Paper 079-2009

Playing Favorites: How to Manage Date Conflicts When Some Date Ranges are Preferred Over Others

Eric C. Wong, Palo Alto Medical Foundation Research Institute, Palo Alto, CA

ABSTRACT

Data with dates often require reconciling conflicting date ranges. Sometimes a set of consecutive, non-overlapping date ranges needs to be created from a set of overlapping date ranges. This is easy when all date ranges are considered equal. However, if some date ranges are preferred over others, more thought is required. One example is reconciling clinical data and determining disease status from overlapping date ranges of normal and abnormal lab values. If abnormal lab values are of interest, then ranges of abnormal values are preferred over normal values when overlap exists. This paper provides one solution to any number of preferences, demonstrates that it executes in a reasonable time with time trial results, and provides a macro implementation.

Keywords: temporal abstraction, date reconciliation, date range conflicts, date range overlap.

INTRODUCTION

Data with dates often require reconciling conflicting date ranges. Sometimes a set of continuous, non-overlapping date ranges needs to be created from a set of overlapping date ranges. This is easily accomplished when all the date ranges are treated equally.[1] However, if some date ranges are preferred over others, more thought is required.

Motivating Example. The motivating example used throughout this paper will be a clinical example of creating a history of a patient's disease or condition over time. Suppose three data sources are available: laboratory values, medications history and physician notes. A timeline showing when a patient had a disease or condition can be built from this information. Normal and abnormal laboratory values, medication usage, or specific diagnoses by a physician can define date ranges of non-disease and disease status. But often these date ranges will be conflicting or overlapping, when instead a set of non-overlapping date ranges is required. Since date ranges of disease will be of greater interest, disease date ranges are *preferred* over non-disease date ranges when date range conflicts are reconciled.

Assumptions. I assume that every date range can be assigned a date range type. The example above is a situation when there are two types, disease and non-disease. For simplicity, I will assume that preferences always exist and are denoted *type 2 > type 1* when *type 2* is preferred. I also assume that preferences are transitive, that is, if *type 2 > type 1* and *type 3 > type 2*, then *type 3 > type 1*. And, I assume types are sorted in ascending order of preference, namely, *type n > type n-1 > … type 1*.

Types of date ranges. What types of date ranges appear? Date ranges that are not conflicting at all are called "disjoint." Among conflicting date ranges, I will define ranges called "touch," "overlap," and "subset," as illustrated below. Time periods that occur between date ranges, will be called "gap" date ranges.

Figure 1: Examples of date range types.

Definition	Illustration	Example data		
		date_left	date_right	class
Disjoint: No conflict or overlap.		1/1/2008	2/1/2008	1
		4/1/2008	6/1/2008	2
Touch: Two date ranges share one end point.		1/1/2008	4/1/2008	1
		4/1/2008	6/1/2008	2
Overlap: Two date ranges overlap over a range.		1/1/2008	4/1/2008	1
		2/1/2008	10/1/2008	2
Subset: One date range is a subset of another.		1/1/2008	6/1/2008	1
		2/1/2008	4/1/2008	2
Gap: Period between date ranges.		2/1/2008	4/1/2008	.

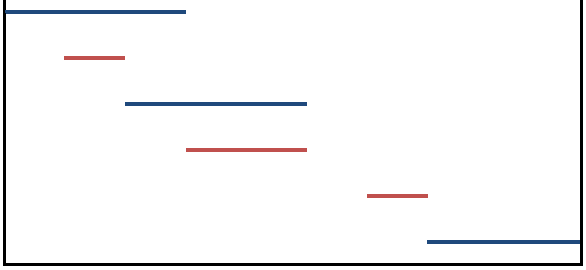
SOLUTION

The solution reconciles conflicting date ranges into a sequence of consecutive, non-overlapping date ranges. The illustration below contains non-overlapping, overlapping and gap date ranges. The desired, reconciled result is shown next.

Figure 2: Illustrations and examples of input and output data.

