Paper CT03

Resolving algorithmic SMQs – using AI chatbot and other techniques

Corentin Le Scanff, Elderbrook Solutions, Bietigheim-Bissingen, Germany
Steve Prust, Elderbrook Solutions, Bietigheim-Bissingen, Germany

ABSTRACT

This paper focuses on how to process algorithmic Standardized MedDRA Queries (SMQs). It examines how algorithms can generally be resolved, the issues specific to SMQ algorithms, programming techniques and efficiency. Using the example of SMQs, methods and platforms for resolving algorithms are examined as well as AI chatbots capacity to understand the problem. SMQ algorithms need events to happen together and determining "together" is potentially complex. How do we consider events as overlapping? Do we consider only start dates or start to end intervals? Are adjacent events allowable, if so, what is the allowed gap? We present a solution that reverses the problem. Instead of looking at adverse events in relation to each other, we examine patients to see if the SMQ is satisfied. We show other techniques including using hash tables and custom functions.

INTRODUCTION

Standardised MedDRA Queries (SMQs)

Standardised MedDRA Queries (SMQs) are a feature of the Medical Dictionary for Regulatory Activities (MedDRA), which is widely used in the pharmaceutical industry for coding adverse events. SMQs are predefined sets of terms from MedDRA, grouped together to help identify potential safety issues or adverse event patterns in clinical and post-marketing data. By using standardized groupings of MedDRA terms (SOC, HLGT, HLT, PT, LLT), SMQs can simplify the analysis of complex safety data. Instead of manually reviewing individual adverse event terms, SMQs can be programmed to quickly identify relevant safety concerns or patterns. For more details about scope, please refer to "Introductory Guide for Standardised MedDRA Queries (SMQs) Version 27.0"

One example of SMQ is Drug reaction with eosinophilia and systemic symptoms syndrome. This is the one that will serve as an example at the beginning of this paper. Drug reaction with eosinophilia and systemic symptoms (DRESS) is a severe, potentially life-threatening, delayed-onset drug reaction characterized by skin eruption, fever, hematologic abnormalities (eosinophilia, atypical lymphocytes), lymphadenopathy, and internal organ involvement.

SMQs Algorithms

But what is of highest interest for programmers is that this DRESS SMQ is an algorithmic SMQ. It means that identifying a term alone is not enough, it is a combination of terms that can identify an event.

For DRESS SMQ, terms are classified in 5 categories, as described below:

- *Category A – Terms directly referring to DRESS syndrome*
- *Category B – Terms related to organ damage, including skin involvement, viral reactivation, general hypersensitivity*
- *Category C – Terms related to fever*
- *Category D – Terms related to lymphadenopathy*
- *Category E – Terms related to hematologic abnormalities commonly seen in DRESS case*

A patient is considered as having a DRESS if he satisfies the following algorithm:

- *A or (B and C and D) or (B and C and E) or (B and D and E)*

It means either the patient has an obvious DRESS event with a category A term, or the patient has a combination of B, C, D and E terms which also characterize a DRESS event.

Togetherness of events

Another key piece of information about this DRESS SMQ is about timing. MedDRA guideline states the following: *cases are excluded that: Did not report a temporal relationship between suspect drug and reaction; note: this may include cases in which the multiple signs & symptoms associated with DRESS did not occur within one month of each other (e.g. skin eruption followed by lymphadenopathy and fever 6 months later).*

This information about timing or togetherness adds complexity for programmers, but also for the whole team because the definition provided in the guideline is not perfect. It first uses the conditional "may include" and it also doesn't precisely how to handle the "one month of each other" period. This leaves the following questions open for interpretation:

- Do we consider time between onset of AEs, or do we consider time between AE whole duration? In other words, is it acceptable to consider a positive temporal relationship of less than 30 days if one event ends 10 days before another event start, despite there are 60 days between onset of events?
- Do we consider that events B, C and D must all occur in a period of 30 days, or is it acceptable to have 30 days between B and C and to have 30 days between C and D?

Unfortunately, there is no perfect decision here. Such a decision must be taken in close collaboration with members of the team who have medical backgrounds. And such a decision might depend on the SMQ itself or on the project of interest. Even for a given SMQ, having a given algorithm, the decision on how to implement it could be different depending on the therapeutic area or on the drug's own mechanisms.

For some SMQs like DRESS, programmers will need to consider the timing of events, additionally to the algorithm. Let's see if such a problem can be understood and solved by ChatGPT. To be complete about SMQs, concepts of scope (narrow and broad) and weight need to be considered but will not be detailed in this paper.

CHECK THE CAPACITY OF AI CHATBOTS TO UNDERSTAND AND SOLVE THE PROBLEM

See what ChatGPT initially says

In order to expect a good answer, you'd rather ask a good question. But before doing so, it is interesting to assess where you start from. The first thing we did was to dump a text file shown below on ChatGPT with a simple question:

Can you guess anything about this set of data?

STUDYID	USUBJID	COMMENT	AEPTCD	AEDECOD	ASTDT	AENDT
1234_5678	1234_5678-11111111	A term	10073508	Drug reaction with eosinophilia and systemic symptoms	01FEB2024	11FEB2024
1234_5678	1234_5678-22222222	B C D exact same day	10037844	Rash	01MAR2024	11MAR2024
1234_5678	1234_5678-22222222	B C D exact same day	10037660	Pyrexia	01MAR2024	11MAR2024
1234_5678	1234_5678-22222222	B C D exact same day	10014950	Eosinophilia	01MAR2024	11MAR2024
1234_5678	1234_5678-22222222	B C D exact same day	10014950	Eosinophilia	01SEP2024	11SEP2024
1234_5678	1234_5678-33333333	B C D adjacent	10037844	Rash	01APR2024	11APR2024
1234_5678	1234_5678-33333333	B C D adjacent	10037660	Pyrexia	20APR2024	30APR2024
1234_5678	1234_5678-33333333	B C D adjacent	10014950	Eosinophilia	30APR2024	10MAY2024
1234_5678	1234_5678-44444444	B C D E not adjacent	10037844	Rash	01MAY2024	11MAY2024
1234_5678	1234_5678-44444444	B C D E not adjacent	10037660	Pyrexia	05MAY2024	15MAY2024
1234_5678	1234_5678-44444444	B C D E not adjacent	10014950	Eosinophilia	15JUN2024	25JUN2024
1234_5678	1234_5678-44444444	B C D E not adjacent	10025197	Lymphadenopathy	20JUN2024	30JUN2024

ChatGPT 4o *Based on the data provided in the file, it seems to be related to adverse event tracking, possibly from a clinical study. Here's what I can infer:*

1. *Study and Subject Identification: The STUDYID column identifies the study (e.g., 1234_5678), and USUBJID provides a unique identifier for each subject. Multiple entries with the same USUBJID indicate that the same subject experienced more than one event.*
2. *Adverse Event Codes and Descriptions: AEPTCD represents the adverse event code (e.g., 10073508), while AEDECOD gives the corresponding medical term (e.g., "Drug reaction with eosinophilia and systemic symptoms" or "Rash").*
3. *Date Ranges for Events: ASTDT and AENDT columns specify the start and end dates for each adverse event. Some subjects have overlapping events, indicating multiple symptoms or conditions reported within the same time period.*

The first answer from ChatGPT is already quite interesting. It tells us that it understands the data. Now let's see if it understands the problem itself and ask the next question.

