SETL Newsletter 104

Gray Jennings
March 28, 1973

# An Algorithm to Represent a Collection of Sets as a Direct Product of Intervals on the Line

In SETL Newsletter 94 we gave an algorithm to represent each member of a collection of finite sets $T_1, T_2, \ldots, T_n$ as an interval on the line, i.e., a set of the form $\{x,\ a_j \leq x \leq b_j\}$. That algorithm determines, if possible, an ordering of the elements of the union $\mathcal{T} = \bigcup_{j=1}^{n} T_j$ so that each set $T_i$, $i=1,\ldots,n$, is an interval.

In this newsletter we give an algorithm which determines the minimal number of disjoint sets $S_1,\ldots,S_k$ so that for each index s, all of the sets

$$T_1 \cap S_r,\ T_2 \cap S_r,\ \ldots,\ T_k \cap S_r$$

can be represented as an interval when the elements of $S_r$ are arranged in some order. We require that

$$S_1 + S_2 + \ldots + S_k = \bigcup_j T_j$$

and thus each set $T_j$ can be represented as the direct product of disjoint intervals.

$$T_j = \sum_{j=1}^{k}{}^{+} (T_j \cap S_k)$$

If each interval is located physically on a row of the lattice of points in the plane with integer coordinates, some partitions may allow the representation of each set $T_j$ as a rectangle in the two dimensional lattice.

The algorithm we give first determines all maximal subsets S of $\mathcal{T}$ such that the sets

$$S \cap T_1,\ S \cap T_2,\ \ldots,\ S \cap T_k$$

can be represented as intervals in the points of S. A modification of the algorithm in Newsletter 94 is given for this part. The next step is to pick the minimal number of disjoint sets, each of which is a subset of a maximal set whose union is all of $\mathcal{T}$. We do not specify an algorithm for this part. We invite the reader to choose one from the literature.

We now indicate how to modify the algorithm in Newsletter 94. Suppose that k-1 sets have been considered,by the process specified in Newsletter 94, without determining that simultaneous representation of these k-1 sets as interval is impossible. At this stage, we have calculated a sequence of sets

$$R_1,\ R_2,\ \ldots,\ R_\ell$$

so that each of $T_1, T_2, \ldots, T_{k-1}$ is an interval in any ordering of the elements of $\bigcup_i R_i$ in which each element of $R_j$ precedes each element of $R_{j+1}$. We consider the remaining sets $T_k, T_{k+1}, \ldots, T_n$, the class of exceptional sets, and the union $\mathcal{T} = \bigcup T_j$ as the "state variables" of the calculation. If $\bigcup_i R_i$ is a proper subset of $T_k$ or $T_k$ is a proper subset of some $R_i$, then $T_k$ is added to the collection of exceptional sets and $T_{k+1}$ is then considered. If $T_k$ is not exceptional, $T_k$ is used to refine $R_1, R_2, \ldots, R_\ell$ in a unique way so that all orderings of the elements of $T_1, \ldots, T_k$ are contained in the sequence

$$R_1', R_2', \ldots, R_\ell'.$$

Such a refinement exists if the indices of the sets $R_i$, which $T_k$ intersects nontrivially, form an interval. That is, $T_k$ intersects all $R_i$ when $i \in [\min,\max]$ and $T_k$ covers all $R_i$ for $i \in (\min,\max)$. Then $R_{\min}$ is replaced by $R_{\min}-T_k$, $R_{\min} \cap T_k$, and $R_{\max}$ is replaced by $R_{\max} \cap T_k$, $R_{\max} - T_k$. The algorithm in the case that $T_k$ contains elements not in $\bigcup_i R_i$ is explained in Newsletter 94.