Input: Data with Conflicting Date Ranges

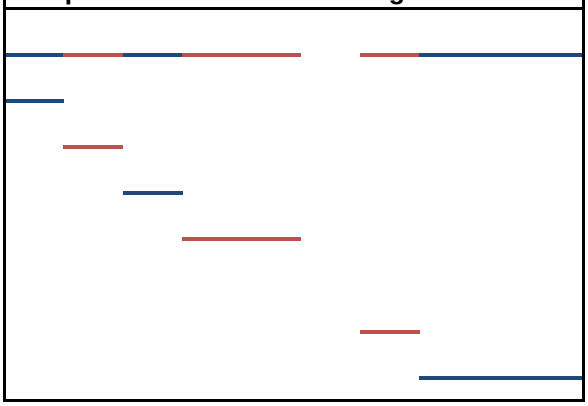
Illustration



Example Input Data

pat_id	date_left	date_right	type
11	1/1/2008	6/1/2008	1
11	3/1/2008	4/1/2008	2
11	4/1/2008	10/1/2008	1
11	6/1/2008	10/1/2008	2
11	12/1/2008	1/1/2009	2
11	1/1/2009	3/1/2009	1

Output: Reconciled Date Ranges



Example Output Data

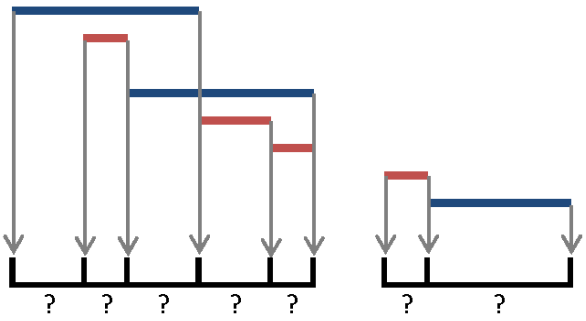
pat_id	date_left	date_right	type
11	1/1/2008	3/1/2008	1
11	3/1/2008	4/1/2008	2
11	4/1/2008	6/1/2008	1
11	6/1/2008	10/1/2008	2
11	10/1/2008	12/1/2008	.
11	12/1/2008	1/1/2009	2
11	1/1/2009	3/1/2009	1

Overview of Solution. The solution presented here contains three steps with an optional fourth step. It begins by breaking up the conflicting date ranges into non-conflicting date ranges. Next, it assigns date range types for all of these non-conflicting ranges. Finally, it collapses consecutive ranges of the same type to create the simplest representation of the date ranges.

STEP 1: DISCRETIZE INTO NON-OVERLAPPING DATE RANGES

Assume there is an input SAS® data set named `testset` that contains four variables: `pat_id`, `date_left`, `date_right`, `type` as in Figure 2. The `pat_id` variable uniquely identifies each patient. The `date_left` and `date_right` variables define the date range. And `type` indicates the date range type, where larger numeric values are preferred over smaller values.

Figure 3: Step 1 visual example.



The first goal, as illustrated to the right, is to break up the conflicting date ranges into smaller non-conflicting ranges as defined by any left and right endpoint. Visually, this can thought of as first "pulling down" the endpoints of the dates. Programmatically, it is allowing any end point to define new non-conflicting date ranges. This will be implemented through DATA steps, sorting, and later, the LAG function in step 2.

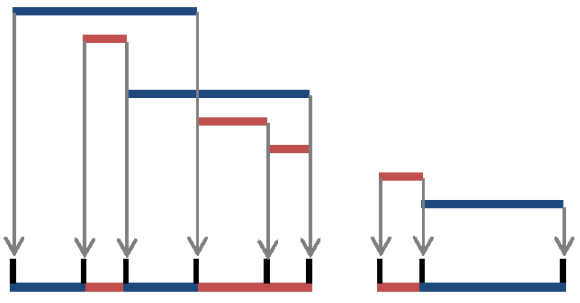
```
DATA work.step1;
   SET testset;
   date=date_left;  timept='start'; OUTPUT;
   date=date_right; timept='end';   OUTPUT;
   KEEP pat_id date timept type;
RUN;
```

At least one sort must be performed, and fortunately the only sort is performed here in Step 1. The THREADS option in the PROC SORT statement of the SORT procedure enables multi-threaded sorting which improve performance on multi-processor machines.

```
PROC SORT DATA=work.step1 OUT=work.step1b THREADS;
   BY pat_id date type DESCENDING timept;
RUN;
```

STEP 2: ASSIGN DATE RANGE TYPES

Figure 4: Step 2 visual example.