Thanks, could you please now identify patients who experienced a "Drug reaction with eosinophilia and systemic symptoms syndrome SMQ"?

ChatGPT 4o *I have identified the patient who experienced "Drug reaction with eosinophilia and systemic symptoms syndrome" (DRESS syndrome). You can now view the details in the table provided. Let me know if you need further analysis or any other information!*

ChatGPT answers that only patient 1234_5678-11111111, the one having a category A term, has a DRESS event. This is not good enough. Apparently ChatGPT didn't consider any MedDRA guideline in his answer. The second answer is disappointing. However, this can be only a problem of specifying the question in proper terms.

Thanks, unfortunately this is not accurate. You need to consider that DRESS SMQ needs to be examined as a combination of events satisfying an algorithm with categories identified as A, B, C, D and E as described in MedDRA guidelines. Would you mind checking you know about it and reassess, if possible, give answer in text and not in a data table?

ChatGPT 4o The answer provided is very long but still doesn't manage to get the algorithm information. It assumes wrongly that A, B, C, D and E are equivalent, and that the algorithm is (A and B and C and D and E).
Now let's try to give better information to ChatGPT.

[give "A or (B and C and D) or (B and C and E) or (B and D and E)" as well as the definition of each category as per guideline]

ChatGPT 4o
Summary of Findings
- *Patient 1 meets the criteria based on having a Category A event alone.*
- *Patients 2 and 3 meet the criteria through Criterion 3 (B and C and E).*
- *Patient 4 meets multiple criteria (2, 3, and 4), reinforcing the assessment.*

Additional Notes
- *While the algorithm doesn't'specify exact time frames, it's'generally important that the events occur within a clinically relevant period.*
- *For Patient 2, all events occurred between 01MAR2024 and 11MAR2024, indicating temporal proximity.*
- *Patient 3's'events are adjacent in time during April and May 2024, which may be acceptable depending on clinical judgment.*
- *Patient 4 experienced events over a more extended period (May to June 2024). The temporal gaps might require clinical evaluation to determine relevance.*

This answer is extremely interesting as ChatGPT clearly identifies the fact that depending on time frames, events can be considered as relevant or not.

Do you think you could do the assessment again, keeping the same logic, but you should only identify a DRESS positively if all contributing events occurs in a time window of 30 days. You should consider event start and stop dates, an event which starts 20 days after the stop of another event is acceptable.

ChatGPT 4o 2nd Eosinophilia for patient 22222222 is considered outside of the 30 days window and discards

After a few more steps, we were finally able to build a question precise enough for ChatGPT to succeed, here is the outcome:

Can you identify which patients in my dataset qualify for DRESS syndrome using the following criteria:
Apply the DRESS SMQ algorithm:
Criterion 1: The patient has a term from Category A (directly referring to DRESS), or
Criterion 2: The patient has terms from Category B (organ damage) AND Category C (fever) AND Category D (lymphadenopathy), or
Criterion 3: The patient has terms from Category B (organ damage) AND Category C (fever) AND Category E (hematologic abnormalities), or
Criterion 4: The patient has terms from Category B (organ damage) AND Category D (lymphadenopathy) AND Category E (hematologic abnormalities).
Apply the 30-day time window constraint: All events contributing to the DRESS diagnosis must occur within a continuous 30-day window, meaning the start of an event must occur within 30 days of the stop of the preceding event.
Can you let me know which patients qualify for DRESS based on these criteria?
I would like a quick summary as an answer, including patients having DRESS but also patients not having DRESS.

ChatGPT 4o *Based on the DRESS SMQ algorithm and the 30-day time window constraint, here's'a quick summary of which patients qualify for DRESS syndrome and which do not:*

Patients Qualifying for DRESS:
- *Patient 1234_5678-11111111: Met Criterion 1, which means this patient had a direct term referring to DRESS (Drug reaction with eosinophilia and systemic symptoms).*
- *Patient 1234_5678-22222222: Met Criterion 3, indicating this patient had terms related to organ damage (rash), fever, and hematologic abnormalities (eosinophilia), all occurring within a 30-day window.*
- *Patient 1234_5678-33333333: Also met Criterion 3, as the patient had terms related to organ damage (rash), fever, and hematologic abnormalities (eosinophilia) within a 30-day window.*

Patients Not Qualifying for DRESS:

- *Patient 1234_5678-44444444: Did not meet any DRESS criteria because the required terms (organ damage, fever, lymphadenopathy, and hematologic abnormalities) did not occur within a continuous 30-day window.*

The answer is correct. It required a lot of steps to come to it, but in the identification of DRESS cases by ChatGPT was correct. Let's now see if ChatGPT can create some SAS code to do the same.

SAS code created by IA

The only step left is to ask ChatGPT to generate some SAS code based on its understanding of the problem.
The full code obtained is available in appendix 1, at tne end of this paper. It consists of:

- A data step with an infile statement to read test data
- A categorization of events into A B C D and E categories
- A proc sort
- A data step combining:
    - Some retain statements used in combination with first statement to create boleans variables.
    - Some variables that contain the first event start inside each category.
    - Some statement comparing min and max dates to see if they belong to a 30 days window.
    - An output statement associated to a last statement.
- A proc print to display the result

The code is also well described by proper commenting.

The first thing to notice is that the code is working fine on the test data provided.
However, it is also important to notice that this code fails to deal with new data constellations that are not present in the test data. It is for example failing as soon as the first even contributing to the part of the algorithm is not satisfying the algorithm while an event occurring later and in the good combination with other events is not selected.

To conclude about AI chatbot, we have demonstrated that:

- AI chatbots can understand the problem providing that you ask the question very precisely.
- AI chat bot can generate working SAS code based on test data.
- AI chat bot can't yet generate code that anticipate every data constellation.

We could consider ChatGPT as a beginner SAS programmer, you need to explain the problem in many details and the code is working but not really usable.

A SIMPLE SAS TECHNIQUE

SAS code

As we've seen above, solving a problem first starts with posing the right question. As a programmer, everyone has a different way to apprehend a problem. For me, the easiest way to apprehend the problem is to "reverse" it. The first idea that came to my mind was to look for a given event satisfying the algorithm and then to check if other necessary events were also in the correct time window. However, another approach is to look at every day a patient is on treatment and make is a treatment interval which correspond to the allowed time window.
Rather examine individually every treatment interval to determine if the algorithm is satisfied or not.

```
*** Take AE data as well as treatment period ***;
proc sort data=__adae0
    out=__adae00 (keep=studyid usubjid trtsdt trtedt) nodupkey;
    by studyid usubjid trtsdt trtedt;
run;
*** Clean ***;
data __adae00;
    set __adae00;
    if missing(trtsdt) or missing(trtedt) then delete;
run;
*** Compute all windows ***;
data __adae01;
    set __adae00;
    format date_low date_high date9.;
    do date_low = (trtsdt-&adjacent) to (trtedt);
        date_high = date_low + &adjacent;
        output;
    end;
run;
```

```
*** Check if any AE is inside a specific window ***;
proc sql noprint;
    create table adae_algo0 as
        select a.*, var_a, var_b, var_c, var_d, var_e, astdt, aedecod
        from __adae01 as a join __adae1 as b
        on (date_low <= b.ASTDT <= date_high or date_low <= b.AENDT <= date_high)
          and a.usubjid = b.usubjid;

    create table adae_algo1 as select studyid, usubjid, date_low, date_high,
         max(var_a) as max_a,
         max(var_b) as max_b,
         max(var_c) as max_c,
         max(var_d) as max_d,
         max(var_e) as max_e
         from adae_algo0
         group by studyid, usubjid, date_low, date_high;
quit;
```

```
*** Identify dates for which UDAEC is identified ***;
data adae_algo2;
    set adae_algo1;
    if max_b = 1 and max_c = 1 and max_d = 1 then BCD = 1;
    if max_b = 1 and max_c = 1 and max_e = 1 then BCE = 1;
    if max_b = 1 and max_d = 1 and max_e = 1 then BDE = 1;
    a = max_a;
      if a or bcd or bce or bde then smq = 1;
      if smq = 1 then output;
run;
```

```
data to_be_added;
    set adae_a
         adae_bcd_:
         adae_bce_:
         adae_bde_:;
    drop var:;
    &prefix.NAM = MUDAEC;
    &prefix.CD  = MUDAECCD;
    &prefix.SC  = "BROAD";
run;

proc sort data=to_be_added nodupkey;
    by _ALL_;
run;
```