If $T_k$ is not exceptional and the sequence cannot be refined, each set $R_{\min}, R_{\min+1}, \ldots, R_{\max}$ is split by $T_k$ into at most two sets, $R_i \cap T_k$ and $T_k - R_i$. For at least one i, $T_k - R_i$ is not empty. We construct a family of maximal sets, $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_\ell$ so that the restrictions to $\mathcal{T}_i$, $i=1,\ldots\ell$, of $T_1, T_2, \ldots, T_k$ can be represented as intervals. We then constitute $\ell$ separate problems. A problem is determined by the state variables, $\mathcal{T}_i$, the former exceptional sets restricted to $\mathcal{T}_i$, the restrictions to $\mathcal{T}_i$ of the sets $T_1, T_2, \ldots, T_k$, and a portion $R_1^i, R_2^i, \ldots, R_\ell^i$

which is calculated from the restriction of $R_1, R_2, \ldots, R_\ell$ to $T_i$. The algorithm is then used separately on each of these smaller problems. It is possible that each problem will be split again by the repeated applications of the algorithm.

The algorithm produces a finite collection of sets $S_1, S_2, \ldots, S_r$ together with a sequence for each which contains implicitly all orderings for which each of

$$T_1 \cup S_j,\ T_2 \cup S_j,\ \ldots,\ T_m \cap S_j$$

is an interval. A single set $\bigcup T_j$ is produced, if an ordering of $\bigcup_j T_j$ exists, so that each of $T_1, T_2, \ldots, T_k$ is an interval.

We now describe the process for determining the sets $T_1, T_2, \ldots, T_\ell$, when a single refinement of $R_1, R_2, \ldots, R_\ell$ does not exist. If there are members $R^* = T_k - \bigcup_i R_i$ then the process is performed first on the partition $R^*, R_1, \ldots, R_\ell$ and then on the partition $R_1, R_2, \ldots, R_\ell, R^*$. $T_k$ divides each $R_i$ into $R_i \cap T_k = R_i^{in}$ and $T_k - R_i = R_i^{out}$. We let min and max denote the minimum and maximum indices for which $R_i^{out} \neq \emptyset$. Then the sequence $R_i$ is refined to

$$R_1^{out},\ R_2^{out},\ \ldots, (R_{min}^{in}, R_{min}^{out}), \ldots,\ (R_i^{in}, R_i^{out}), \ldots,\ R_\ell^{out}$$

If a single refinement were possible there would be at most two sets $R_{min}$ and $R_{max}$ for which each of $R_{\cdot}^{in}$ and $R_{\cdot}^{out}$ were not empty.

For each pair of indices $i < j$, for which $R_i^{out}$ and $R_j^{out}$ are not empty, a set $T_{i,j}$ is formed from the union of

$$R_1^{out}, \ldots, R_i^{out}, R_i^{in}, R_{i+1}^{in}, \ldots, R_j^{in}, R_j^{out}, R_{j+1}^{out}, \ldots, R_\ell^{out}$$

The partition $R_1^{i,j}, R_2^{i,j}, \ldots, R_\ell^{i,j}$ can be calculated from $R_1, R_2, \ldots, R_\ell$ by forming the intersection of each $R_i$ with $T_{i,j}$. When an intersection is empty, the neighbors are replaced by a single set consisting of the union of these two sets.

We detail the functions of the principal routines and data structures:

workpile: set contains tuples of state variables of the form
<partition, universalset, remainset, exceptsets>

partition: the sequence $R_1, R_2, \ldots, R_k$ represented as a tuple

universalset: $\cup R_j$

remainset: $\{T_k, T_{k+1}, \ldots, T_n\}$, sets not yet considered

exceptsets: the exceptional sets

addset: argument is an element of *workpile*
an element of *remainset* is applied to *partition*

