stack durch den ausführlichen Code ersetzen und zum anderen Werte von Makro-Variablen in die Symboltabelle[2] schreiben oder daraus auslesen.

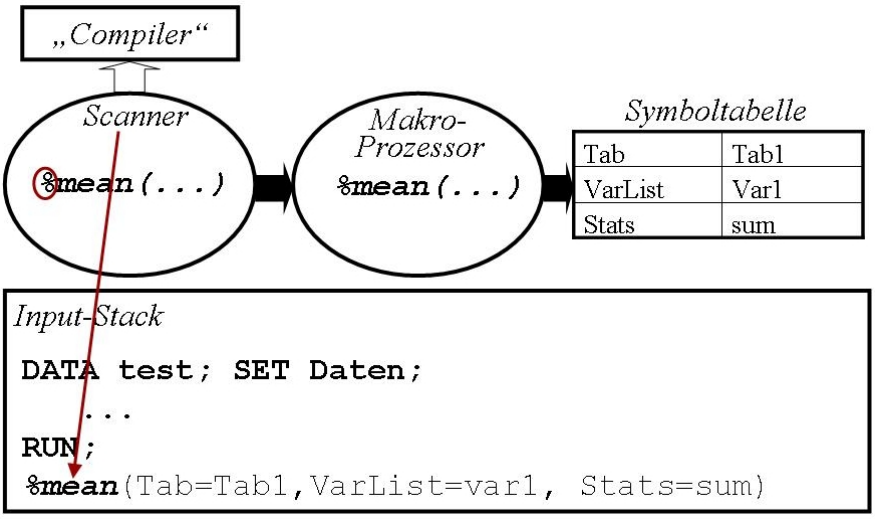

Abbildung 1: Erkennen von Makro-Anweisungen im Scanner
(vereinfachte Darstellung)

Nachdem die Makro-Anweisung im Input-Stack durch den Makro-Prozessor in konkreten SAS-Code aufgelöst wurde (s. Abbildung 2), wird dieser vom Scanner wieder in einzelne Wörter zerlegt und an den Compiler übergeben. Sobald nun ein &-Zeichen auftritt, ist das für den Scanner ein Hinweis, dass es sich dabei um eine Makro-Variable handelt, die vor der Weitergabe an den Compiler durch den entsprechenden Wert in der Symboltabelle ersetzt werden muss (s. Abbildung 2).

[1] SAS arbeitet im eigentlichen Sinne nicht mit einem Compiler sondern mit einem Interpreter. Ein Compiler überprüft und übersetzt den kompletten Code auf einmal in ausführbaren Maschinencode während der Interpreter den Code während der Laufzeit zeilenweise prüft, übersetzt und ausführt. Trotzdem wird in vielen SAS-Unterlagen das Wort Compiler benutzt.
[2] Der Einfachheit halber wird im Folgenden auf die Unterscheidung von globalen und lokalen Makro-Variablen und Symboltabellen verzichtet.

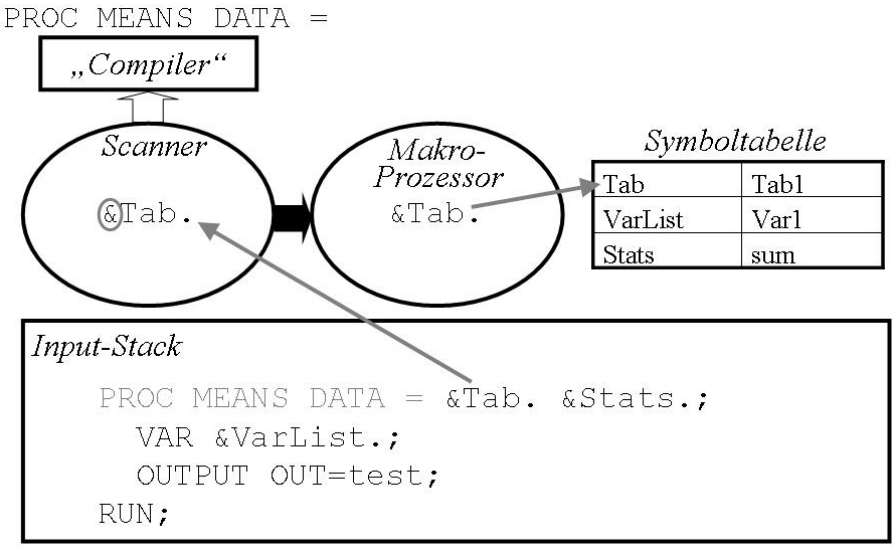

Abbildung 2: Erkennen von Makro-Variablen im Scanner (vereinfachte Darstellung)