The rest of the code is available in appendix 2 at the end of this paper.

Pros and cons

This way of doing also allows to know which adverse events contribute to your SQM, which is usually the level of information you want to keep in your ADAE. It is not enough to only know that a patient fulfils the criteria because you usually want to examine the case in more details. It is also easy to debug, which means easy to ensure your implementation of the algorithm is working.

In the context of writing this algorithm, we only need to derive one algorithmic SMQ and the number of patients is reasonable. Number of patients, duration of on-treatment period, number of studies, number of algorithmic SQMs are a few points that need to be considered when you decide which way you want to go. Using a combination of loops and cartesian joins followed by a proc sort nodupkey is not the fanciest technique, but it does the job correctly without performance issues.

SOLVING THE ALGORITHM GENERALLY

How to solve an algorithm generally; a general approach takes us towards AI-like techniques and in this context we develop some code. From the resultant exercise we evaluate how this code and the underlying techniques might - or might not - correspond to AI needs.

To solve the algorithms generally an initial approach was to use a SAS macro that would take using each distinct algorithm definition and with an Adverse Event dataset determine for each subject whether or not they had an event satisfying the algorithm. From this step we then planned to develop the work to deal with the issue of contemporaneous events.

Initial approach

The macro to examine the Adverse Event data is shown in Appendix 3. Briefly it consists of these elements.

- A loop through each of the different algorithms.
- For each iteration of the loop a data step to read the Adverse Event dataset and produce a result dataset for that algorithm with one observation per patient.
- The data step uses hash table functionality to, for each algorithmic element (A, B, C and so on), store the applicable MedDRA codes for that algorithmic element (this is performed once at the start of the data step)
- For each AE observation it is evaluated to see if it satisfies each separate algorithmic element. If true, then the result of this is stored as a Boolean value against that particular element.
- On the last observation for each patient the separate Boolean results for each element are substituted into the algorithm. For example, "A or (B and C)" might become "1 or (0 and 0)" or "0 or (0 and 1)" or "0 or (1 and 1)" (for example).
- The function RESOLVE is then used to determine the converted algorithm as a Boolean result. This is a nice "trick" as it invokes the macro processor to evaluate a text string - the macro processor is a convenient and quick method of doing this. In the code below the variable SMQALGO_EVAL might contain the text "%EVAL(' 0 and (0 and 1) ')" or "%EVAL(' 0 and (1 and 1) ')". The expression INPUT(RESOLVE(smqalgo_eval),best.) will then return the Boolean results of 0 and 1 respectively which can then be used in an IF statement or whatever. See code section below.
- At the end of the data step there is a result dataset stating if each patient did or did not fulfill the algorithmic SMQ.
- Note that extra coding exists to cope with the "Sum(Category Term Weight)" in one of the algorithms.

Although hash tables are loaded into memory and can have an impact on performance the use of hash tables here is not considered a performance issue due to the low amount of data required.

```
if last.usubjid then do;
  smqalgo_eval = '%eval( ' || trim(smqalgor) || ' )';
  if index(smqalgor,'Sum') then do;
    sumwgt = 0;
    do i = 1 to 9;
      if __termwgt[i] > 0 and __termres[i] > 0 then sumwgt = sumwgt + __termwgt[i];
    end;
    smqalgo_eval = tranwrd(smqalgo_eval,'Sum(Category Term Weight)',compress(put(sumwgt,best.)));
  end;
  do i = 1 to &ncats;
    termcat = substr("&&__cats&i",i,1);
    if index(smqalgo_eval,trim(termcat)) then smqalgo_eval = tranwrd(smqalgo_eval,trim(termcat),compress(
    put(__termres[i],best.)) );
  end;
  smqresult = input(resolve(smqalgo_eval),best.);
  length cats $20;
  cats = "&&__cats&i";
  output;
end;
```

The rest of the code is available in Appendix 3.

Developing the macro further - Contemporaneous event requirement

The initial example assumed that in the case of a term such as "B or C" that it did not matter when B and C might occur in relation to each other. Clearly, if there were years between an event of category B and another of category C then there could be a scientific decision to make as to whether the algorithm applies. In practice the authors of this paper have been instructed to use a 30 day maximum gap between the periods of any applicable events in evaluating the algorithm.

The consequence for the initial macro is that now at each event that qualifies for anything less than a simple term we need different processing. Note: by simple term we mean that in the case of "A and (B or C)" that "A" is "simple" because satisfying events do not need to be contemporaneous with any other event (whereas "(B and C)" is non-simple).

In an earlier example an SQL query was used for the contemporaneous event qualification. In trying to develop a general solution a different approach is taken: for any event that satisfies the first part of a non-simple term (for example we will use "B or C" to illustrate below) then are there any other events within that term for this patient and within a time period defined by the original event start date less 30 days to end date plus 30 days?

This sounds difficult but a technique involving user written functions and the macro facility can make this happen. It relies on a solution given in a paper to the SAS Global Forum in 2012 by the author Mike Rhoads:

> "*what do you do if you need a macro that can be embedded within a SAS statement, but the underlying task requires the execution of one or more DATA or PROC steps?"*

User written function to get an answer from an sql query

Here is a simple example of a user written function to get an "answer" from an SQL query. It consists of

1. A user written macro that will firstly call an SQL query via a macro (see 2.) and then return a numeric value using the macro variable count.
2. A macro that calls an SQL and places the count of a named variable into a macro variable called "count"

```
proc fcmp outlib=sasuser.mysubs.utility;
 function GetSQLans(query $, var $) ;
 length query $ 32000  var $ 32 count $ 32;
 query_arg = dequote(query);
 var_arg   = dequote(var);
 rc = run_macro('GetSQLans_Inner', query_arg, var_arg, count);
 if rc = 0 then return(input(count,best.));
 else return(-1);
 endsub;
run;quit;

%MACRO GetSQLans_Inner;
 %local query ;
 %let query = %superq(query_arg);
 %let query = %sysfunc(dequote(&query));

 %local var ;
 %let var = %superq(var_arg);
 %let var = %sysfunc(dequote(&var));

 %let count = 0;
  proc sql noprint;
    select count(&var) into : count from &query;
  quit;

%MEND GetSQLans_Inner;
```

The next step demonstrates how the function can be used. On the line below "count = GetSQLans (query);" will execute a query that has been defined in the line above - the code is able to say for each observation "how many other observations are like (in terms of sex and age) this observation". We are now reading from the whole dataset at the same time as processing individual observations.

```
options cmplib=sasuser.mysubs;
data example;
 set sashelp.class;
 length query $32000;
 query = "sashelp.class (where=(sex = '" || sex || "' and age = " || put(age,2.) || "))";
 count = GetSQLans (query, "name");
 drop query;
run;
proc print data=example;
run;
```

In the results the COUNT column contains how many other members of this dataset are there with this age and sex.

