An Algorithm to Represent
a Collection of Sets as Intervals

We consider a collection of finite sets $T_1, T_2, \ldots, T_n$ and approach the problem of determining an ordering of the elements of $S = \bigcup T_j$ so that each set $T_j$ is an interval, i.e., a set of the form $\{x, a_j \leq x \leq b_j\}$. Not every collection of sets admits to a simultaneous representation of its members as intervals. An example is the collection whose members are $\{a, b\}, \{b, c\},$ and $\{a, c\}$. If an ordering of the elements of $S$ exists so that each set $T_j$ is an interval, the algorithm we give will produce such an ordering. It will terminate when the discovery is made that such an ordering does not exist. We give an example to illustrate the strategy of the algorithm.

Let $T_1 = \{a, b, d\}, T_2 = \{b, c, d\},$ and $T_3 = \{b, c, e\}$. We seek to produce an ordering of the letters $\{a, b, c, d, e\} = S$ so that each of $T_1$, $T_2$, and $T_3$ is an interval in that ordering. We start with the observation that if each of $T_1$ and $T_2$ is to be an interval then $T_1 \cap T_2$ is an interval and separates the elements of $T_1 - T_2$ from the elements of $T_2 - T_1$.
We write these conditions symbolically as
\[
\{a\} < \{b, d\} < \{c\}
\]
That is, in any ordering a precedes each of b and d and each of these precedes c. Conversely, each of \(T_1\) and \(T_2\) is an interval in any ordering of \(\{a, b, c, d\}\) which respects these conditions. If \(T_3\) is also to be an interval in an ordering in which each of \(T_1\) and \(T_2\) is an interval, then as d is not a member of \(T_3\), d cannot separate b and c which are in \(T_3\). Hence, we have the conditions
\[
\{a\} < \{d\} < \{b\} < \{c\} < \{e\}
\]
We have tacked \(\{e\}\) onto the right end because \(T_3\) covers the set \(\{c\}\) on the right in the ordering (*) . At this point the ordering of \(S = \{a, b, c, d, e\}\) is completely determined by these relations. Reversal of the \(\langle\) sign produces another but equivalent ordering. An additional set which contains elements of \(S\) must contain a single interval in this ordering, if that set is to admit a simultaneous representation with \(T_1, T_2, T_3\) as intervals.

This strategy can be extended to any number of sets \(T_1, T_2, \ldots, T_n\). As in the above example, the algorithm maintains a list
\[
S_1 < S_2 < \ldots < S_k
\]
of subsets of \(S = \bigcup_j T_j\) which contains implicitly all orderings in which a finite number of sets \(T_1, T_2, \ldots, T_k\) are intervals.
in the following precise sense. An arrangement of the elements
of \( \bigcup_{j} S_j \) is an ordering in which each of the sets \( T_1, T_2, \ldots, T_k \)
is an interval if and only if each element of \( S_{i-1} \) precedes
every element of \( S_i \). The list \( S_1, S_2, \ldots, S_k \) is constructed
in the following manner. First, it is initialized as \( T_1 \).
Then each set \( T_j \) is considered in turn. The conditions
imposed by the \( j+1 \)st set are included in the list in the
following manner. Note that if \( T_{j+1} \) contains \( \bigcup_{i} S_i \) then it
is not possible to make a choice of the location
of the elements of \( T_{j+1} - \bigcup_{i} S_i \) so that the relation of the
list to the orderings of the elements of \( \bigcup_{i} T_i \) in which
each set is an interval is preserved. In this case \( T_{j+1} \)
is declared to be exceptional and is put aside until the
remaining sets have been considered.

Now suppose that \( T_{j+1} \) does not contain one of the
elements of \( S_i \). The elements in \( T_{j+1} - \bigcup_{i} S_i \), if this set
is nonempty, are attached as a single set to the left of
\( S_1 \), if \( T_{j+1} \supset S_i \) or to the right of \( S_k \), if \( T_{j+1} \supset S_i \). If
both of these conditions are satisfied, then the next step
in the process will determine that no order of \( S \) exists
and the algorithm will terminate no matter what choice is
made. If neither condition is satisfied, the additional
elements are attached to an end which \( T_{j+1} \) intersects
nontrivially. If \( T_{j+1} \) intersects nontrivially both or
neither end, then an arbitrary choice is made and the next step detects that no order is possible. The next step depends on the observation that if $T_{j+1}$ is to be an interval, then the indices of the sets $S_i$ which $T_{j+1}$ intersects nontrivially must form an interval and $T_{j+1}$ must contain each of these sets $S_i$ except possibly the sets on either extreme of the interval.

If this condition is satisfied, we consider first the case that $T_{j+1}$ is a proper subset of some set $S_i$. In this case, there is no way to alter the list $S_1, S_2, \ldots, S_k$ so that the list maintains its relationship to all orderings of the sets $T_j$ which have been considered prior to this step and were not exceptional. We declare $T_{j+1}$ to be exceptional and put it aside until the remaining sets have been considered. It is possible that one of the succeeding sets will separate the elements of $S_i$ in such a way that $T_{j+1}$ is no longer a subset of any set $S_i$. On the other hand, if the minimum and maximum indices of the sets $S_i$ which $T_{j+1}$ intersects nontrivially are different, then changes are made in the list $S_1, S_2, \ldots, S_k$. Let $\min$ and $\max$ denote these indices respectively. If $T_{j+1} \cap S_{\min}$ is a proper subset of $S_{\min}$, then $S_{\min} - T_{j+1}$ and $T_{j+1} \cap S_{\min}$ replace $S_{\min}$ in the list in this order. Similarly, if $T_{j+1} \cap S_{\max}$ is a proper subset of $S_{\max}$, then $T_{j+1} \cap S_{\max}$ and $S_{\max} - T_{j+1}$ replace $S_{\max}$ in the list in this order. The relationship of the list to all orderings of the elements of the first
j+1 sets which are not exceptional is preserved. After all
of the sets $T_j$ are considered on the first pass, each of the
exceptional sets is reconsidered. The process which we
have described above is repeated. If any exceptional sets
remain after this pass, another complete iteration of the
procedure is performed. Iterations of the exceptional
sets are made until either no exceptional sets remain,
a complete pass results in no exceptional sets $T_j$ being
successfully processed, or until an error condition occurs.
If the latter occurs, the algorithm terminates.