Next, date range types must be assigned according to the hierarchy of preferences that *type n* > *type n-1* > … *type 1*. Two components are important to accomplish this. First, the LAG function which returns the value previously stored for that variable, usually previous record's value, and second, an array that maintains what date range types are in use during any given date range. This array is called `type_status` and has indices over the possible date range types ranging from the macro variables `TYPE_MIN` and `TYPE_MAX`. The values in the array represent the number of date range types present within that period. For example, suppose four date ranges overlap within a period, three date ranges of *type 1* and one range of *type 2*. Then, `type_status[1]=3` and `type_status[2]=1`. These values are incremented and decremented when appropriate to represent types only present within the period being considered.

Under the imposed preferences, the date range type with the largest numeric value is preferred over others. Final date range type assignment is made by walking across the array in descending order from TYPE_MAX to TYPE_MIN and the first non-zero value in the array corresponds to the most preferred date range type. When all values of the array are zero, this indicates no types are present. A gap date range type defined by the macro variable TYPE_GAP, with a suggested value of missing ('.'), is assigned.

By the nature of how the data set is constructed, some spurious records are created for single-day date ranges, i.e. date ranges where date_left=date_right, must be removed. This is done with an IF statement at the end of the DATA step.

```
DATA work.step2;
   SET work.step1b (rename=(date=date_right
                            type=type_right
                            timept=timept_right));
   BY pat_id;

   date_left=LAG(date_right);
   type_left=LAG(type_right);
   timept_left=LAG(timept_right);

   IF NOT first.pat_id;
   FORMAT date_left date_right mmddyy10.;
RUN;

DATA work.step2b;
   SET work.step2;
   BY pat_id;

   * construct type status array;
   ARRAY type_status [&TYPE_MIN.:&TYPE_MAX.]
      type_status&TYPE_MIN.-type_status&TYPE_MAX.;
   IF (first.pat_id) THEN DO i=&TYPE_MIN. TO &TYPE_MAX.;
         type_status[i]=0;
   END;
        IF timept_left = 'start' THEN type_status[type_left]+1;
   ELSE IF timept_left = 'end'   THEN type_status[type_left]+(-1);

   * walk through array and assign type to ranges;
   DO i=&TYPE_MAX. TO &TYPE_MIN. BY -1 UNTIL (type_status[i] > 0);
   END;
   IF i>=0 THEN type=i;
   ELSE type=&TYPE_GAP.;

   * special handling for single-day date ranges;
   lag_date_right = LAG(date_right);
   lag_type       = LAG(type);
   IF ((NOT first.pat_id)
        AND (date_left=date_right)
        AND (date_left=lag_date_right)
        AND type<lag_type) THEN DELETE;

   KEEP pat_id date_left date_right type;
RUN;
```

STEP 3: COLLAPSE DATE RANGE TYPES

Neighboring date ranges of the same type should be collapsed together into one date range. Since the data is already sorted by pat_id and made up of consecutive non-overlapping ranges, this can be accomplished using the MEANS procedure with the NOTSORTED option in the BY statement. This forces PROC MEANS to produce statistics without sorting on the by-groups as defined on the BY statement. Taking the minimum of date_left and maximum of date_right when blocked by pat_id and type, produces the desired result.

```
PROC MEANS DATA=work.step2b MIN MAX NWAY NOPRINT;
   BY pat_id type NOTSORTED;
   OUTPUT OUT=work.step3 (DROP=_TYPE_ _FREQ_)
          MIN(date_left)=date_left
          MAX(date_right)=date_right;
RUN;
```

An example of the final data set is shown below.

Table 1: Example output data set

pat_id	date_left	date_right	type
1001	1/1/2008	3/1/2008	1
1001	3/1/2008	4/1/2008	2
1001	4/1/2008	6/1/2008	1
1001	6/1/2008	10/1/2008	2
1002	10/1/2008	12/1/2008	.
1002	12/1/2008	1/1/2009	2
1004	1/1/2009	3/1/2009	1

STEP 4: (OPTIONAL) MAKING DATE RANGES DISJOINT

Sometimes it is desirable to have consecutive date ranges begin on the next day. For example, suppose instead of the result given above, the following result is desired.