Obs	NAME	SEX	AGE	HEIGHT	WEIGHT	COUNT
1	Alfred	M	14	69.0	112.5	2
2	Alice	F	13	56.5	84.0	2
3	Barbara	F	13	65.3	98.0	2
4	Carol	F	14	62.8	102.5	2
5	Henry	M	14	63.5	102.5	2
6	James	M	12	57.3	83.0	3
7	Jane	F	12	59.8	84.5	2
8	Janet	F	15	62.5	112.5	2
9	Jeffrey	M	13	62.5	84.0	1
10	John	M	12	59.0	99.5	3
11	Joyce	F	11	51.3	50.5	1
12	Judy	F	14	64.3	90.0	2
13	Louise	F	12	56.3	77.0	2
14	Mary	F	15	66.5	112.0	2
15	Philip	M	16	72.0	150.0	1
16	Robert	M	12	64.8	128.0	3
17	Ronald	M	15	67.0	133.0	2
18	Thomas	M	11	57.5	85.0	1
19	William	M	15	66.5	112.0	2

Applying the user written function to the contemporaneous event requirement

The initial code was now adapted in the following respects:

- The output dataset is for those AEs that satisfy one of the terms of the respective algorithm
- If a simple term is satisfied then this AE is output
- If, for a non-simple term (for example "B or C"), the first element (in this example "B") is satisfied then a search is made for all the other events belonging to this patient with the other elements in this term (in this example "C") within a specified time period of the first event. The function is then used to resolve the term and - if true - the event and relevant associated information is output. This aspect of the code is shown in the code section below.

Note: the "Sum(Category Term Weight)" element is capable of being coded in this method also. For reasons of space It has not been included in this paper.

```
/* for each clause assess perform the following steps                                    */
/* 1. does the observation under consideration relate to the first category letter A, B or whatever in this clause ? */
/*    if not then clause result is 0 and move onto next item, if it is then move onto step 2.          */
/* 2. for all the following category letters in the clause check for each letter :                    */
/*     - if this patient had an AE in this category within the date range (plus margin) of the AE found in step 1    */
/*     - substitute the result of this check into the clause in place of the category letter          */
   do i = 1 to &__nclause;
     /* loop through all the categories in a term */
     catcount = 0;
     firstcatres = 0;
     firstcatst = .;
     firstcaten = .;
     format firstcatst firstcaten date9.;
     length nextcat $1;
     nextcatpos = indexc(__clauseanal[i],'ABCDEFGH');
     do while (nextcatpos);
       catcount = catcount + 1;
       nextcat = substr(__clauseanal[i],nextcatpos,1);
       nextcatres = '0';
       nextcatidx = index("&&__cats&i",nextcat);   /* the array index where the terms for this cat stored */

       if catcount = 1 then do;  /* if first time through loop then check terms for this category - are any satisfied ? */

         do j = 1 to __termdef[nextcatidx];
           if aeptcd = input(__term[nextcatidx,j],best.) then do;
             put ' hit ! ' usubjid i=  nextcat= nextcatidx= astdt= aendt= aedecod= __clauseanal[i]= __clause[i]=;
             firstcat    = nextcat;
```

```
          firstcatres = 1;
          firstcatst  = astdt - &margin;
          if missing(aendt) then
            firstcaten = today();
          else
            firstcaten = min(aendt + &margin, today());
        end;
      end;
      if firstcatres then nextcatres = '1';
    end;
    else do;  /* if subsequent iteration of loop -->  continue to check the rest of the categories in term - */
      if firstcatres then do;
        put 'check if this patient has any items from cat ' nextcat ' within dates from ' firstcatst ' to ' firstcaten;
        /* construct SQL query        */
        length query $10000;
        query = "ae_raw (where=(usubjid='" || trim(usubjid) || "' and "
            || "(   ( '" || put(firstcatst,date9.) || "'d <= astdt <= '" || put(firstcaten,date9.) || "'d ) "   /* date element */
            || " or ( '" || put(firstcatst,date9.) || "'d <= aendt <= '" || put(firstcaten,date9.) || "'d ) "
            || " or ( astdt < '" || put(firstcatst,date9.) || "'d and aendt > '" || put(firstcaten,date9.) || "'d ) )"
            || " and aeptcd in ( ";
        do j = 1 to __termdef[nextcatidx];              /* loop through terms for NEXTCAT */
          if j > 1 then query = trim(query) || ",";
          query = trim(query) || " " || trim(__term[nextcatidx,j]) ;
        end;
        query = trim(query) || ") ))";
        count = GetSQLans (query, "usubjid");         /* this performs the SQL query */
        if count > 0 then nextcatres = '1';
      end;
    end;

    if firstcatres then  put __clauseanal[i]= nextcatpos= nextcat= nextcatres=;
    __clauseanal[i] = tranwrd(__clauseanal[i],nextcat,nextcatres);
    nextcatpos = indexc(__clauseanal[i],'ABCDEFGH');
  end;
  if firstcatres then do;
    length smqalgo_eval $100;
    smqalgo_eval = '%eval( ' || trim(__clauseanal[i]) || ' )';
    __clauseres[i] = input(resolve(smqalgo_eval),best.);
    if __clauseres[i] then do;  /* if this clause is satisfied then output */
      clause_satisfied = __clause[i];
      output;
    end;
  end;
end;  /* of loop through all clauses */
```

This is an extract of a larger data step – for the rest of the code see appendix 4.

A review of techniques used to solve an algorithm

There are a few points to highlight from the above generalised code:

- The RESOLVE function is a good way in SAS to take an algorithm defined in a text field and find the Boolean result in such a way that program execution can be controlled on an observation level.
- Hash table functionality and response times can be problematic when loading large amounts of data into memory; it would have been a solution to the contemporaneous problem to use a hash table to read all the AEs for a patient but this was rejected for those reasons.
- The use of hash tables is restricted to reading the MedDRA information, this is considered an appropriate use of hash tables and unlikely to cause memory issues (an alternative method could have been to pre-load codes into arrays via PROC TRANSPOSE but this is not as efficient as hash tables and make debugging more complicated.
- The user-written function presented here and in other papers is an extremely powerful method of accessing data. Anything that can be put in an SQL query can be used on an observation-by-observation basis in the SAS data step (see paper AD11 from PHUSE 2023 "*Creating applications with mash-ups: user written functions, stored sql queries, and interfaces*" for some more examples of this).

- The attributes of the algorithmic SMQ definitions have been exploited when programming these generalised solutions; the clauses in each algorithm are separated by "and"s, and within each clause the first category must always be satisfied for the entire clause to be evaluated as true (for example "D and (B or C)").