Der Makro-Prozessor ersetzt dann die Makro-Variable im Input-Stack durch den entsprechenden Wert in der Symboltabelle. Daraufhin wird dieser konkrete Wert vom Scanner erfasst und nun an den Compiler weitergegeben. Es werden also erst sämtliche Makro-Variablen und Befehle in einem Code durch die entsprechenden Einträge in der Symboltabelle zu konkreten Werten bzw. den Operationen darauf aufgelöst bevor der Code im Compiler dann tatsächlich ausgeführt wird.

2.7.1 Auswirkungen der Programmverarbeitung auf die Programmierung

Aus dem skizzierten Ablauf der Programmverarbeitung ergeben sich weitreichende Konsequenzen für die Programmierung von Makros. Das folgende Beispiel soll dies illustrieren:

```
%MACRO test;
    DATA test;
       IF a THEN DO;
          %LET MarkoVar = Var1;
       END; ELSE DO;
          %LET MarkoVar = Var2;
       END;
    RUN;
%MEND test;
```

Der Code wird vor der Ausführung im Scanner zerlegt und die entsprechenden Makro-Befehle werden vor der eigentlichen Ausführung des Codes angewendet. D.h. dass hier durch den %LET-Befehl zunächst Var1 (als Text!) für die Makrovariable MakroVar in die Symboltabelle geschrieben wird. Dieser Eintrag wird dann durch den Eintrag Var2 überschrieben. Es erfolgt allerdings keine Zuweisung des Wertes der Variablen Var1 oder Var2 aus dem Data-Set. Nachdem die Makro-Befehle durch den Makroprozessor

aufgelöst und abgearbeitet sind, bleiben für die Übergabe an den Compiler lediglich folgende Code-Zeilen übrig:

```
DATA test;
     IF a THEN DO;
     END; ELSE DO;
     END;
RUN;
```

2.7.2 Abhilfe

Um die Zuweisung von Werten von Makro-Variablen während der Laufzeit im Compiler trotzdem zu ermöglichen, existieren verschiedene Möglichkeiten. So können z.B. Werte an Makro-Variablen aus einem Data-Set im Makro mit der Funktion CALL SYMPUT während der Laufzeit in die Symboltabelle geschrieben bzw. mit CALL SYMGET aus der Symboltabelle gelesen werden:

```
%MACRO test;
     DATA test;
          IF a THEN DO;
               CALL SYMPUT('MakroVar',Var1);
          END; ELSE DO;
               CALL SYMPUT('MakroVar',Var2);
          END;
     RUN;
%MEND test;
```

Außerhalb eines Data-Step besteht die Möglichkeit, mit PROC SQL Werte eines Data-Set an Makro-Variablen zu übergeben:

```
%MACRO test;
     ...
     PROC SQL NOPRINT;
          SELECT Var1 INTO: MakroVar
          FROM Tabelle;
     QUIT;
     ...
%MEND test;
```

3 Verallgemeinerung des einführenden Beispiels

Im Folgenden soll nun das einführende Beispiel mit PROC MEANS so verallgemeinert werden, dass damit für eine beliebige Anzahl von Variablen und Statistiken jeweils das Output-Out-Statement in seiner vollen Funktionalität genutzt werden kann.