If the iterative process terminates without an
error being detected and with exceptional sets remaining,
then recursive invocations of the algorithm are made to
order the elements contained in the exceptional sets.
More precisely, the set union $= \bigcup S_1$ is ordered by ordering
the elements of each of the sets $S_j$. If $S_j$ contains
any exceptional sets $T_i^j$, $T_j^j$, ..., $T_k^j$ the algorithm
we have described above is used recursively to
sequence the elements of the union of these sets.
The elements of the set $S_j - \bigcup_{i=1}^{k} T_i^j$ are sequenced arbitrarily.
The sequences of the sets $S_j$ are then concatenated in the
order of their indices. The exceptional sets which contain
union $T_1^g, T_2^g, ..., T_k^g$ are used to produce an ordering of
the remaining elements by applying the algorithm to the
sets $T_1^F$-union, $T_2^F$-union, $\ldots$, $T_k^F$-union. This order is concatenated to the order of union produced above.

If any of these recursive invocations of the ordering algorithm discovers that an order does not exist, then an error flag is set which is propagated to the initial invocation of the algorithm and the process terminates.

We do not explore the calculation of all partial orderings of $S$ although a straightforward modification of the code we give below will produce all such orderings.

We now give code in SETL for this process. The decisions which may be vaguely described above are precisely specified in this code. For the convenience of the reader, we detail the function of the principal routine and its prominent data structures:

arrangelts(•) -- argument is a collection of sets
result is a tuple which contains an ordering of $T_j$, if one exists, the null tuple otherwise.

failflag - global failure flag which is set upon discovering that no order exists

detailsets - tuple of sets which contains the sequence $S_1, S_2, \ldots, S_k$

/* set failflag to 1 prior to first invocation */
definef arrangelts(tset);
/* failflag is global; tset contains the sets to be made into intervals */
s1 from tset; listset={s1}; union = nl;
exceptsets=nl; insert= t;
/* exceptsets contains exceptional sets found on current pass */
(while insert doing tset = exceptsets; exceptsets = nil;
(\forall x \in tset)

flow

exceptg?

inexcept

extraelts?

onsmallend?

calcints+

onsmall+

onbig+

calcints calcints

nexcepts?

conflict?

inexcept

(failflag=t; makinsert

return nil;)

exceptg := x \in tset or x*union eq nil;
inexcept : x \in exceptsets; continue \forall x;
extraelts := x - union is extraelts ne nil;
onsmallend := x*listsets(1) ne nil and
n x*listsets(#listsets) eq listsets(#listsets);
onsmall := listsets =< extraelts > + listsets; union = union + extraelts;
onbig := listsets = listsets +< extraelts > ;union = union + extraelts;
calcint: indicescov =\{ lset,lset \in listsets \lset \in x eq lset \};
indicecessub =\{ lset,lset \in listsets \lset \in x ne nil \};

minm1 =([\min; y \in indicescov]y)-1;
maxp1 = ([\max; y \in indicecessub]y)+1;
nexcepts := indicescov eq nil and #indicecessub eq 1;

/* x is a subset of some member of listsets
and is therefore exceptional if above is t */

conflict := \# (interval (indicescov) and
indicecessub \lt (indicescov +\{ minm1,maxp1\}));

/* if true then raise the error flag as no
order exists */
makinsert:
  if (minm1 ∈ indicesub)
    then listsets = listsets(1:minm1-1) +
        <listsets(minm1)-x, listsets(minm1)*x> +
        listsets(minm1+1:);
  end if;

  if (maxp1 ∈ indicesub)
    then listsets = listsets(1:maxp1-1) +
        <listsets(maxp1)*x, listsets(maxp1)-x> +
        listsets(maxp1+1:);
  end if;

end flow;
end ∀x;
end while;

/* factor the exceptional sets which are larger than union */
grossets = {x-union, x ∈ exceptsets | x ≥ union};
order = arrangelts(grossets);
if (failflag) then return null ;;
(∀ x ∈ listsets)
  exceptx = {y, y ∈ exceptsets | y < x};
  order = order + maktyp( x-exceptx ) + arrangelts(exceptx);
  if (failflag) then return null ;;
end ∀ x;

/* if fall out, have successfully ordered the sets */
return order;
end arrangelts;
define maktup(set);
/* makes tuples out of elements of set */
if ( set eq nil ) then return null ;
return [+: x ∈ set] <x> ;
end maktup;

define interval(setofintegers);
/* determines if input set is an interval */
if (#setofintegers le 1 ) then return t ;
minset = [min, i ∈ setofintegers] i ;
maxset = [max, i ∈ setofintegers] i ;
return (setofintegers eq {i, minset ≤ i ≤ maxset} ) ;
end interval;