

CSL *COORDINATED SCIENCE LABORATORY*

APPLIED COMPUTATION THEORY GROUP

**BOOLEAN EXPRESSIONS
OF RECTILINEAR POLYGONS
WITH VLSI APPLICATIONS**

MICHELE PRACCHI

APPROVED FOR PUBLIC RELEASE. DISTRIBUTION UNLIMITED.

REPORT R-978

UILU-ENG 82-2244

UNIVERSITY OF ILLINOIS - URBANA, ILLINOIS

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) BOOLEAN EXPRESSIONS OF RECTILINEAR POLYGONS WITH VLSI APPLICATIONS		5. TYPE OF REPORT & PERIOD COVERED Technical Report
7. AUTHOR(s) Michele Pracchi		6. PERFORMING ORG. REPORT NUMBER R-978 UILU-ENG 82-2244 (ACT-36)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Coordinated Science Laboratory University of Illinois at Urbana-Champaign Urbana, IL 61801		8. CONTRACT OR GRANT NUMBER(s) NSF ECS 81-06939; N00014-79-C-0424
11. CONTROLLING OFFICE NAME AND ADDRESS Joint Services Electronics Program		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		12. REPORT DATE November 1982
		13. NUMBER OF PAGES 50
		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) VLSI design rule checking, VLSI masks, boolean operations on masks, plane sweep techniques		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This work is devoted to the analysis of some aspects of the problem of mask verification in the design of integrated circuits. Boolean operations on masks - union, intersection, complement - are the fundamental steps in the development of these verification strategies. We first consider the problem of union and intersection of two masks in a rectilinear environment. The input masks are described by means of polygonal circuits. The output mask is the result of the boolean operation and it is always described by means of polygonal circuits. We present a plane sweep technique algorithm that solves this problem in time		

$O(N \log N + k)$ and using memory $O(N + k)$, where N is the total number of edges that define the polygonal circuits of the two input masks and k is the number of intersection points.

The natural generalization of the problem is the construction of the plane regions that verify a boolean expression, whose variables are a set of masks. We first present an algorithm that has $O(Nh \log N + k')$ running time and uses memory $O(N \log h + k'h + k')$, where N is the total number of edges that define the polygonal circuits of all the masks, h the size of the boolean expression and k' is the number of intersection points that are vertices of the polygonal circuits of the output mask.

BOOLEAN EXPRESSIONS OF RECTILINEAR POLYGONS WITH VLSI APPLICATIONS

Michele Pracchi

ABSTRACT

This work is devoted to the analysis of some aspects of the problem of mask verification in the design of integrated circuits. Boolean operations on masks - union, intersection, complement - are the fundamental steps in the development of these verification strategies. We first consider the problem of union and intersection of two masks in a rectilinear environment. The input masks are described by means of polygonal circuits. The output mask is the result of the boolean operation and it is always described by means of polygonal circuits. We present a plane sweep technique algorithm that solves this problem in time $O(N \log N + k)$ and using memory $O(N + k)$, where N is the total number of edges that define the polygonal circuits of the two input masks and k is the number of intersection points.

The natural generalization of the problem is the construction of the plane regions that verify a boolean expression, whose variables are a set of masks. We first present an algorithm that has $O(N \log N + k')$ running time and uses memory $O(N \log h + k' h + k')$, where N is the total number of edges that define the polygonal circuits of all the masks, h the size of the boolean expression and k' is the number of intersection points that are vertices of the polygonal circuits of the output mask.

1. INTRODUCTION

The development of computer aids for the design of integrated circuits is the object of increasing interest, [1],[2],[3],[4],[5], as the problem of the efficient design of Very Large Scale Integrated (VLSI) circuits cannot be solved without the introduction of new and more sophisticated software and hardware tools.

One of the aspects that have to be investigated is mask verification. Masks are used in the production of integrated circuits and, when they are manually or semiautomatically drawn, a check is essential for the verification of their compliance with the design rules.

This work is devoted to the analysis of some aspects of the problem of mask verification. Boolean operations on masks - union, intersection, complement - are the fundamental steps in the development of these verification strategies.

We first consider the problem of union and intersection of two masks in a rectilinear environment. The input masks are described by means of polygonal circuits. The output mask is the result of the boolean operation and it is always described by means of polygonal circuits. We present a plane sweep technique algorithm that solves this problem in time $O(N\log N+k)$ and using memory $O(N+k)$, where N is the total number of edges that define the polygonal circuits of the two input masks and k is the number of points in which the polygonal circuits of the two input masks intersect. Since N is the size of the input and the k intersection points are endpoints of the polygonal circuits of the output mask, their determination and report are essential for the solution of the stated problem. This algorithm has an

optimal worst case running time. The recently proposed algorithm by Yap [1] solves the same problem using time $O((N+k)\log N)$ and memory $O(N+k)$. We note that the time performance is not optimal: the number of intersection points k is multiplied by $\log N$. Chapter 2 is devoted to the description of this topic.

The report of the plane regions that verify a boolean expression, whose variables are a set of masks, is the problem solved in Chapter 3. We remain in a rectilinear environment and the input masks are described by means of polygonal circuits. We note now that the boolean expression is an additional input to our problem. We will account for this when evaluating the performance of the algorithms by introducing a new parameter h , defined as the number of variables and connectives in the expression. We first present an algorithm that has $O(Nh\log N+k)$ running time and uses memory $O((N+k)\log h)$, where N is the total number of edges that define the polygonal circuits of all the masks, h the size of the boolean expression, and k the total number of intersection points in which the polygonal circuits of the input masks intersect. N and h are the parameters that describe the size of the input. However, not all of the k intersection points are necessary for the description of the output mask. Let us call k' the number of intersection points that are endpoints of the polygonal circuits of the output mask: k' is less than or equal to k . The second algorithm presented in Chapter 2 has a time and space performance depending on k' instead of k . It runs in time $O(Nh\log N+k')$ and uses space $O(N\log h+k''h+k')$, where k'' is the smaller of the numbers of the horizontal and vertical edges that define the k' intersection points. To the best of our knowledge, this problem has not hitherto been considered in the literature.