3.1 Zerlegen der Eingabevektoren

Zunächst müssen sämtliche Eingabevektoren (also hier VarList und Stats) in ihre Einzelteile zerlegt und an Makro-Variablen übergeben werden. Erst dadurch wird es später möglich sein, das Output-Out-Statement flexibel zu gestalten indem auf die einzelnen Teile der Eingabevektoren zurückgegriffen wird:

```
%MACRO mean(Tab=,VarList=,Stats=);
    %LET VarNr = 0;
    %DO %WHILE(%SCAN(&VarList.,&VarNr.+1,%STR( ))^=);
        %LET VarNr = %EVAL(&VarNr.+1);
        %LET Var&VarNr.= %SCAN(&VarList.,&VarNr.,%STR( ));
    %END;
    ...
%MEND mean;
```

Die Do-Schleife scannt den Eingabevektor (dabei handelt es sich ja um nichts anderes als einen Text-String) auf einzelne Wörter die durch Leerzeichen getrennt sind. Anhand des mitlaufenden Zählers werden dann einzelne Makro-Variablen generiert, an die diese Wörter des Eingabevektors einzeln übergeben werden (Eintrag in der Symboltabelle).

Ein Aufruf des Makros könnte folgendermaßen aussehen:
%mean(Tab=Tab2,VarList=Variable1 Variable2, Stats=sum max)

In der Symboltabelle werden beim Aufruf zunächst die Makrovariablen (Eingabevektoren) Tab, VarList und Stats angelegt und die entsprechenden Zuweisungen aus der Übergabe eingetragen. Obige Do-Schleife zerlegt dann den Eingabevektor VarList aus der Symboltabelle und legt weitere Makro-Variablen an (in der folgenden Tabelle kursiv dargestellt).

Tabelle 2: Einträge in der Symboltabelle nach Aufruf des Makros und Auflösung der ersten Do-Schleife

Makro-Variable	Eintrag
Tab	Tab2
VarList	Variable1 Variable2
Stats	sum max
Var1	*Variable1*
Var2	*Variable2*
VarNr	*2*

Auf dieselbe Weise wird dann der zweite Eingabevektor (Stats) zerlegt und als weitere Makro-Variablen in die Symboltabelle eingetragen.

Tabelle 3: Vollständige Symboltabelle nach Aufruf des Makros und Auflösung der beiden Do-Schleifen

Makro-Variable	Eintrag
Tab	Tab2
VarList	Variable1 Variable2
Stats	sum max
Var1	*Variable1*
Var2	*Variable2*
VarNr	*2*
Stats1	*Sum*
Stats2	*Max*
StatsNr	*2*

3.2 Flexibles Erzeugen von Code beim Output-Out-Statement

Nun können die einzelnen Makro-Variablen verwendet werden, um durch den Makro-Prozessor vor der eigentlichen Ausführung des Codes im Compiler neuen Code entsprechend der Anzahl der Variablen und Statistiken beim Output-Out-Statement zu generieren:

```
%MACRO mean(Tab=,VarList=,Stats=);
   ...
   PROC MEANS DATA = &Tab. &Stats.;
      VAR &VarList.;
      OUTPUT OUT = results
         %DO i = 1 %TO &StatsNr.;
            &&Stats&i. =
               %DO ii = 1 %TO &VarNr.;
                  %LET out_var= %SYSFUNC(CATS(&&Stats&i.,_,&&Var&ii.));
                  &out_var.
               %END;
         %END;
         ;
   RUN;
   ...
%MEND mean;
```

Die erste Do-Schleife referenziert auf den Zähler der Statistiken in der Symboltabelle. Danach folgt eine indirekte Referenz (&&Stats&i.). Bei der Auflösung der Do-Schleife werden nun zwei Code-Abschnitte generiert bei denen jeweils die indirekte Referenz &&Stats&i. aufgelöst wird zu &Stats1. und &Stats2. Diese beiden Makro-Variablen referenzieren direkt auf die Einträge in der Symboltabelle, so dass diese durch sum und max ersetzt werden können:

```
%MACRO mean(Tab=,VarList=,Stats=);
   ...
   PROC MEANS DATA = &Tab. &Stats.;
```

```
        VAR &VarList.;
        OUTPUT OUT = results
            sum =
               %DO ii = 1 %TO &VarNr.;
                  %LET out_var= %SYSFUNC(CATS(sum,_,&&Var&ii.));
                  &out_var.
               %END;
            max =
               %DO ii = 1 %TO &VarNr.;
                  %LET out_var= %SYSFUNC(CATS(max,_,&&Var&ii.));
                  &out_var.
               %END;
            ;
    RUN;
    ...
%MEND mean;
```