Table 2: Example output data step after optional step 4.

pat_id	date_left	date_right	type
1001	1/1/2008	**2/28/2008**	1
1001	3/1/2008	4/1/2008	2
1001	**4/2/2008**	**5/31/2008**	1
1001	6/1/2008	10/1/2008	2
1002	10/1/2008	**11/30/2008**	.
1002	12/1/2008	1/1/2009	2
1004	**1/2/2009**	3/1/2009	1

An additional DATA step and PROC MEANS to collapse the data is required.

```
* STEP 4 MAKE DISJOINT;
DATA work.step4      (where =(date_left <= date_right));
   SET work.step3    (rename=(date_left  = date_left_old
                              date_right = date_right_old
                              type       = type_old));
   BY pat_id;

   type1              = LAG2(type_old);
   date2_left         = LAG(date_left_old);
   date2_right        = LAG(date_right_old);
   type2              = LAG(type_old);
   date3_left         = date_left_old;
   date3_right        = date_right_old;
   type3              = type_old;
   lag_pat_id         = LAG(pat_id);
   lag_first_pat_id   = LAG(first.pat_id);

   * initialize;
   IF first.pat_id THEN DO;
         type1=.;
         date2_left=.;
         date2_right=.;
         type2=.;
   END;
   IF (lag_pat_id = pat_id) AND (lag_first_pat_id=1) THEN type1=.;

   * body: fit boundaries;
   IF (first.pat_id=0 AND last.pat_id=0) THEN DO;
         IF (type2 > type1 OR lag_first_pat_id=1)
                           THEN date_left  = date2_left;
         ELSE                   date_left  = date2_left + 1;
         IF (type2 > type3) THEN date_right = date2_right;
         ELSE                   date_right = date2_right - 1;
         type = type2;
         OUTPUT;
   END;

   /*ELSE IF (first.pat_id=1 AND last.pat_id=0) THEN DO; *do nothing;
   END;*/

   ELSE IF (first.pat_id=0 AND last.pat_id=1) THEN DO;
         IF (type2 > type1 OR lag_first_pat_id=1)
                           THEN date_left  = date2_left;
         ELSE                   date_left  = date2_left + 1;
         IF (type2 > type3) THEN date_right = date2_right;
         ELSE                   date_right = date2_right - 1;
         type = type2;
         OUTPUT;

         IF (type3 > type2) THEN date_left  = date3_left;
         ELSE                   date_left  = date3_left + 1;
         date_right = date3_right;
         type = type3;
         OUTPUT;
   END;

   ELSE IF (first.pat_id=1 AND last.pat_id=1) THEN DO;
         date_left  = date3_left;
         date_right = date3_right;
         type     = type3;
         OUTPUT;
   END;
```

```
   FORMAT date2_left date2_right
          date3_left date3_right
          date_left  date_right mmddyy10.;
   KEEP pat_id date_left date_right type;
RUN;

PROC MEANS DATA=work.step4 MIN MAX NWAY NOPRINT;
   BY pat_id type NOTSORTED;
   OUTPUT OUT=work.result (DROP=_TYPE_ _FREQ_)
          MIN(date_left)=date_left
          MAX(date_right)=date_right;
RUN;
```

AN ALTERNATIVE (BUT SLOWER) SOLUTION

Another alternative, albeit slower, solution is to output a single observation for every date within all date ranges. Then use PROC MEANS to assign date range types for each date then collapse date ranges. The source code is not presented due to space, but is available upon request. This alternative solution was used as a comparison in time trials.

TIME TRIAL RESULTS

For input data sets with over 5 million records, this solution executed in reasonable time ranging 20-30 seconds (CPU time) and 1-2 minutes (real time), and was significantly faster than the alternative solution mentioned above (70-300% faster). Time trials were performed on SAS® data sets of random data varying the following parameters: the number of IDs, the number of date ranges per ID, the number of date range types, the prevalence of overlaps, and the distribution of specific overlaps. This was repeated five times for each data set. Data sets with 2 date range types and 10 date range types were examined. All time results were similar. An abbreviated table of results is presented in the appendix. More detailed results are available upon request. Time trials were performed using SAS® 9.1.3, Enterprise Guide 4.1, on a server with quad core 2.4 GHz processors and 8 Gbs of memory.