2. ALGORITHMS FOR CONSTRUCTING THE INTERSECTION AND THE UNION OF MASKS

The first algorithm we shall present solves the following problem: two masks A and B are given. Every mask is a set of polygonal circuits subject to the following rules:

Rule a: every edge of the polygons is either vertical or horizontal (orthogonal geometry);

Rule b: the interiors of the polygons of a mask do not overlap;

Rule c: we assume a counterclockwise orientation for the polygons, i.e., each polygon lies to the left side of any of its boundary edges.

We want to have a description of the mask that results from the application of the boolean connective AND or of the connective OR to the two masks A and B. This mask will always be described by means of polygonal circuits subject to the rules previously stated. For the AND connective it is possible that the output is the empty polygon. We will first describe the (A AND B) boolean operation.

2.1 Mask Intersection

The construction of the (A AND B) output masks consists of finding the endpoints of the edges of the polygons of that mask and of assigning to each of these edges its orientation. We will first consider the problem of determining the endpoints of the edges.

The endpoints belong to two disjoint classes:

1. The class of inclusion endpoints: these are the endpoints of the edges of the polygons of mask A that belong to the interior of a polygon of mask B and the endpoints of the edges of the polygons of B that belong to the interior of a polygon of A.
2. The class of intersection endpoints: these endpoints result from the intersections of edges of the two masks.

We shall consider separately the computation of these two classes.

1. Inclusion endpoints. The inclusion in B of an endpoint P of A can be determined if we consider the configuration of mask B at the x-value of P. Let $\text{ORIENTATION}(P,B)$ be the orientation of the horizontal edge of mask B whose intercept with a vertical line through P is immediately above P.

We have the following theorem:

Theorem: An endpoint P of mask A is enclosed in a polygon of mask B if $\text{ORIENTATION}(P,B)$ is right-to-left.

Proof: (by contradiction) Let us consider a vertical line v through P. Line v intersects h , the horizontal edge of B immediately above P, in point Q (see Figure 2.1). Edge h is oriented right-to-left, so that the region immediately below h is a region of mask B. Assuming, for a contradiction, that P does not belong to a region of mask B, the Jordan Curve Theorem requires v to intersect an edge of mask B whose ordinate is between the one of P and the one of Q. That is against the hypothesis. Thus, P lies in a region of B. \square

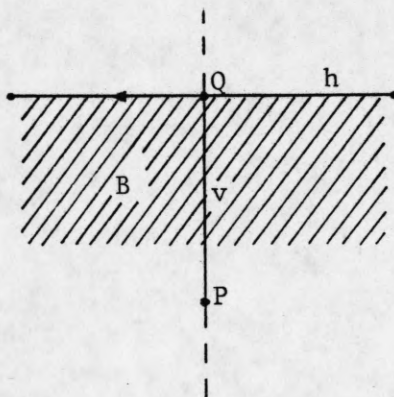


Figure 2.1. Situation where P is enclosed in a polygon of B.

We will use this result in the algorithm that will construct the mask (A AND B).

2. Intersection endpoints. The problem of finding all intersection pairs in a set of horizontal and vertical line segments has been solved by Bentley-Ottmann [6] in optimal time and space. Thus, we will use their technique to determine and report the intersection endpoints.

The detection and report of the endpoints of the output mask are just the first steps in the construction of the polygonal circuits of that mask. Every endpoint defines an incoming and an outgoing edge of the output mask and the concatenations of these points and edges define the polygonal circuits bounding the planar regions that satisfy the boolean intersection operation. In the case of inclusion endpoints, the incoming and outgoing edges of the output mask are already defined, as there is just one incoming and one outgoing edge. The intersection endpoints require more care. In fact we have two incoming and two outgoing edges and we need to know which incoming and which outgoing edge will define the polygonal circuits of the output mask. An intersection endpoint has an incoming and an outgoing edge of the output mask. The selection of the appropriate edges depends on the boolean connective we consider. For every possible intersection configuration we can select the appropriate edges. That is shown in Figure 2.2: any intersection endpoint is determined by an edge of mask A and an edge of mask B. These edges have an orientation and the regions belonging to either a mask are specified by these orientations, according to the specification of Rule c given at the beginning of this chapter. From that information we can determine the incoming and outgoing

edges of the output masks. Figure 2.2 shows just some of the intersection endpoint configurations and the corresponding endpoint of the output mask for boolean intersection. The appendix is devoted to the complete description of all configurations for union and for intersection.

The determination of the endpoints of the output mask and of their incoming and outgoing edges is the information we need. In fact we can concatenate these endpoints and have a complete description of the intersection regions.