Auf dieselbe Weise werden dann die Do-Schleifen für die Variablen aufgelöst, wobei hier eine Konkatenation zur Generierung einer neuen Output-Variablen verwendet wird. Letztendlich wird der komplette Code mit den Makro-Befehlen und -Variablen durch konkreten SAS-Code vor der Kompilierung ersetzt:

```
%MACRO mean(Tab=,VarList=,Stats=);
     ...
     PROC MEANS DATA = &Tab. &Stats.;
          VAR &VarList.;
          OUTPUT OUT = results
               sum = sum_Variable1 sum_Variable2
               max = max_Variable1 max_Variable2
               ;
     RUN;
     ...
%MEND mean;
```

Hier sieht man nun auch, wie wichtig es ist, das Semikolon am Ende der Output-Out-Anweisung nicht zu vergessen (oben mit den Makro-Befehlen erscheint zunächst ein doppeltes Semikolon).

4 Anwendungsbeispiel: Matching-Makro

Bei der KSFE 2011 wurde vom Autor eine optimierte Methode zum 1:N Matching von Fällen und Kontrollen in epidemiologischen Studien vorgestellt. Kernstück dieser Methode ist eine SQL-Prozedur die eine M_F:M_K-Verknüpfung der Fall- und Kontroll-Tabellen vornimmt. Anschließend werden für jeden Fall genau N Kontrollen nach einem bestimmten Algorithmus ausgewählt. Dabei werden bei gleichen möglichen Zuordnungen von mehreren Kontrollen zu verschiedenen Fällen unter anderem Zufallszahlen verwendet, um die Kontrollen zuzuordnen. Nach mehreren Versuchen mit jeweils neu

generierten Zufallszahlen wird das Ergebnis mit den meisten 1:N Treffern ausgewählt. Folgende Abbildung veranschaulicht noch einmal den Matching-Prozess[3]:

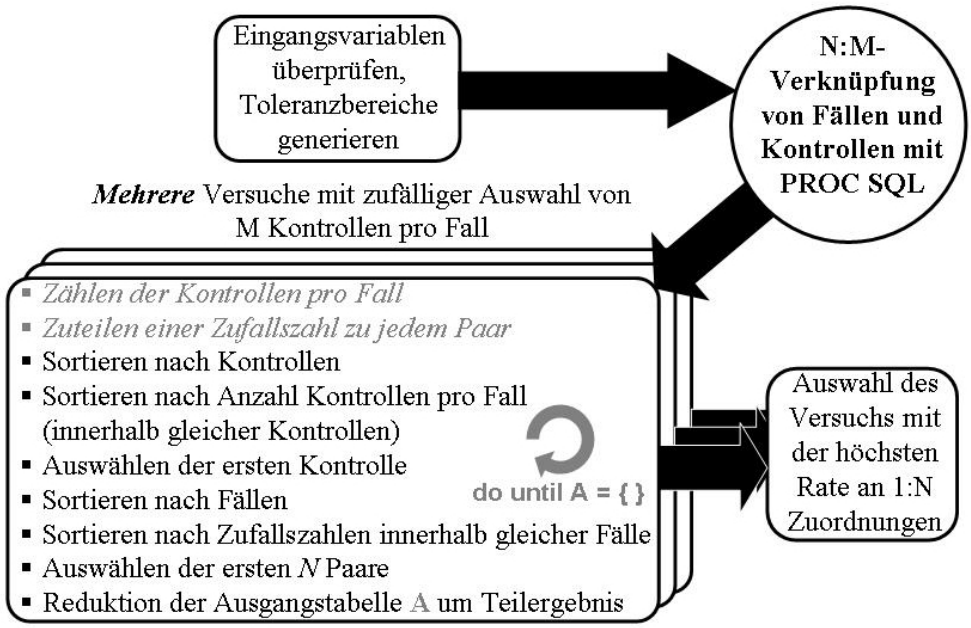

Abbildung 3: Ablauf-Schema des Matching

4.1 Erstellen eines situations-flexiblen Matching-Makros: Behandlung der Eingabeparameter und Erzeugung der Toleranzgrenzen