CONCLUSION

Using AI

AI has the capacity to understand complex problems providing you ask very precise questions. AI has the capacity to solve such problems based on test data and to provide SAS working SAS code. However, we noticed that AI is not there yet in terms of providing SAS code that would cover new data constellations, which is what you would expect from a SAS programmer. This paper also shows that changing the perspective about how to approach a problem can allow you to implement solutions that are easier to program. This is also something that can be expected from experienced SAS programmers.

Algorithm calculation and resolution

In resolving the algorithm for an SMQ we have perfect knowledge; something is either true or false and we do not have to make deductions. There may be other situations where we incorporate deductions (from a variety of software platforms and data sources) and then act upon them an algorithm. This paper shows how a range of techniques can be used to facilitate data gathering and reasoning beyond what is normally considered possible in a SAS data step.

Generalization

This paper focuses on SQMs, which is one example of “identifying cases” among many examples that SAS programmers can have to deal with. Identifying cases can range from trivial to very complicated. As programmers, we can deal with many complications such as adding time constraints, scope, weight, medical history, lab results etc... There are practically no limits in terms of what can be achieved.

REFERENCES

Use the Full Power of SAS in Your Function-Style Macros, Mike Rhoads, Paper 004-2012, SAS Global Forum 2012
Creating applications with mash-ups: user written functions, stored SQL queries, and interfaces, Steve Prust, Paper AD11, PHUSE EU Connect 2023

Introductory Guide for Standardised MedDRA Queries (SMQs) Version 27.0
https://admin.meddra.org/sites/default/files/guidance/file/SMQ_intguide_27_0_English.pdf

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:
Corentin Le Scanff & Stephen Prust
Elderbrook Solutions Gmbh
Borsigstraße 4, 74321 Bietigheim-Bissingen, Germany
Work Phone: +49 7142 339 29 19
Email:corentin.lescanff@elderbrooksolutions.com
Email:stephen.prust@elderbrooksolutions.com

Appendix 1: Code generated by ChatGPT

```
/* Load the dataset */
data dress_data;
    infile '/path_to_your_data/my_data4.txt' delimiter='09'x dsd firstobs=2;
    input STUDYID $ USUBJID $ COMMENT $ AEPTCD AEDECOD $ ASTDT :date9. AENDT :date9.;
run;

/* Define categories based on AEDECOD */
data dress_data_categorized;
    set dress_data;
    format ASTDT AENDT date9.;

    /* Define categories */
    if AEDECOD = "Drug reaction with eosinophilia and systemic symptoms" then Category_A = 1;
    else Category_A = 0;

    if AEDECOD = "Rash" then Category_B = 1;
    else Category_B = 0;

    if AEDECOD = "Pyrexia" then Category_C = 1;
    else Category_C = 0;

    if AEDECOD = "Lymphadenopathy" then Category_D = 1;
    else Category_D = 0;

    if AEDECOD = "Eosinophilia" then Category_E = 1;
    else Category_E = 0;
run;

/* Sort data by USUBJID and date */
proc sort data=dress_data_categorized;
    by USUBJID ASTDT AENDT;
run;

/* Check for patients meeting the criteria */
data dress_patients;
    set dress_data_categorized;
    by USUBJID;

    retain has_B has_C has_D has_E first_A first_B first_C first_D first_E;
    format first_A first_B first_C first_D first_E date9.;

    if first.USUBJID then do;
        has_B = 0; has_C = 0; has_D = 0; has_E = 0;
        first_A = .; first_B = .; first_C = .; first_D = .; first_E = .;
    end;

    /* Criterion 1: Direct reference to DRESS (Category A) */
    if Category_A = 1 then do;
        first_A = ASTDT;
        output;
    end;

    /* Collect first occurrence of each category */
    if Category_B = 1 and has_B = 0 then do; first_B = ASTDT; has_B = 1; end;
    if Category_C = 1 and has_C = 0 then do; first_C = ASTDT; has_C = 1; end;
    if Category_D = 1 and has_D = 0 then do; first_D = ASTDT; has_D = 1; end;
    if Category_E = 1 and has_E = 0 then do; first_E = ASTDT; has_E = 1; end;
```

```
    /* Criterion 2: B + C + D within 30 days */
    if has_B = 1 and has_C = 1 and has_D = 1 then do;
       if max(first_B, first_C, first_D) - min(first_B, first_C, first_D) <= 30 then do;
          output;
       end;
    end;

    /* Criterion 3: B + C + E within 30 days */
    if has_B = 1 and has_C = 1 and has_E = 1 then do;
       if max(first_B, first_C, first_E) - min(first_B, first_C, first_E) <= 30 then do;
          output;
       end;
    end;

    /* Criterion 4: B + D + E within 30 days */
    if has_B = 1 and has_D = 1 and has_E = 1 then do;
       if max(first_B, first_D, first_E) - min(first_B, first_D, first_E) <= 30 then do;
          output;
       end;
    end;

    if last.USUBJID then do;
       has_B = 0; has_C = 0; has_D = 0; has_E = 0;
       first_A = .; first_B = .; first_C = .; first_D = .; first_E = .;
    end;

run;

/* Output the results */
proc print data=dress_patients;
    var USUBJID STUDYID first_A first_B first_C first_D first_E;
run;
```