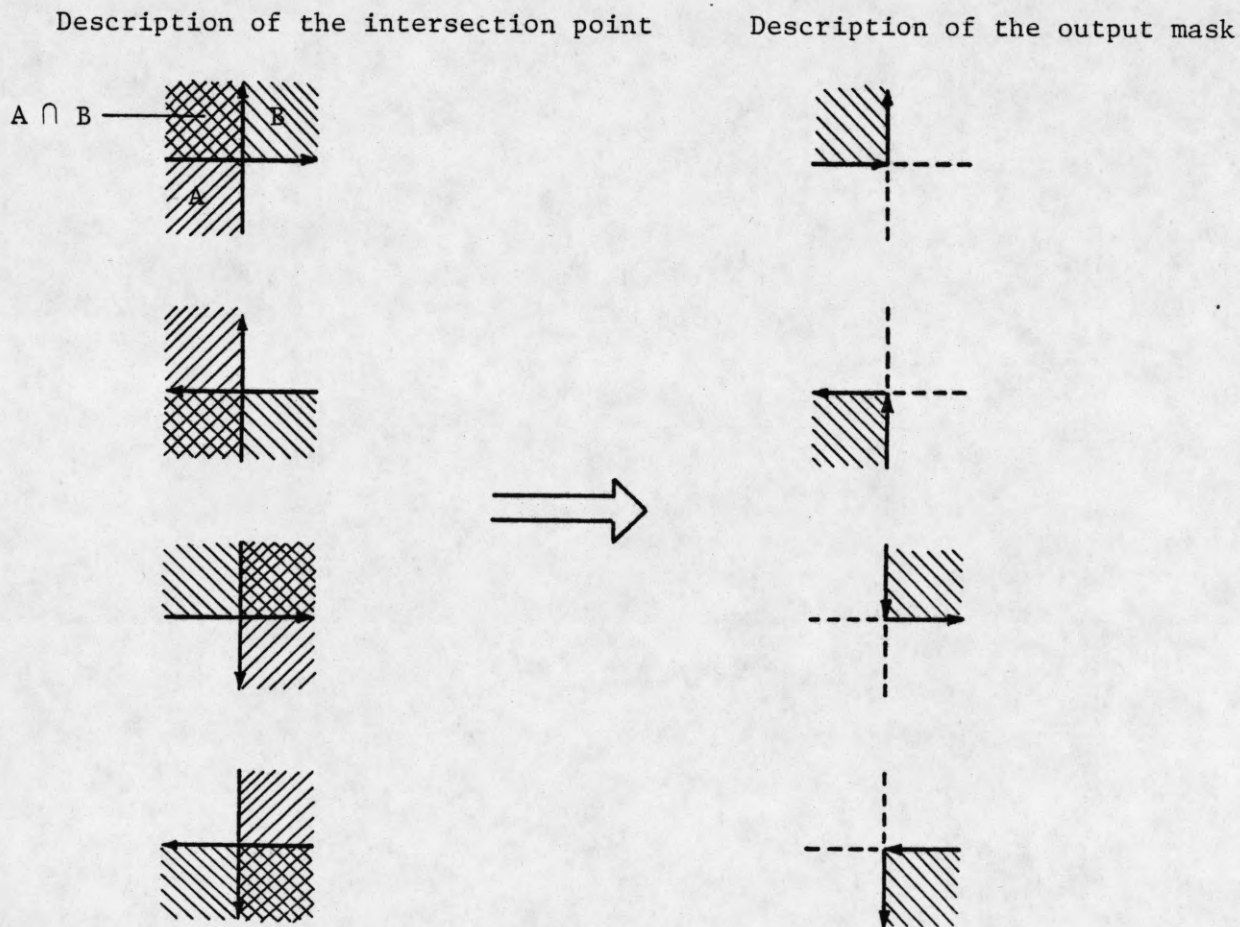




Figure 2.2. Detection of the edges of the intersection mask. ("Always turn left"). Mask A is illustrated , mask B .

Section 2.2 is devoted to the presentation of an algorithm, based on the results of this section, for the construction of the intersection mask.

2.2 Algorithm MASK INTERSECTION

We present a plane sweep algorithm that, by sweeping the plane unidirectionally, determines the boundaries of the polygons of the output mask. The input data consists of two cyclic lists of points: $V_i = (x_i, y_i)$, $i = 1, \dots, N_A$, and $W_j = (x_j, y_j)$, $j = 1, \dots, N_B$; we also let $N = N_A + N_B$. The V_i 's are the endpoints of the edges of the polygons of mask A, the W_j 's the ones of mask B. The record pertaining to an endpoint has attached the information concerning its incoming and outgoing edges. The desired result is a list of all regions that verify the boolean intersection operation, where each region is given by a cyclic list of its vertices. The orientation of the boundaries is in accordance with Rule c.

The dynamic aspects of the algorithm are described by means of the cross section, which is a vertical line in the plane at abscissa x , along with the sequence of the polygons' edges it cuts, ordered from bottom to top.

Data structures maintained by the algorithm. Algorithm MASK INTERSECTION uses four data structures. Two of these, the x-structure and the y-structure, are common to all plane-sweep algorithms. As the line that sweeps the plane advances in the direction of the x-axis, the x-structure represents a queue of tasks to be accomplished. The y-structure represents the state of the current cross section. The third data-structure, the i-structure, is specific to our needs. It represents the intersection endpoints. The fourth data-structure, the s-structure, represents the polygonal circuits of mask (A AND B). We shall next illustrate these data structures in detail.

The x-structure X contains the x-coordinates of the vertical edges of the input polygons. Every entry describes one of the configuration types shown in Figure 2.3. Every edge of X has a pointer to its record in the s-structure. The x-structure can be implemented as a sorted linear list. Sorting the elements of X and storing them in increasing order takes time $O(N \log N)$.

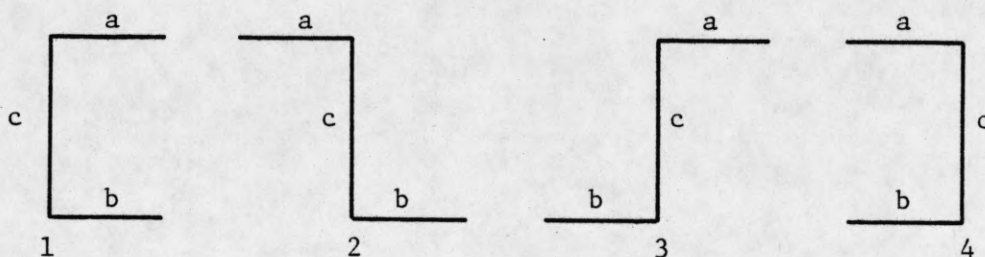


Figure 2.3. The four possible configurations for an assigned value of x .

The y-structure Y contains the description of the current cross section. Y has an entry for each horizontal edge intersected by the sweep line. These entries are sorted according to increasing y-coordinates. Y is what is conventionally called a dictionary and must support operations INSERT, DELETE, PREDECESSOR, SUCCESSOR within time bound $O(\log a)$ when it contains a entries. This result can be achieved if we use a height-balanced tree.

The i-structure I provides the information about the intersection points and their incoming and outgoing edges. It is initialized to be empty and terminates empty. Data structure I is a linear list. Every record of the list represents an intersection point P and has a pointer IEDGE(P) to the

record of the s-structure that represents the outgoing edge. The i-structure is shown in Figure 2.4. It supports the following operations: $I \leftarrow P$, which adds record P to the tail of list I; $P \leftarrow I$, which extracts record P from the head of list I; $IDDELETE(P)$ deletes record P from list I, anywhere in the list.