Zunächst werden optionale Eingabeparameter auf deren Existenz geprüft und ggf. auf Default-Werte gesetzt[4]:

```
%MACRO match(TabCases=, TabControls=, TabOut=, caseID=, controlID=,
               N=, attempts=,VarList=, RangeVarList=, RangeNrList=);
     %IF &N.= %THEN %LET N=1;
     %IF &attempts.= %THEN %LET attempts=1;
     ...
%MEND match;
```

Da die Makro-Variablen in Text-Format in der Symboltabelle gespeichert sind, genügt die Überprüfung wie im Code gezeigt[5]. Danach erfolgt eine Zerlegung der optionalen und flexiblen Eingabevektoren VarList, RangeVarList und RangeNrList. VarList beinhaltet dabei die Variablen die beim Matching fixiert bleiben sollen (z.B. Geschlecht),

[3] Details zur Matching-Prozedur können nachgelesen werden in:
Deckert A.: **1:N Matching von Fällen und Kontrollen: Propensity Score vs. PROC SQL.** KSFE 2011 - Proceedings der 15. Konferenz der SAS-Anwender in Forschung und Entwicklung. shaker Verlag; 105-119.
[4] Default-Werte können auch direkt in der Schnittstelle des Makros-Kopfes angegeben werden.
[5] Alternativ können Anführungszeichen verwendet werden: %IF "&N."="" %THEN %LET N=1;
Generell wird Code in doppelten Anführungszeichen vom Scanner in einzelne Worte zerlegt, Code in einfachen Anführungszeichen wird dagegen nicht aufgelöst!

RangeVarList dagegen gibt Variablen an, für die ein Toleranzbereich (in RangeNrList) definiert wird (z.B. Geburtsjahr ± 2 Jahre).

```
%MACRO match(TabCases=, TabControls=, TabOut=, caseID=, controlID=,
                N=, attempts=, VarList=, RangeVarList=,
RangeNrList=);
   ...
   %LET VarNr = 0;
   %DO %WHILE(%SCAN(&VarList.,&VarNr.+1,%STR( ))^=);
      %LET VarNr = %EVAL (&VarNr.+1);
      %LET Var&VarNr.= %SCAN(&VarList.,&VarNr.,%STR( ));
   %END;

   %LET RangeVarNr = 0;
   %DO %WHILE(%SCAN(&RangeVarList.,&RangeVarNr.+1,%STR( ))^=);
      %LET RangeVarNr = %EVAL (&RangeVarNr.+1);
      %LET RangeVarNr&RangeVarNr.=%SCAN(&RangeVarList.,
                                          &RangeVarNr.,%STR( ));
      %LET RangeVarNr&RangeVarNr._low=%SYSFUNC(CATS
                                  (&&RangeVarNr&RangeVarNr.,_low));
      %LET RangeVarNr&RangeVarNr._high=%SYSFUNC(CATS
                               (&&RangeVarNr&RangeVarNr.,_high));
   %END;

   %LET RangeNr = 0;
   %DO %WHILE(%SCAN(&RangeNrList.,&RangeNr.+1,%STR( ))^=);
      %LET RangeNr = %EVAL (&RangeNr.+1);
      %LET Range&RangeNr. = %SCAN(&RangeNrList.,&RangeNr.,%STR( ));
      %LET _Range&RangeNr. = INPUT(&&Range&RangeNr.,best12.);
   %END;
   ...
%MEND match;
```