Appendix 2: Code generated by programmer

```
    *** Select SQM terms ***;
    proc sort data=dict.msmqbase (where=(MEDRAVER = "&mver." and MUDAEC="&mudaec"))
              out=smq0 (keep=mtmcd mtm TERMCAT SMQALGO MUDAECCD MUDAEC);
        by mtmcd mtm;
    run;

    data smq0;
        set smq0;
        temp = 1;
        aeptcd = input(mtmcd,best.);
    run;

    *** Get algorythms letters ***;
    proc transpose data=smq0
                   out=smq1 (drop=_NAME_) prefix=var_;
        by MUDAECCD MUDAEC aeptcd mtm;
        id TERMCAT;
        var temp;
    run;

    *** Get AE data ***;
    proc sort data=&indata
              out=__adae0 (keep=studyid usubjid trtsdt trtedt aedecod aeptcd astdt aendt);
        by aeptcd studyid usubjid;
    run;

    *** Flag AE with letters ***;
```

```
    data __adae1;
        merge __adae0 (in=a)
            smq1    (in=b);
        by aeptcd;
        if a;
    run;

    *** Take AE data as well as treatment period ***;
    proc sort data=__adae0
        out=__adae00 (keep=studyid usubjid trtsdt trtedt) nodupkey;
        by studyid usubjid trtsdt trtedt;
    run;
    *** Clean ***;
    data __adae00;
        set __adae00;
        if missing(trtsdt) or missing(trtedt) then delete;
    run;

    *** Compute all windows ***;
    data __adae01;
        set __adae00;
        format date_low date_high date9.;
        do date_low = (trtsdt-&adjacent) to (trtedt);
            date_high = date_low + &adjacent;
            output;
        end;
    run;

    *** Check if any AE is inside a specific window ***;
    proc sql noprint;
        create table adae_algo0 as select a.*, var_a, var_b, var_c, var_d, var_e, astdt, aedecod from __adae01 as
a join __adae1 as b
        on (date_low <= b.ASTDT <= date_high or date_low <= b.AENDT <= date_high) and a.usubjid = b.usubjid;

        create table adae_algo1 as select studyid, usubjid, date_low, date_high,
            max(var_a) as max_a,
            max(var_b) as max_b,
            max(var_c) as max_c,
            max(var_d) as max_d,
            max(var_e) as max_e
            from adae_algo0
            group by studyid, usubjid, date_low, date_high;
    quit;

    *** Identify dates for which UDAEC is identified ***;
    data adae_algo2;
        set adae_algo1;
        if max_b = 1 and max_c = 1 and max_d = 1 then BCD = 1;
        if max_b = 1 and max_c = 1 and max_e = 1 then BCE = 1;
        if max_b = 1 and max_d = 1 and max_e = 1 then BDE = 1;
        a = max_a;
          if a or bcd or bce or bde then smq = 1;
          if smq = 1 then output;
    run;

    data adae1_a;         set __adae1;       where var_a = 1; run;
    data adae1_b;         set __adae1;       where var_b = 1; run;
    data adae1_c;         set __adae1;       where var_c = 1; run;
    data adae1_d;         set __adae1;       where var_d = 1; run;
    data adae1_e;         set __adae1;       where var_e = 1; run;
    data adae_algo_A;      set adae_algo2;   where A     = 1; run;
    data adae_algo_BCD;    set adae_algo2;   where BCD   = 1; run;
    data adae_algo_BCE;    set adae_algo2;   where BCE   = 1; run;
    data adae_algo_BDE;    set adae_algo2;   where BDE   = 1; run;
```

```
proc sql noprint;

    create table adae_a as select distinct a.*, b.smq from adae1_a as a join adae_algo_A as b
    on (date_low <= ASTDT <= date_high or date_low <= AENDT <= date_high) and a.usubjid = b.usubjid;


    create table adae_bcd_b as select distinct a.*, b.smq from adae1_b as a join adae_algo_BCD as b
    on (date_low <= ASTDT <= date_high or date_low <= AENDT <= date_high) and a.usubjid = b.usubjid;

    create table adae_bcd_c as select distinct a.*, b.smq from adae1_c as a join adae_algo_BCD as b
    on (date_low <= ASTDT <= date_high or date_low <= AENDT <= date_high) and a.usubjid = b.usubjid;

    create table adae_bcd_d as select distinct a.*, b.smq from adae1_d as a join adae_algo_BCD as b
    on (date_low <= ASTDT <= date_high or date_low <= AENDT <= date_high) and a.usubjid = b.usubjid;


    create table adae_bce_b as select distinct a.*, b.smq from adae1_b as a join adae_algo_BCE as b
    on (date_low <= ASTDT <= date_high or date_low <= AENDT <= date_high) and a.usubjid = b.usubjid;

    create table adae_bce_c as select distinct a.*, b.smq from adae1_c as a join adae_algo_BCE as b
    on (date_low <= ASTDT <= date_high or date_low <= AENDT <= date_high) and a.usubjid = b.usubjid;

    create table adae_bce_e as select distinct a.*, b.smq from adae1_e as a join adae_algo_BCE as b
    on (date_low <= ASTDT <= date_high or date_low <= AENDT <= date_high) and a.usubjid = b.usubjid;


    create table adae_bde_b as select distinct a.*, b.smq from adae1_b as a join adae_algo_BDE as b
    on (date_low <= ASTDT <= date_high or date_low <= AENDT <= date_high) and a.usubjid = b.usubjid;

    create table adae_bde_c as select distinct a.*, b.smq from adae1_d as a join adae_algo_BDE as b
    on (date_low <= ASTDT <= date_high or date_low <= AENDT <= date_high) and a.usubjid = b.usubjid;

    create table adae_bde_e as select distinct a.*, b.smq from adae1_e as a join adae_algo_BDE as b
    on (date_low <= ASTDT <= date_high or date_low <= AENDT <= date_high) and a.usubjid = b.usubjid;

quit;

data to_be_added;
    set adae_a
        adae_bcd_:
        adae_bce_:
        adae_bde_:;
    drop var:;
    &prefix.NAM = MUDAEC;
    &prefix.CD  = MUDAECCD;
    &prefix.SC  = "BROAD";
run;

proc sort data=to_be_added nodupkey;
    by _ALL_;
run;

proc sort data=to_be_added (drop=TRTSDT TRTEDT MUDAEC MUDAECCD SMQ MTM);
    by studyid usubjid AEPTCD aedecod ASTDT AENDT;
run;

proc sort data=&indata out=orig_data;
   by studyid usubjid AEPTCD aedecod ASTDT AENDT;
run;

data &outdata;
    merge orig_data   (in=a)
        to_be_added (in=b);
   by studyid usubjid AEPTCD aedecod ASTDT AENDT;
    if a;
run;
```

```
    proc datasets nolist nodetails;
         delete adae_algo: adae_b: adae_a adae1: smq0 smq1 __adae: orig_data to_be_added;
    run;quit;
```