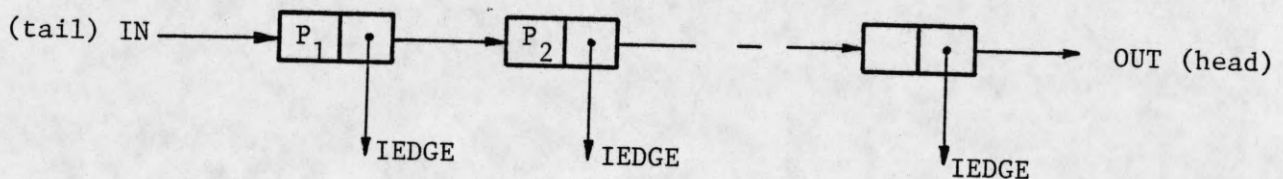


Figure 2.4. The i-structure I.

The s-structure S provides the information about the polygonal circuits of masks A and B. Referring to Figure 2.5, a cyclic list (primary structure) is used to represent each polygonal circuit. Each record of the list represents an edge of the polygonal circuit, points to the following edge, $SSUCCESSOR(e)$, with respect to the orientation of the polygon, and to a list L (secondary structure) of pointers to the records of I representing the intersection points lying on that edge. L is a queue if the orientation of the edge is left-to-right or bottom-to-top, otherwise it is a stack. The s-structure is illustrated in Figure 2.5. S supports the following operations: $L_{e_n} \leftarrow P$, $P \leftarrow L_{e_n}$, and $SSUCCESSOR(e)$. $L_{e_n} \leftarrow P$ adds a record

to the list L_{e_n} of edge e_n in a fixed position (called tail): this record is a pointer to the record of I that represents point P . $P \leftarrow L_{e_n}$ extracts the record from the head of L_{e_n} if L_{e_n} is a queue; from the tail otherwise.

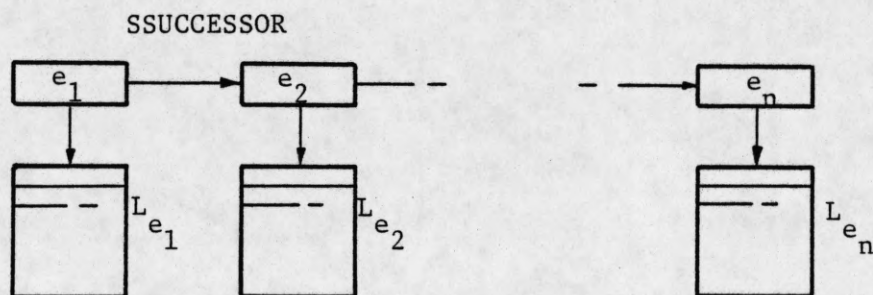


Figure 2.5. The s-structure.

The output mask. The polygons of the output mask will be described by means of the o-structure. The o-structure O contains the polygonal circuits of mask (A AND B). A cyclic list is used to represent each polygonal circuit. Every record of the list represents an endpoint of the polygonal circuit. O supports the following operation: $O \leftarrow P$, which adds point P to list O .

Description of Algorithm MASK INTERSECTION. The algorithm that sweeps the plane and forms the intersection regions has the following structure (operator $*$ is the list concatenation operator):

Procedure MASK INTERSECTION

begin X \leftarrow N/2 abscissae of the vertical edges sorted by x-coordinates;

S \leftarrow polygonal circuits of masks A and B;

I \leftarrow INTERSECTION(X,S);

O \leftarrow BOUNDARIES(S,I);

O' \leftarrow INCLUSION(X,S);

O \leftarrow O * O'

end.

We shall now present the three procedures INTERSECTION(X,S), BOUNDARIES(S,I) and INCLUSION(X,S). Procedure INTERSECTION(X,S) finds the intersection points and inserts them into I.

Procedure INTERSECTION(X,S)

begin I \leftarrow ϕ ; Y \leftarrow ϕ ;

while X \neq ϕ do begin

x \leftarrow MIN(X);

I' \leftarrow TRANSITION(x,Y,S);

I \leftarrow I * I'

end

return I

end.

Procedure TRANSITION(x,Y,S) is the advancing mechanism of the algorithm. The orthogonal geometry allows just one of four possible configuration types to occur for each value of x in the x-structure. These are illustrated in Figure 2.3. Thus there are four cases for TRANSITION(x,Y,S):

Case 1	INSERT(a) INSERT(b) INTERSECT(c,Y)
Case 2	DELETE(a) INSERT(b) INTERSECT(c,Y)
Case 3	INSERT(a) DELETE(b) INTERSECT(c,Y)
Case 4	DELETE(a) DELETE(b) INTERSECT(c,Y)

INSERT and DELETE are operations on Y. Procedure INTERSECT(c,Y) finds all the intersection points on a vertical edge c and inserts them into a queue I'. The intersection point P is the input of Procedure CROSSING(P). This procedure determines in constant time the incoming edge u and the outgoing edge v of P in mask (A AND B). The case analysis is described in the appendix.

Procedure INTERSECT(c,Y)

begin

I' ← ϕ ;

T ← ordinate of top endpoint (c);

B ← ordinate of bottom endpoint (c);

S ← SUCCESSOR(B) (/the least y-value greater than the y-value of B in Y/)

If S ≠ T then

foreach horizontal edge h between B and T do

begin

P ← intersection point of c and h;

(u,v) ← CROSSING(P); (/u and v are the two output edges meeting at P/)

I' ← P;

IEDGE(P) ← v;

L_u ← P

end;

return I'

end.

Procedure INTERSECTION(X,S) has an $O(N\log N+k)$ running time, where k is the number of intersections. It has an $O(N+k)$ memory use.

Procedure BOUNDARIES(S,I) builds the polygonal circuits of the output mask. Its inputs are S and I , and its output is O .