CONCLUSION

This paper provides an efficient solution to hierarchical date range conflict reconciliation.

REFERENCES

1. Shannon D., Bannister W. “Overlapping Date Segments: How to Clean Up the Mess. VALSUG 2003.

ACKNOWLEDGMENTS

I thank Latha Palaniappan for her important support, Tom Makielski for his early input, and Laura Eaton, Lan Xiao, and Qiwen Huang for comments on the manuscript. This work was partly funded by a grant from the American Heart Association (0885049N).

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Eric C. Wong
Palo Alto Medical Foundation Research Institute
795 El Camino Real
Palo Alto, CA 94301
wonge@pamfri.org

APPENDIX

Each time trial was repeated five times on randomly generated data sets, and the mean and standard deviation (sd) are presented. Each data set had 100,000 unique patient ids with a random number of dates. Varying probabilities are denoted as $p_{conflict}$ (proportion of any date range conflict), p_{touch} (proportion of conflicting date ranges touching), $p_{overlap}$ (proportion of conflicting date ranges overlapping), p_{subset} (proportion of conflicting date ranges that are subsets).

Time Trial Results (abbreviated): 2 date range types

$p_{conflict}$	Distribution of conflicts			Input	System CPU time (m:ss)		Real Time (m:ss)	
	p_{touch}	$p_{overlap}$	p_{subset}	N records	mean	sd	mean	sd
20%	33%	33%	33%	5,795,283	0:33	0:01	1:53	0:03
	80%	10%	10%	5,793,866	0:31	0:01	1:54	0:03
	10%	80%	10%	5,798,091	0:32	0:01	1:58	0:04
	10%	10%	80%	5,800,282	0:33	0:01	1:59	0:03
80%	33%	33%	33%	5,798,521	0:22	0:00	1:28	0:05
	80%	10%	10%	5,801,018	0:22	0:01	1:27	0:01
	10%	80%	10%	5,795,767	0:23	0:00	1:28	0:04
	10%	10%	80%	5,800,040	0:22	0:00	1:27	0:06

Time Trial Results (abbreviated): 10 date range types

$p_{conflict}$	Distribution of conflicts			Input	System CPU time (m:ss)		Real Time (m:ss)	
	p_{touch}	$p_{overlap}$	p_{subset}	N records	mean	sd	mean	sd
20%	33%	33%	33%	5,795,283	0:39	0:01	2:11	0:03
	80%	10%	10%	5,793,866	0:39	0:01	2:11	0:04
	10%	80%	10%	5,798,091	0:39	0:01	2:18	0:04
	10%	10%	80%	5,800,282	0:39	0:01	2:17	0:02
80%	33%	33%	33%	5,798,521	0:30	0:01	1:48	0:03
	80%	10%	10%	5,801,018	0:31	0:01	1:49	0:04
	10%	80%	10%	5,795,767	0:30	0:01	1:49	0:02
	10%	10%	80%	5,800,040	0:28	0:00	1:43	0:05