We start with $T_1$, $T_2$, ..., $T_n$ in *tset*.

```
    workpile = nl;  maxsets = nl;
    set from tset;
<<set>, set, tset, nℓ> in workpile;
(while workpile ne nl)

tuple from workpile;
if tuple(3) eq nl            /* remainset eq nl */
  then /* add exceptional sets */
    if tuple(4) is tuple(3) eq nl
      then tuple(1:2) in maxsets;
             continue while;
      else tuple(4) = nl;
    endif;
endif;
    addset (tuple);
end while;

/* maxsets contains <partition,univset> such that T1,T2,...,Tn
relativized to univsets are intervals - use any algorithm for
extracting a minimal collection of univsets */

define addset (tuple);
<listsets, union, remainset, exceptsets> = tuple;
x from remainset;
  savelistsets = listsets;
```

flow

```
        exceptg?

   inexcept+                  extraelts?
   inworkpile      onsmallend?               calcints+

             onsmall+       onbig+

             calcints       calcints

                                            nexcepts?

                                  conflict?         inexcept+
                                                    inworkpile
                             manysets     makinsert+
                                          inworkpile
```

```
exceptg:= x ge union or x*union eq nl;

inexcept:= x in exceptsets; continue ∀ x;

extraelts:= x - union is xtraelts ne nl;

onsmallend:= x*listsets(1) ne nl and
              n x*listsets(#listsets) eq listsets(#listsets);

onsmall:= listsets=<xtraelts>+ listsets; union = union+ xtraelts ;

onbig:= listsets = listsets +<xtraelts>; union = union + xtraelts;

calcint:= indicescov = {lset,lset ε listsets | lset*x eq lset};

          indicessub = {lset,lset ε listsets | lset*x ne nl};

          minml = ([min:y ε indicescov]y])-1;

          maxpl = ([max:y ε indicessub]y) + 1;

nexcepts:= indicescov eq nl and #indicessub eq 1;

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

conflict:= n (interval (indicescov) and
             indicessub lt (indicescov + {minml, maxpl}));

    /* if true then create more maximal sets */
```

```
makinsert:
  if (minml ε indicessub)
    then listsets = listsets(1:minml-1) +
        <listsets(minml)=x,listsets(minml)*x> +
         listsets(minml+1:);
  endif;

  if (maxpl ε indicessub)
    then listsets = listsets(1:maxpl-1) +
        <listsets(maxpl)*x, listsets(maxpl)-x> +
         listsets(maxpl+1:);
  endif;
inworkpile:
<listsets, union, remainset, exceptsets> in workpile; return;


manysets:
  if (xtraelts ne nl)
    then makstatevar(x,<xtraelts> + savelistsets);
      savelistsets = savelistsets + <xtraelts>;
  endif;
  makstatevar(x,savelistsets);
  return;
  endflow;
end addset;


definef makstatevar(x,listsets);
intuple = [+ : 1 ≤ j ≤ #listsets] <listsets * x> ;
outtuple = [+ : 1 ≤ j ≤ #listsets] <x-listsets> ;

(min ≤ ∀i ≤ max)
  (i ≤ ∀j ≤ max | jlt max imp (outtuple(j) ne nl)
  newstatevar(i,j) in workpile
  end ∀j;
end ∀i;
return;
```

We now give code for the function

```
definef newstatevar(i,j);
/* intuple, outtuple, exceptsets, remainsets are global */
partition = outtuple(1:i) + intuple(i:j) + outtuple(j:);
  univset = [+ : 1<j<#partition] partition (j);
/* the new maximal set */
/* suppresses nℓ in partition */
  first = if (1<∃j<#partition | partition(j) ne nℓ)
          then j else Ω;
  newseq =  partition(first) ;
  j = first + 1;

(while j le #partition doing j = j + 1;)
  if partition(j) eq nℓ
    then newseq(#newseq) = newseq(#newseq) + partition(j+1);
      j = j + 1;
    else newseq = newseq + <partition(j)>;
  endif;
endwhile;


newremain = {x * univset, x ε remainset};

newexcept = {x * univset, x ε exceptset};

return <newseq, univset, newremain, newexcept>;

end newstatevar;


definef interval(setofintegers);

/* determines if input set is an interval */

if (#setofintegers le 1) then return t;;

minset = [min, i ε setofintegers] i;

maxset = [max, i ε setofintegers] i;

return (setofintegers eq {i, minset<i<maxset});

end interval;
```