Procedure BOUNDARIES(S,I)

begin

$O \leftarrow \phi$;

while $I \neq \phi$ do begin

$P \leftarrow I$; (/a new region of masks (A AND B) is defined/)

while $P \neq \Lambda$ do

begin

$O \leftarrow P$; (/insertion into the output list O of an intersection point/)

$e \leftarrow \text{IEDGE}(P)$;

while $L_e = \phi$ do

begin

$E \leftarrow$ other endpoint of e

$O \leftarrow E$; (/insertion into the output list O of an endpoint/)

$e \leftarrow \text{SSUCCESSOR}(e)$

end

$P \leftarrow L_e$; (/an intersection point lies on e /)

IDELETE(P)

end

end

end.

Procedure BOUNDARIES builds the polygonal circuits of the output mask. These circuits contain intersection points and, possibly, endpoints of masks A and B . We start the description of each polygonal circuit from the intersection point extracted from the front of the i -structure. We march along its output edge and, if an intersection point lies on it, we insert the intersection point into the o -structure and delete it from the i -structure.

Otherwise we simply insert the appropriate endpoint of the edge into the O -structure. In either case we reach a new edge; the process is repeated for this edge, and terminates when we reach the starting intersection point and we do not find it in I . If I is not empty, some other polygonal circuits of the output mask have to be generated and the procedure is continued. Procedure $BOUNDARIES(S,I)$ constructs the polygonal circuits of the output mask marching along their edges. These are $O(N+k)$ and so the procedure runs in time $O(N+k)$. The memory use is the one required for the S and I structures and so it is $O(N+k)$.

Procedure $INCLUSION(X,S)$ performs the recognition of the polygons of mask A enclosed in polygons of mask B and of the polygons of mask B enclosed in polygons of mask A . The endpoints of the edges of the enclosed polygons are inserted into O . After the execution of $INTERSECTION(X,S)$ we know the set A' of polygons of mask A that do not intersect polygons of mask B . We shall use a new sequence X' of abscissae, defined as follows: X' contains the x -coordinates of all vertical edges of the polygons of B and the x -coordinate of one vertical edge for every polygon of A' , sorted by x -coordinates. Every entry corresponds to one of the configuration types shown in Figure 2.3.

Procedure $INCLUSION(X,S)$

begin

$X' \leftarrow$ abscissae of the vertical edges of the polygons of B and of one vertical edge of every polygon of A' sorted by x -coordinates;

$O' \leftarrow \phi$;

$Y \leftarrow \phi$;

while $X' \neq \phi$ do

```

begin
  x ← MIN(X');
  If c is an edge of mask B then INSERT and/or DELETE a and b from Y
  else T ← ordinate of top endpoint(c);
    ORIENTATION(T,B) ← orientation of the horizontal edge in Y
                        with ordinate equal to SUCCESSOR(T);
    If ORIENTATION(T,B) = right-to-left then
      O' ← endpoints of the boundary of the polygon of T
  end;
  return O'
end.

```

The inclusion of polygons of B in polygons of A is recognized by the same method.

Procedure INCLUSION(X,S) inserts into and/or deletes from the Y-structure the horizontal edges of mask B, and checks the orientation of the SUCCESSOR in Y of one horizontal edge for each polygon of mask A. The Y-structure is implemented as a height-balanced tree with a maximum of N entries; operations INSERT, DELETE and SUCCESSOR require each time $O(\log N)$. Thus, the performance time is $O(N \log N)$. The input and output data use memory $O(N)$. This is the memory use of the Y-structure, too. Thus, the memory use is $O(N)$.

Thus, we finally have:

Theorem: Algorithm MASK INTERSECTION runs in time $O(N \log N + k)$ using memory $O(N + k)$, where k is the number of intersection points.

2.3 Mask Union

The (A OR B) boolean operation requires the detection of the endpoints of the edges of the (A OR B) mask and the orientation of these edges. The endpoints belong to two disjoint classes:

- (1) The class of nonenclosed endpoints: these are the endpoints of the edges of the polygons of mask A that are not enclosed in a polygon of mask B and the endpoints of the edges of the polygons of mask B that are not enclosed in a polygon of mask A. Their determination is an immediate consequence of the result of Section 2.1.
- (2) The class of intersection endpoints: this class is the same class described in Section 2.1.

The edge orientation problem has already been solved in the appendix. Thus, an algorithm similar to Algorithm MASK INTERSECTION can solve the mask Union problem in time $O(N\log N+k)$ using space $O(N+k)$.

2.4 Further Results

The boolean union and intersection operations can be performed on both masks A and B and on their complements \bar{A} and \bar{B} or any combination of them. We note that the complement of a mask corresponds to reversing the orientations of all the edges of that mask. (Convention for complemented masks: all regions are bounded; the boundary of the chip is the most external boundary of all masks.) Once this operation has been performed, Algorithms MASK INTERSECTION or MASK UNION can be applied to the masks without any change. Thus, even the boolean operations \bar{A} AND B, A AND \bar{B} , \bar{A} AND \bar{B} , or the equivalents with the OR boolean connective can be performed in time $O(N\log N+k)$ using space $O(N+k)$.

2.5 An Example

This section illustrates the activities performed by Algorithm MASK INTERSECTION. The algorithm runs on the example of Figure 2.6.

Without loss of generality, all abscissae and ordinates are assumed to be integers. (In the general case an $O(N\log N)$ sorting and ranking achieves this result.)