Dabei werden für jede im Eingabevektor RangeVarList gefundene Variable (z.B. Geburtsjahr) mit Hilfe von indirekten Referenzen zwei neue Makro-Variablen erzeugt. Mit diesen wird dann wieder über indirekte Referenzen die untere und obere Grenze für jede Toleranz-Variablen im Datensatz der Kontrollen berechnet:

```
DATA _controls; SET &TabControls.;
     %DO I = 1 %TO %EVAL(&RangeVarNr.);
          &&RangeVarNr&I._low = &&RangeVarNr&I. - &&_Range&I.;
          &&RangeVarNr&I._high = &&RangeVarNr&I. + &&_Range&I.;
     %END;
RUN;
```

4.2 Laufzeit-Optimierung

Wie oben schon erwähnt werden innerhalb des Makros mehrere Versuche mit verschiedenen Zufallszahlen für die Zuordnung der Kontrollen zu den Fällen durchgeführt. Man könnte nun für jeden dieser Versuche eine Verknüpfung von Fällen und Kontrollen mit

PROC SQL und den anschließenden Sortier- und Reduzier-Algorithmus separat durchführen. Dabei können aber je nach Beschaffenheit und Größe der Datensätze größere Rechenzeiten anfallen. Eine spezielle Stärke von SAS besteht darin, wenige aber dafür sehr große Datensätze statt vieler kleiner Datensätze zu bearbeiten. Daher empfiehlt es sich hier, VOR der Verknüpfung von Fällen und Kontrollen den Datensatz der Fälle entsprechend der Anzahl der Versuche mit verschiedenen Zufallszahlen zu vervielfältigen und dann die ganzen nachfolgenden Prozeduren jeweils über diesen großen Datensatz laufen zu lassen, entsprechend gruppiert z.B. durch BY-Statements[6]:

```
DATA _cases; SET &TabCases.;
     %DO _attempts=1 %TO %EVAL(&attempts.);
          _random=UNIFORM(0);
          _turn=&_attempts.;
          OUTPUT;
     %END;
RUN;
```

4.3 Flexibles Verknüpfen von Fällen und Kontrollen mit PROC SQL

Die Verknüpfung von Fällen und Kontrollen erfolgt mit PROC SQL. Diese Prozedur soll nun so gestaltet werden, dass der Makro-Prozessor vor der eigentlichen Verarbeitung des Codes im Compiler die notwendigen Code-Zeilen im SELECT- und WHERE-Statement selbst generiert. Der Einfachheit halber ist dabei nur der Code für die fixen Variablen dargestellt, für die Toleranz-Variablen gestaltet es sich aber ähnlich.

```
PROC SQL;
   CREATE TABLE _reduction AS SELECT
      %IF (&VarNr. ne 0) %THEN %DO;
         %DO I=1 %TO %EVAL(&VarNr.);
            A.&&VarNr&I. AS case_&&VarNr&I.,
            B.&&VarNr&I. AS control_&&VarNr&I.,
         %END;
      %END;
      ...
      A.&CaseID. AS case_ID, B.&ControlID. AS control_ID,
      A._random AS _random, A._turn AS _turn,
      A._dummy = B._dummy
      FROM _cases A, _controls B
      WHERE (
         %IF (&VarNr. ne 0) %THEN %DO;
            %DO I=1 %TO %EVAL(&VarNr.);
               A.&&VarNr&I. = B.&&VarNr&I. AND
            %END;
         %END;
         ...
```

[6] Für ausführlichere Informationen zur Laufzeit-Optimierung durch Erstellung großer Datensätze siehe:
Englert S.: **Empirische Poweranalyse.** KSFE 2011 - Proceedings der 15. Konferenz der SAS-Anwender in Forschung und Entwicklung. shaker Verlag; 147-154.

```
            A._dummy = B._dummy)
        ORDER BY _turn, case_ID;
QUIT;
```

Die IF-Bedingung beim SELECT- und WHERE-Statement prüft zunächst, ob eine fixe Variable überhaupt vorhanden ist. Danach wird jeweils über die Do-Schleife und indirekte Referenzen Code für jede fixe Variable vor der Laufzeit generiert. Beim WHERE-Statement werden die einzelnen Bedingungen über AND verknüpft. Um dabei das letzte AND abzufangen wurde hier einfach in beiden Datensätzen eine dummy-Variable generiert und auf 1 gesetzt[7].