Appendix 3: Resolve function

```
%* list of algorithmic smqs;
proc sort data=meddra_dict (where=(smqalgo ne 'N' and termstat='A' and smqstat='A')) out=smqalgo_u
(keep=mudaeccd mudaec smqalgo) nodupkey;
  by mudaeccd mudaec smqalgo;
run;
data _null_;
  set smqalgo_u end=last;
  call symput ('__algo'||compress(put(_n_,best.)),trim(mudaeccd));
  call symput ('__algoname'||compress(put(_n_,best.)),trim(mudaec));
  call symput ('__algodef'||compress(put(_n_,best.)),trim(smqalgo));
  if last then call symput ('__nalgo',compress(put(_n_,best.)));
run;

%macro smqalgo_loop ();

%do i = 1 %to &__nalgo;
  %put Processing SMQ code: &&__algo&i, Algorithm: &&__algodef&i, SMQ name: &&__algoname&i   ;
  proc sort data=dict out=smqalgo&i (keep=termscop termcat termwght smqlevel mudaeccd mudaec mlevel mtm
mtmcd);
    by mudaeccd mudaec smqalgo;
    where smqalgo ne 'N' and termstat='A' and smqstat='A' and mudaeccd="&&__algo&i";
  run;
  %local __cats&i;
  proc sql noprint;
    select distinct termcat into :__cats&i separated by "" from smqalgo&i order by termcat;
    select countcd into :lim from
        (select termcat, count(mtmcd) as countcd from smqalgo&i group termcat)
      having countcd=max(countcd) order by termcat ;
  quit;

  %let ncats = %length(&&__cats&i);
  %put &=ncats __cats&i=&&__cats&i &=lim;
  %let ncatslim = %eval(&ncats*&lim);

  data ae_smq_&i;
    attrib
      termcat        length=$1    format=$1.            label="Term category"
      termwght       length=8     format=8.             label="Term weight"
      smqalgo        length=$60   format=$60.             label="SMQ algorithm"
      mudaeccd       length=$10   format=$10.             label="MedDRA user defined AE category code"
      mudaec         length=$200  format=$200.            label="MedDRA user defined AE category"
      mlevel         length=$10   format=$10.           label="MedDRA level"
      mtmcd          length=$10   format=$10.             label="MedDRA term code"
      mtm            length=$200  format=$200.            label="MedDRA term"
    ;
    set ae_raw ;
    by usubjid;
    /* codes that belong to each category */
    length term1-term&ncatslim $10;
    retain term1-term&ncatslim ;
    array __term [&ncats,&lim] term1-term&ncatslim;

    /* number of terms in each category, 0 indicates there are no items to check */
    length termdef1-termdef&ncats 8.;
    retain termdef1-termdef&ncats;
    array __termdef [&ncats] termdef1-termdef&ncats;
```

```
/* result of search for terms in a category */
length termres1-termres&ncats 8.;
retain termres1-termres&ncats;
array __termres [&ncats] termres1-termres&ncats;

/* term weights for each category - applies if sum of category term weights is used */
length termwgt1-termwgt&ncats 8.;
retain termwgt1-termwgt&ncats;
array __termwgt [&ncats] termwgt1-termwgt&ncats;

if _n_ = 1 then do;
  call missing(mudaeccd, mudaec, smqalgo, termcat, termwght, mtm, mtmcd, mlevel);

  /* hash table for SMQ def */
  declare hash dictdef(dataset:"dict", multidata: 'y');
  dictdef.defineKey("mudaeccd");
  dictdef.defineData("mudaec", "smqalgo");
  dictdef.defineDone();

  mudaeccd = "&&__algo&i";

  rc = dictdef.find();
  if rc=0 then do;
    put mudaec= mudaeccd= smqalgo= ;
    smqalgor = smqalgo;
    retain smqalgor;
    /* hash table for SMQ components */
    declare hash dict(dataset:"dict", multidata: 'y');
    dict.defineKey("mudaeccd", "termcat");
    dict.defineData("mtm", "mtmcd", "mlevel", "termwght");
    dict.defineDone();

    do i = 1 to &ncats;
      termcat = substr("&&__cats&i",i,1);
      __termdef[i] = 0;
      n = 0;
      rc2 = dict.find();
      do while (rc2=0);
        n = n+1;
        __term[i,n]  = mtmcd;
        __termdef[i] = n;
        __termwgt[i] = termwght;
        rc2 = dict.find_next();
      end;
      put termcat= __termdef[i]=  __termwgt[i]=;
    end;

  end;
  else do;
    put "%str(ERR)OR: Key not found";
    goto leave;
  end;

end;

if first.usubjid then do;
  do i = 1 to &ncats;      __termres[i]=0;    end;
end;
do i = 1 to &ncats;
  do j = 1 to __termdef[i];
    if aeptcd = input(__term[i,j],best.) then do;
      __termres[i] = 1;
    end;
  end;
```

```
    end;
    if last.usubjid then do;
      smqalgo_eval = '%eval( ' || trim(smqalgor) || ' )';
      if index(smqalgor,'Sum') then do;
        sumwgt = 0;
        do i = 1 to 9;
          if __termwgt[i] > 0 and __termres[i] > 0 then sumwgt = sumwgt + __termwgt[i];
        end;
        smqalgo_eval = tranwrd(smqalgo_eval,'Sum(Category Term Weight)',compress(put(sumwgt,best.)));
      end;
      do i = 1 to &ncats;
        termcat = substr("&&__cats&i",i,1);
        if index(smqalgo_eval,trim(termcat)) then smqalgo_eval = tranwrd(smqalgo_eval,trim(termcat),
compress(put(__termres[i],best.)) );
      end;
      smqresult = input(resolve(smqalgo_eval),best.);
      length cats $20;
      cats = "&&__cats&i";
      output;
    end;
    keep usubjid cats smqalgor smqalgo_eval termdef1-termdef&ncats termres1-termres&ncats termwgt1-
termwgt&ncats sumwgt smqresult;

    leave:
  run;
  %end;

%end;

%mend;
```