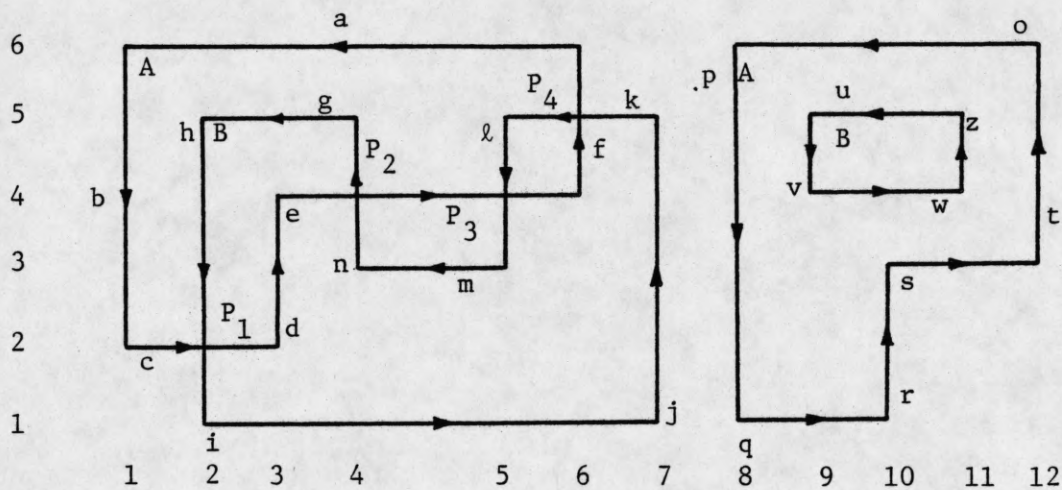
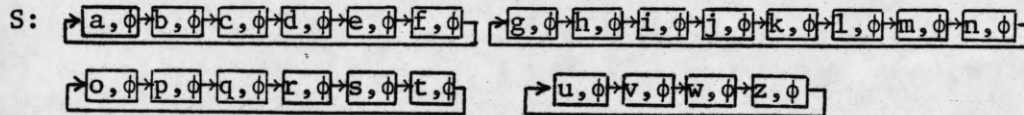


Figure 2.6. Input masks.

We begin by considering the contents of the data structures before the execution of Algorithm MASK INTERSECTION. The x-structure contains the abscissae of the vertical edges of masks A and B and specifies the mask and the type of configuration of each entry (the four types are shown in Figure 2.3). Each record consists of three fields: the first contains the abscissa of the vertical edge, the second the mask, and the third the type of configuration of the entry.

X: $\boxed{1, A, 1} \rightarrow \boxed{2, B, 1} \rightarrow \boxed{3, A, 3} \rightarrow \boxed{4, B, 2} \rightarrow \boxed{5, B, 3} \rightarrow \boxed{6, A, 4} \rightarrow$
 $\rightarrow \boxed{7, B, 4} \rightarrow \boxed{8, A, 1} \rightarrow \boxed{9, B, 1} \rightarrow \boxed{10, A, 3} \rightarrow \boxed{11, B, 4} \rightarrow \boxed{12, A, 4} .$

Each record of the s-structure consists of two fields: the first contains the name of the edge, the second a pointer to the list of pointers to the intersection points lying on that edge. This list is initially empty.



Execution of Procedure INTERSECTION(X,S)

Operations on X	Operations on Y	Operations on S	Operations on I
$x \leftarrow 1, A, 1$	INSERT (a) INSERT (c) INTERSECT (b)	none	none
$x \leftarrow 2, B, 1$	INSERT (g) INSERT (i) INTERSECT (h)	$L_h \leftarrow P_1$	$I \leftarrow P_1$ $IEDGE(P_1) \leftarrow c$
$x \leftarrow 3, A, 3$	DELETE (c) INSERT (e) INTERSECT (d)	none	none
$x \leftarrow 4, B, 2$	DELETE (g) INSERT (m) INTERSECT (n)	$L_e \leftarrow P_2$	$I \leftarrow P_2$ $IEDGE(P_2) \leftarrow n$
$x \leftarrow 5, B, 3$	DELETE (m) INSERT (k) INTERSECT (l)	$L_l \leftarrow P_3$	$I \leftarrow P_3$ $IEDGE(P_3) \leftarrow e$
$x \leftarrow 6, A, 4$	DELETE (c) DELETE (a) INTERSECT (f)	$L_f \leftarrow P_4$	$I \leftarrow P_4$ $IEDGE(P_4) \leftarrow k$
$x \leftarrow 7, B, 4$	DELETE (i) DELETE (k) INTERSECT (j)	none	none
$x \leftarrow 8, A, 1$	INSERT (o) INSERT (q) INTERSECT (p)	none	none
$x \leftarrow 9, B, 1$	INSERT (u) INSERT (w) INTERSECT (v)	none	none
$x \leftarrow 10, A, 3$	DELETE (q) INSERT (s) INTERSECT (r)	none	none
$x \leftarrow 11, B, 4$	DELETE (w) DELETE (u) INTERSECT (z)	none	none
$x \leftarrow 12, A, 4$	DELETE (s) DELETE (o) INTERSECT (t)	none	none

Execution of Procedure BOUNDARIES(S,I)

Operations on I	Operations on S	Operations on O
$P_1 \leftarrow I; c \leftarrow \text{IEDGE}(P_1)$	$P_1 \leftarrow L_h$	$O \leftarrow P_1$
	C final endpoint of c	$O \leftarrow C$
	D final endpoint of d	$O \leftarrow D$
	P_2 intersection point on e	
$P_2 \leftarrow I; n \leftarrow \text{IEDGE}(P_2)$	$P_2 \leftarrow L_e$	$O \leftarrow P_2$
	N final endpoint of n	$O \leftarrow N$
	G final endpoint of g	$O \leftarrow G$
	P_1 intersection point on g	
$P_3 \leftarrow I; c \leftarrow \text{IEDGE}(P_3)$	$P_3 \leftarrow L_\ell$	$O \leftarrow P_3$
	E final endpoint of e	$O \leftarrow E$
	P_4 intersection point on f	
$P_4 \leftarrow I; k \leftarrow \text{IEDGE}(P_4)$	$P_4 \leftarrow L_f$	$O \leftarrow P_4$
	K final endpoint of k	$O \leftarrow K$
	P_3 intersection point on ℓ	

Execution of Procedure INCLUSION(X,S)

X': 1,A, 1 → 3,A, 3 → 6,A, 4 → 8,A, 1 → 9,B, 1 → 10,A, 3 → 12,A, 4

Operations on X'	Operations on Y	Operations on O
x ← 1,A, 1	INSERT(a) INSERT(c)	none
x ← 3,A, 3	DELETE(c) INSERT(e)	none
x ← 6,A, 4	DELETE(c) DELETE(a)	none
x ← 8,A, 1	INSERT(o) INSERT(q)	none
x ← 9,B, 1	ORIENTATION(T,A)	insert the endpoints of the polygon of x in O.
x ← 10,A, 3	DELETE(q) INSERT(s)	none
x ← 12,A, 2	DELETE(s) DELETE(o)	none

The polygon circuits of the output mask are shown in Figure 2.7.