4.4 Auswahl der Kontrollen mit Laufzeit-Optimierung

Nach der Verknüpfung von Fällen und Kontrollen erfolgt die Auswahl und eigentliche Zuordnung der Kontrollen zu den Fällen (s. Schema Abbildung 3). Der Code dafür ist hier nur angedeutet, es soll aber noch einmal gezeigt werden, wie dabei ebenfalls die Optimierung der Laufzeit durch die vorherige Vervielfachung des Datensatzes anstatt einer weiteren DO-Schleife für sämtliche Versuche erreicht wird. Alle Prozeduren und Data-Steps werden dafür mit der ersten Variablen _turn in den BY-Statements entsprechend den verschiedenen Versuchen gruppiert. So können alle Versuche auf einmal abgehandelt werden. Ebenfalls in diesem Code zu sehen ist am Ende die Übergabe der Zählung der verbleibenden Einträge in der Augsangstabelle an eine Makro-Variable, die dann die übergeordnete Do-Schleife steuert.

```
%DO %UNTIL (&_count. = 0);
    PROC SORT DATA=_reduction;
        BY _turn control_ID num_controls _random;
    RUN;
    DATA _set1; SET _reduction; BY _turn control_ID;
        IF first.control_ID;
    RUN;
    ...
    DATA _reduction;
        MERGE _reduction _set2 (IN = b KEEP = _turn control_ID);
        BY _turn control_ID;
        IF NOT b;
    RUN;
    ...
    PROC SQL NOPRINT; SELECT COUNT(*) INTO :_count
        FROM _reduction; QUIT;
%END;
```

[7] Hier gibt es sicher noch weitere und auch elegantere Lösungen wie z.B. die Programmierung der DO-Schleife bis &VarNr.-1 und einer extra erzeugten Zeile für die letzte Variable. Dabei könnten dann allerdings evtl. weitere Probleme auftreten, falls z.B. nur eine einzige fixe Variable vorhanden ist.

4.5 Flexible Ergebnisausgabe

Das mit der bedingten Verarbeitung von Code flexibel gestaltete Matching-Makro kann nun die unterschiedlichsten Situationen handhaben. Beispielsweise kann ein Aufruf eines 1:1 Matching mit 3 festen Variablen aber ohne Toleranz-Variable und ohne Benutzung der Zufallsversuche erfolgen:

```
%match(TabCases=cases, TabControls=controls, TabOut=test,
       caseID=ParticipantID, controlID=control_ID,
       VarList=sex y_birth y_move)
```

Die Ergebnisausgabe sieht dann folgendermaßen aus:

```
---------------------------RESULTS-------------------------------
For 17 case(s) no control was found.
For 483 case(s) a complete 1:1 matching was performed.
```

Aber es kann auch ein 1:3-Matching mit mehreren Toleranz-Variablen und mehreren Zufallsversuchen durchgeführt werden:

```
%match(TabCases=cases, TabControls=controls, TabOut=test,
       caseID=ParticipantID, controlID=control_ID,
       RangeVarList=y_birth y_move, RangeNrList=3 4, VarList=sex,
       N=3, attempts=20);
```

Hier erscheint im Log-Fenster folgende Ergebnisausgabe:

```
---------------------------RESULTS-------------------------------
Attempt Nr. 7 was most successful.
Altogether 285 cases were matched with at least one control.
For 15 case(s) no control was found.
For 39 case(s) a reduced set of 1 control(s) was found.
For 22 case(s) a reduced set of 2 control(s) was found.
For 224 case(s) a complete 1:3 matching was performed.
```

Literatur

[1] Deckert A.: 1:N Matching von Fällen und Kontrollen: Propensity Score vs. PROC SQL. KSFE 2011 - Proceedings der 15. Konferenz der SAS-Anwender in Forschung und Entwicklung. shaker Verlag; 105-119.

[2] Englert S.: Empirische Poweranalyse. KSFE 2011 - Proceedings der 15. Konferenz der SAS-Anwender in Forschung und Entwicklung. shaker Verlag; 147-154.