Macro Implementation of Solution

```
/* MACRO: Solution
 * PARAMETERS:
       * INDSN:              input data set name
       * NAME_ID:            name of id variable
       * NAME_DATE_LEFT:     variable name for the left endpoint of a date range
       * NAME_DATE_RIGHT:    variable name for the right endpoint of a date range
       * TYPE_MIN:           the minimum value that the date range type assumes
       * TYPE_MAX:           the maximum value that the date range type assumes
       * TYPE_GAP:           the value to assign for gap date ranges
       * OUTDSN:             output data set name
 * DESCRIPTION: This macro reconciles overlapping date ranges.  These date ranges have
   a hierarchy imposed such that date range types of higher numeric value are
   preferred over those of lower numeric value.*/


%MACRO Solution( INDSN, NAME_ID, NAME_DATE_LEFT, NAME_DATE_RIGHT, NAME_TYPE,
                 TYPE_MIN, TYPE_MAX, TYPE_GAP, OUTDSN);

%* STEP 1: PULL DOWN;
DATA work.step1;
       LENGTH        &NAME_ID.     8.
                     date          8.
                     timept        $5.
                     &NAME_TYPE.   8.;
       SET &INDSN.;
       date=&NAME_DATE_LEFT.;  timept='start'; OUTPUT;
       date=&NAME_DATE_RIGHT.; timept='end';   OUTPUT;
       KEEP &NAME_ID. date timept &NAME_TYPE.;
RUN;


%* STEP 1b: SORT;
PROC SORT DATA=work.step1 OUT=work.step1b THREADS;
       BY &NAME_ID. date &NAME_TYPE. DESCENDING timept;
RUN;


%*STEP 2: ASSIGN TYPES;
DATA work.step2;
       LENGTH        &NAME_ID.            8.
                     &NAME_DATE_LEFT.     8.
                     &NAME_DATE_RIGHT.    8.
                     timept_left          $5.
                     &NAME_TYPE._left     8.
                     timept_right         $5.
                     &NAME_TYPE._right    8.;

       SET work.step1b (rename=(date= &NAME_DATE_RIGHT.
                                      &NAME_TYPE.=&NAME_TYPE._right
                                      timept=timept_right));
       BY &NAME_ID.;

       &NAME_DATE_LEFT.=LAG(&NAME_DATE_RIGHT.);
       &NAME_TYPE._left=LAG(&NAME_TYPE._right);
       timept_left=LAG(timept_right);
       IF NOT first.&NAME_ID.;

       FORMAT &NAME_DATE_LEFT. &NAME_DATE_RIGHT. mmddyy10.;
RUN;
```

```
DATA work.step2b;
       SET work.step2;
       BY &NAME_ID.;

       %* construct &NAME_TYPE. status array;
       ARRAY  &NAME_TYPE._status [&TYPE_MIN.:&TYPE_MAX.]
              &NAME_TYPE._status&TYPE_MIN.-&NAME_TYPE._status&TYPE_MAX.;
       IF (first.&NAME_ID.) THEN DO i=&TYPE_MIN. TO &TYPE_MAX.;
              &NAME_TYPE._status[i]=0;
       END;
            IF timept_left = 'start' THEN &NAME_TYPE._status[&NAME_TYPE._left]+1;
       ELSE IF timept_left = 'end'   THEN &NAME_TYPE._status[&NAME_TYPE._left]+(-1);

       %* assign &NAME_TYPE. to ranges;
       DO i=&TYPE_MAX. TO &TYPE_MIN. BY -1 UNTIL (&NAME_TYPE._status[i] > 0);
       END;
       IF i>=0 THEN &NAME_TYPE.=i;
       ELSE &NAME_TYPE.=&TYPE_GAP.;

       %* special handling for point dates;
       lag_&NAME_DATE_RIGHT. = LAG(&NAME_DATE_RIGHT.);
       lag_&NAME_TYPE.       = LAG(&NAME_TYPE.);
       IF ((NOT first.&NAME_ID.)
           AND (&NAME_DATE_LEFT.=&NAME_DATE_RIGHT.)
           AND (&NAME_DATE_LEFT.=lag_&NAME_DATE_RIGHT.)
           AND  &NAME_TYPE.<lag_&NAME_TYPE.) THEN DELETE;

       KEEP &NAME_ID. &NAME_DATE_LEFT. &NAME_DATE_RIGHT. &NAME_TYPE.;
RUN;


%* STEP 3 COLLAPSE;
PROC MEANS DATA=work.step2b MIN MAX NWAY NOPRINT;