Polygonal circuits 1 and 2 are the output of Procedure BOUNDARIES(S,I);

polygonal circuit 3 is the output of Procedure INCLUSION(X,S).

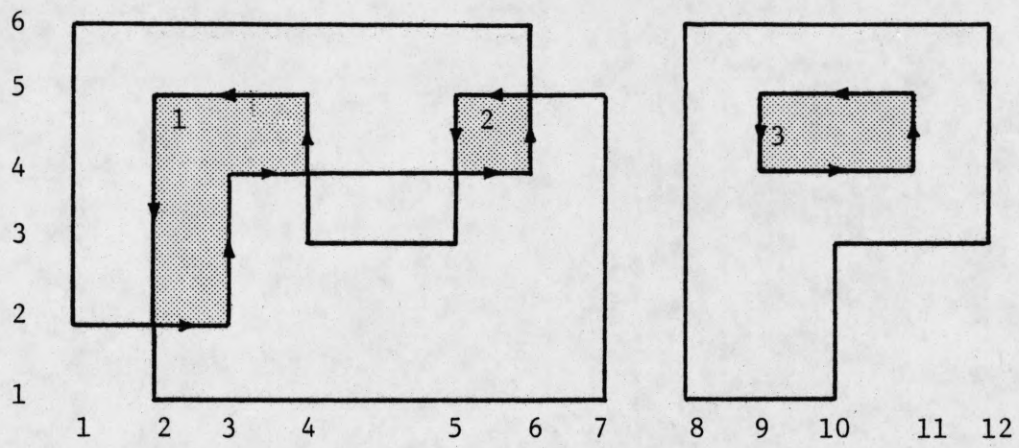


Figure 2.7. The output mask.

3. ALGORITHMS FOR THE VERIFICATION OF A BOOLEAN EXPRESSION OF MASK VARIABLES.

This chapter presents the solution of a generalization of the problem stated in Chapter 1. Instead of two masks A and B, we consider n planar masks A_1, A_2, \dots, A_n . Every mask is a set of polygonal circuits subject to the three rules stated at the beginning of Chapter 2. We want to determine the regions of the plane that verify a boolean expression whose variables are masks A_1, A_2, \dots, A_n and whose operators are the standard AND, OR and NOT connectives. Without loss of generality, we may assume that all complements be placed directly on the variables, since repeated application of DeMorgan's rule can transform an AND-OR-NOT expression into one in this form without changing its length. Variables may appear more than once in the boolean expression. Figure 3.1 illustrates an example. The region that verifies the boolean expression is marked.

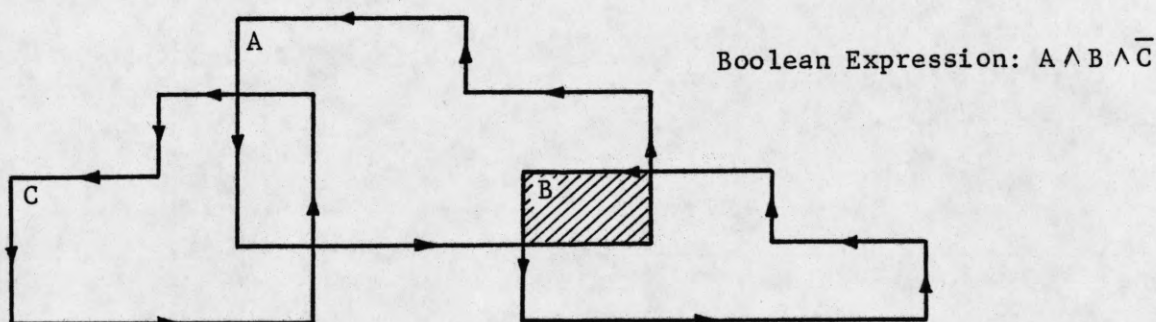
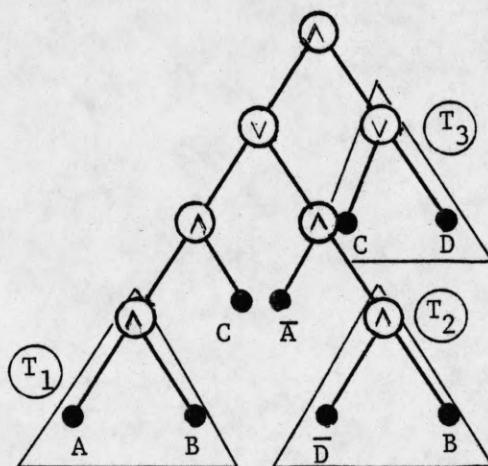


Figure 3.1. Verification of a boolean expression.

The boolean expression is an additional input to our problem. We will call h its size: h is the number of variables and of AND and OR connectives in the expression. The performance of the procedure will depend both on N and on h ; $N = N_1 + N_2 + \dots + N_n$ is the sum of the edges of the polygon of the masks A_1, A_2, \dots, A_n .

3.1 A Straightforward but Inefficient Solution

For a given boolean expression E , $T(E)$ denotes the computation tree of E . The leaves of $T(E)$ are the variables of E ; each internal vertex is associated with one of the boolean connectives AND and OR. We note that algorithms MASK INTERSECTION or MASK UNION developed in the preceding chapter can be applied to some of the subtrees of $T(E)$. These are the terminal subtrees: subtrees of $T(E)$ with three vertices, whose leaves are leaves of $T(E)$. Figure 3.2 illustrates an example.



Algorithm Mask INTERSECTION or Algorithm Mask UNION, as appropriate, are applied to T_1 , T_2 , T_3 .

Figure 3.2. Boolean expression tree.