Appendix 4: Generalized solution

```
proc sort data=dict (where=(smqalgo ne 'N' and termstat='A' and smqstat='A')) out=smqalgo_u2 (keep=mudaeccd
mudaec smqalgo) nodupkey;
  by mudaeccd mudaec smqalgo;
run;
data _null_;
  set smqalgo_u end=last;
  where smqalgo ne 'A or Sum(Category Term Weight)>6' ;
  call symput ('__algo'||compress(put(_n_,best.)),trim(mudaeccd));
  call symput ('__algoname'||compress(put(_n_,best.)),trim(mudaec));
  call symput ('__algodef'||compress(put(_n_,best.)),trim(smqalgo));
  if last then call symput ('__nalgo',compress(put(_n_,best.)));
run;


%macro smqalgo_loopx (runcode = 0, margin=30);

%do i = 1 %to &__nalgo;
  %put Processing SMQ code: &&__algo&i, Algorithm: &&__algodef&i, SMQ name: &&__algoname&i   ;
  %put ==========================================================
  ==========================================;
  options nonotes;
  data _null_;
    length clause clausetype $50 smqalgo $200;
    smqalgo = tranwrd("&&__algodef&i", '(' , ' ( ' );
    smqalgo = tranwrd(smqalgo, ')' , ' ) ' );

    clausecount = 0;
    i = 1 ;
    code = 0;
    level = 0;
    length canditem $20;
    canditem = scan(smqalgo,i," ");
```

```
    clause = ' ';
    do while (canditem ne ' ');
      if canditem = '(' then level = level + 1;
      if canditem = ')' then level = level - 1;
      if level = 0 and canditem = 'or' then do;
        link addlevel;
      end;
      else clause = trim(left(clause)) || ' ' || canditem;

      i=i+1;
      canditem = scan(smqalgo,i," ");
    end;
    link addlevel;

    call symput ("__nclause",compress(put(clausecount,best.)));
  return;

  addlevel:
       clausecount = clausecount + 1;
       clause = left(clause);
       if substr(clause,1,1) = '(' and substr(clause,length(clause),1) = ')' then clause =
left(substr(clause,2,length(clause)-2));
       call symput ("__clause" || compress(put(clausecount,best.)) , trim(clause) );
       clause = ' ';
  return;
  run;
  options notes;

  %put Number of distinct clauses: &__nclause;
  %do j = 1 %to &__nclause;
    %put clause &j: &&__clause&j ;
  %end;

  proc sort data=dict out=smqalgo&i (keep=termscop termcat termwght smqlevel mudaeccd mudaec mlevel mtm
mtmcd);
    by mudaeccd mudaec smqalgo;
    where smqalgo ne 'N' and termstat='A' and smqstat='A' and mudaeccd="&&__algo&i";
  run;
  %local __cats&i;
  proc sql noprint;
    select distinct termcat into :__cats&i separated by "" from smqalgo&i order by termcat;
    select countcd into :lim from
       (select termcat, count(mtmcd) as countcd from smqalgo&i group termcat)
      having countcd=max(countcd) order by termcat ;
  quit;

  %let ncats = %length(&&__cats&i);
  %put &=ncats __cats&i=&&__cats&i &=lim;
  %let ncatslim = %eval(&ncats*&lim);

  %if &runcode %then %do;

  data ae_smq_&i;
    attrib
      termcat        length=$1    format=$1.            label="Term category"
      termwght       length=8     format=8.             label="Term weight"
  /*    smqlevel       length=8     format=1.             label="SMQ Level"*/
      smqalgo        length=$60   format=$60.           label="SMQ algorithm"
      mudaeccd       length=$10   format=$10.           label="MedDRA user defined AE category code"
      mudaec         length=$200  format=$200.          label="MedDRA user defined AE category"
      mlevel         length=$10   format=$10.           label="MedDRA level"
      mtmcd          length=$10   format=$10.           label="MedDRA term code"
      mtm            length=$200  format=$200.          label="MedDRA term"
    ;
    set ae_raw ;
    by usubjid;
```

```
/* clauses in algorithm */
length clause1-clause&__nclause $50;
retain clause1-clause&__nclause ;
array __clause [&__nclause] clause1-clause&__nclause;

/* location to analyses clauses in algorithm for each observation */
length clauseanal1-clauseanal&__nclause $50;
retain clauseanal1-clauseanal&__nclause ;
array __clauseanal [&__nclause] clauseanal1-clauseanal&__nclause;

/* clauses result */
length clauseres1-clauseres&__nclause 8.;
retain clauseres1-clauseres&__nclause ;
array __clauseres [&__nclause] clauseres1-clauseres&__nclause;

/* codes that belong to each category */
length term1-term&ncatslim $10;
retain term1-term&ncatslim ;
array __term [&ncats,&lim] term1-term&ncatslim;

/* number of terms in each category, 0 indicates there are no items to check */
length termdef1-termdef&ncats 8.;
retain termdef1-termdef&ncats;
array __termdef [&ncats] termdef1-termdef&ncats;

/* term weights for each category - applies if sum of category term weights is used */
length termwgt1-termwgt&ncats 8.;
retain termwgt1-termwgt&ncats;
array __termwgt [&ncats] termwgt1-termwgt&ncats;

if _n_ = 1 then do;

 %do j = 1 %to &__nclause;
  __clause[&j] = "&&__clause&j";
 %end;

 call missing(mudaeccd, mudaec, smqalgo, termcat, termwght, mtm, mtmcd, mlevel);

 /* hash table for SMQ def */
 declare hash dictdef(dataset:"dict", multidata: 'y');
 dictdef.defineKey("mudaeccd");
 dictdef.defineData("mudaec", "smqalgo");
 dictdef.defineDone();

 mudaeccd = "&&__algo&i";

 rc = dictdef.find();
 if rc=0 then do;
  put mudaec= mudaeccd= smqalgo= ;
  /* hash table for SMQ components */
  declare hash dict(dataset:"dict", multidata: 'y');
  dict.defineKey("mudaeccd", "termcat");
  dict.defineData("mtm", "mtmcd", "mlevel", "termwght");
  dict.defineDone();

  do i = 1 to &ncats;
   termcat = substr("&&__cats&i",i,1);
   __termdef[i] = 0;
   n = 0;
   rc2 = dict.find();
   do while (rc2=0);
    n = n+1;
    __term[i,n]  = mtmcd;
    __termdef[i] = n;
    __termwgt[i] = termwght;
    rc2 = dict.find_next();
```

```
        end;
        put termcat= __termdef[i]=  __termwgt[i]=;
      end;

    end;
    else do;
      put "%str(ERR)OR: Key not found";
      goto leave;
    end;

  end;

  /* at the start prepare the clause analysis area and set all clause results to 0 [false] */
  do i = 1 to &__nclause;
    __clauseanal[i] = __clause[i];
    __clauseres[i] = 0;
  end;

  /* for each clause assess perform the following steps                                    */
  /* 1. does the observation under consideration relate to the first category letter A, B or whatever in this clause ?
*/
  /*    if not then clause result is 0 and move onto next item, if it is then move onto step 2.          */
  /* 2. for all the following category letters in the clause check for each letter :                    */
  /*     - if this patient had an AE in this category within the date range (plus margin) of the AE found in step 1    */
  /*     - substitute the result of this check into the clause in place of the category letter          */
  do i = 1 to &__nclause;

    /* loop through all the categories in a term */
    catcount = 0;
    firstcatres = 0;
    firstcatst = .;
    firstcaten = .;
    format firstcatst firstcaten date9.;
    length nextcat $1;
    nextcatpos = indexc(__clauseanal[i],'ABCDEFGH');
    do while (nextcatpos);
      catcount = catcount + 1;
      nextcat = substr(__clauseanal[i],nextcatpos,1);
      nextcatres = '0';
      nextcatidx = index("&&__cats&i",nextcat);   /* the array index where the terms for this cat stored */

      if catcount = 1 then do;  /* if first time through loop then check terms for this category - are any satisfied ? */

        do j = 1 to __termdef[nextcatidx];
          if aeptcd = input(__term[nextcatidx,j],best.) then do;
            put ' hit ! ' usubjid i=  nextcat= nextcatidx= astdt= aendt= aedecod= __clauseanal[i]= __clause[i]=;
            firstcat    = nextcat;
            firstcatres = 1;
            firstcatst  = astdt - &margin;
            if missing(aendt) then
              firstcaten = today();
            else
              firstcaten = min(aendt + &margin, today());
          end;
        end;
        if firstcatres then nextcatres = '1';
      end;
      else do;  /* if subsequent iteration of loop -->  continue to check the rest of the categories in term - */
        if firstcatres then do;
          put 'check if this patient has any items from cat ' nextcat ' within dates from ' firstcatst ' to ' firstcaten;
          /* construct SQL query        */
          length query $10000;
          query = "ae_raw (where=(usubjid='" || trim(usubjid) || "' and "
              || "(   ( '" || put(firstcatst,date9.) || "'d <= astdt <= '" || put(firstcaten,date9.) || "'d ) "   /* date element */
              || " or ( '" || put(firstcatst,date9.) || "'d <= aendt <= '" || put(firstcaten,date9.) || "'d ) "
              || " or ( astdt < '" || put(firstcatst,date9.) || "'d and aendt > '" || put(firstcaten,date9.) || "'d ) )"
```

```
              || '' and aeptcd in ( '';
            do j = 1 to __termdef[nextcatidx];              /* loop through terms for NEXTCAT */
              if j > 1 then query = trim(query) || ',';
              query = trim(query) || ' ' || trim(__term[nextcatidx,j]) ;
            end;
          query = trim(query) || '') ))';
          count = GetSQLans (query, "usubjid");
          if count > 0 then nextcatres = '1';
        end;
      end;

      if firstcatres then  put __clauseanal[i]= nextcatpos= nextcat= nextcatres=;
      __clauseanal[i] = tranwrd(__clauseanal[i],nextcat,nextcatres);
      nextcatpos = indexc(__clauseanal[i],'ABCDEFGH');
    end;
    if firstcatres then do;
      length smqalgo_eval $100;
      smqalgo_eval = '%eval( ' || trim(__clauseanal[i]) || ' )';
      __clauseres[i] = input(resolve(smqalgo_eval),best.);
      put "Resolve this ... " __clauseanal[i]= smqalgo_eval= __clauseres[i]=;
      if __clauseres[i] then do;  /* if this clause is satisfied then output */
        clause_satisfied = __clause[i];
        output;
      end;

    end;
  end;  /* of loop through all clauses */
  /* variables that might be kept to allow date checks to also be applied - for each term (A, B, C ...) keep the
AESEQ(s) and maybe ASTDT(s), AENDT(s) and so on */
  keep usubjid aeptcd astdt aendt firstcat: clause_satisfied query smqalgo_eval;

  leave:
  return;

 run;
 %end;

%end;

%mend;
%smqalgo_loopx (runcode=1);
```