       BY &NAME_ID. &NAME_TYPE. NOTSORTED;
       OUTPUT OUT=work.step3 (DROP=_TYPE_ _FREQ_)
              MIN(&NAME_DATE_LEFT.)=&NAME_DATE_LEFT.
              MAX(&NAME_DATE_RIGHT.)=&NAME_DATE_RIGHT.;
RUN;

%* STEP 4 MAKE DISJOINT;
DATA work.step4            (where =(&NAME_DATE_LEFT. <= &NAME_DATE_RIGHT.));
       SET work.step3      (rename=(&NAME_DATE_LEFT.  = &NAME_DATE_LEFT._old
                                    &NAME_DATE_RIGHT. = &NAME_DATE_RIGHT._old
                                    &NAME_TYPE.       = &NAME_TYPE._old));
       BY &NAME_ID.;

       &NAME_TYPE.1        = LAG2(&NAME_TYPE._old);
       date2_left          = LAG (&NAME_DATE_LEFT._old);
       date2_right         = LAG (&NAME_DATE_RIGHT._old);
       &NAME_TYPE.2        = LAG (&NAME_TYPE._old);
       date3_left          = &NAME_DATE_LEFT._old;
       date3_right         = &NAME_DATE_RIGHT._old;
       &NAME_TYPE.3        = &NAME_TYPE._old;
       lag_&NAME_ID.       = LAG(&NAME_ID.);
       lag_first_&NAME_ID. = LAG(first.&NAME_ID.);

       * initialize;
       IF first.&NAME_ID. THEN DO;
              &NAME_TYPE.1=.;
              date2_left=.;
              date2_right=.;
              &NAME_TYPE.2=.;
       END;
       IF (lag_&NAME_ID. = &NAME_ID.) AND (lag_first_&NAME_ID.=1) THEN &NAME_TYPE.1=.;

       * body: fit boundaries;
```

```
        IF (first.&NAME_ID.=0 AND last.&NAME_ID.=0) THEN DO;
               IF   (&NAME_TYPE.2 > &NAME_TYPE.1) OR (lag_first_&NAME_ID.=1) THEN
                     &NAME_DATE_LEFT.  = date2_left;
               ELSE  &NAME_DATE_LEFT.  = date2_left + 1;
               IF   (&NAME_TYPE.2 > &NAME_TYPE.3) THEN &NAME_DATE_RIGHT. = date2_right;
               ELSE  &NAME_DATE_RIGHT. = date2_right - 1;
               &NAME_TYPE. = &NAME_TYPE.2;
               OUTPUT;
        END;

        %*ELSE IF (first.&NAME_ID.=1 AND last.&NAME_ID.=0) THEN DO;
               %*do nothing;
        %*END;

        ELSE IF (first.&NAME_ID.=0 AND last.&NAME_ID.=1) THEN DO;
               IF   (&NAME_TYPE.2 > &NAME_TYPE.1) OR (lag_first_&NAME_ID.=1)
               THEN  &NAME_DATE_LEFT.  = date2_left;
               ELSE  &NAME_DATE_LEFT.  = date2_left + 1;
               IF   (&NAME_TYPE.2 > &NAME_TYPE.3) THEN &NAME_DATE_RIGHT. = date2_right;
               ELSE  &NAME_DATE_RIGHT. = date2_right - 1;
               &NAME_TYPE. = &NAME_TYPE.2;
               OUTPUT;

               IF    (&NAME_TYPE.3 > &NAME_TYPE.2) THEN &NAME_DATE_LEFT.  = date3_left;
               ELSE   &NAME_DATE_LEFT.  = date3_left + 1;
                      &NAME_DATE_RIGHT. = date3_right;
                      &NAME_TYPE.     = &NAME_TYPE.3;
                      OUTPUT;
        END;

        ELSE IF (first.&NAME_ID.=1 AND last.&NAME_ID.=1) THEN DO;
               &NAME_DATE_LEFT.  = date3_left;
               &NAME_DATE_RIGHT. = date3_right;
               &NAME_TYPE.       = &NAME_TYPE.3;
               OUTPUT;
        END;

        FORMAT date2_left date2_right
                  date3_left date3_right
                  &NAME_DATE_LEFT.  &NAME_DATE_RIGHT. mmddyy10.;

        KEEP &NAME_ID. &NAME_DATE_LEFT. &NAME_DATE_RIGHT. &NAME_TYPE.;
RUN;

%* STEP 4b COLLAPSE;
PROC MEANS DATA=work.step4 MIN MAX NWAY NOPRINT;
        BY &NAME_ID. &NAME_TYPE. NOTSORTED;
        OUTPUT OUT=&OUTDSN. (DROP=_TYPE_ _FREQ_)
               MIN(&NAME_DATE_LEFT.)=&NAME_DATE_LEFT.
               MAX(&NAME_DATE_RIGHT.)=&NAME_DATE_RIGHT.;
RUN;

%MEND Solution;
```