Once we have applied Algorithm MASK INTERSECTION or Algorithm MASK UNION to an appropriate terminal subtree, we substitute this terminal subtree with a new variable that represents the mask just obtained. In this manner we get a new expression E' . We use the same technique with $T(E')$. If this technique is applied, in the appropriate sequence, as many times as there are internal vertices of $T(E)$, the last

application computes the function of the root of $T(E)$ and we have obtained the mask that solves our problem. We shall now describe the algorithm BOOLEAN MASK 1, which carries out this task.

Let us call MASK UNION ($\text{leftson}(V)$, $\text{rightson}(V)$), MASK INTERSECTION ($\text{leftson}(V)$, $\text{rightson}(V)$) the mask derived from the application of Algorithms MASK UNION, MASK INTERSECTION, respectively, to the terminal subtree rooted at V . CONN(V) is the boolean connective associated with vertex V . Algorithm BOOLEAN MASK 1 has the following pebble game [7] structure:

Procedure BOOLEAN MASK 1

begin

$V \leftarrow \text{root of } T(E);$

while $T(E)$ has more than one vertex do

begin

If $\text{leftson}(V) = \text{leaf}$ then

If $\text{rightson}(V) = \text{leaf}$ then

If $\text{CONN}(V) = \text{AND}$ then $V \leftarrow \text{MASK INTERSECTION}(\text{leftson}(V), \text{rightson}(V))$

else $V \leftarrow \text{MASK UNION}(\text{leftson}(V), \text{rightson}(V));$

$V \leftarrow \text{father}(V)$

else $V \leftarrow \text{rightson}(V)$

else $V \leftarrow \text{leftson}(V)$

end

end.

Algorithms MASK UNION and MASK INTERSECTION are executed $\frac{h-1}{2}$ times; indeed this is the number of internal vertices of $T(E)$, i.e., the number of boolean connectives of E . Using the result obtained in Section 2.2 of the preceding chapter, if N_1 and N_2 are the numbers of edges of the polygons of masks A_1 and A_2 , respectively, Algorithm MASK INTERSECTION or Algorithm MASK UNION works in time $O((N_1 + N_2)\log(N_1 + N_2) + k_{12})$ using space $O(N_1 + N_2 + k_{12})$, where k_{12} is the number of points in which the polygons of A_1 and A_2 intersect. The number of edges of mask A_{12} , the mask that results from combining masks A_1 and A_2 , is $O(N_1 + N_2 + k_{12})$. Generally, when processing an internal vertex V of $T(E)$, its left and right sons represent masks obtained from successive applications of Algorithm MASK INTERSECTION and/or Algorithm MASK UNION. The number of edges of these masks is always less than or equal to $O(N+k)$, where k is the total number of points in which the polygons of the n masks A_1, A_2, \dots, A_n intersect. Thus, the time used in the construction of the polygons whose interiors verify the boolean expression is $O(N+k)h \log N$. The memory use is $O((N+k)\log h)$. In fact a pebble game on a binary tree of h vertices needs at most $\lceil \log_2 h \rceil + 1$ pebbles [7] and each vertex of $T(E)$ uses memory bounded by $O(N+k)$.

Algorithm BOOLEAN MASK 1 has time and space performances depending on N , h , and k ; k , the number of intersection points, is the size of the output of the algorithm. It is possible, however, that just a small subset of the k intersection points be necessary for the description of the regions that verify the boolean expression. Let us call k' the number of the intersection points in the output mask (final intersection points). Obviously, we have $0 \leq k' \leq k$. Figure 3.3 illustrates an example and the region where a boolean expression E is verified. We note that $k > k'$.

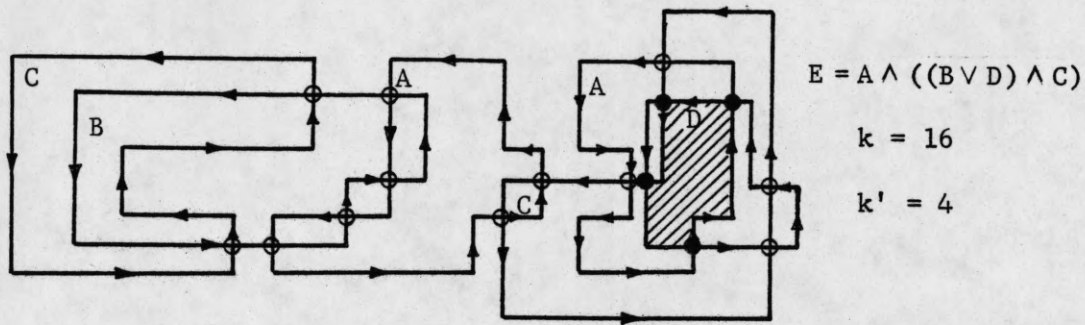


Figure 3.3. The number of intersection points k (solid and hollow in the figure) is greater than the number of final intersection points k' (solid in the figure).

It is therefore highly desirable to seek an algorithm whose time and space performances depend on N , h , and on k' , rather than on k . This is the subject of Section 3.2.

3.2. Outline of Algorithm BOOLEAN MASK 2

We present a plane-sweep algorithm that, by sweeping the plane unidirectionally, determines the boundaries of the polygons of the output mask.

The input data are: a sequence of N points, $V_i^{(1)}$, ($i=1, \dots, N_1$), \dots , $V_j^{(n)}$, ($j=1, \dots, N_n$), and a boolean expression E of size h . The $V_i^{(k)}$'s are the endpoints of the edges of the polygons of mask A_1, \dots, A_n . Every endpoint is associated with the data concerning its incoming and outgoing edges.

The desired result is a description of all regions that verify the boolean expression, where each region is given by a cyclic list of its vertices. This objective is achieved once we have found the endpoints of the edges of the polygons of the output mask and defined their incoming and outgoing edges in the output mask. In fact we can later concatenate these endpoints by means of an algorithm similar to Procedure BOUNDARIES described in the preceding chapter.

Data Structures maintained by the algorithm. The algorithm we are about to describe is called BOOLEAN MASK 2 and uses four data structures. These data structures are very similar to the ones presented in Section 2.2 of the preceding chapter. Two of these, the x-structure and the y-structure, are common to all plane sweep algorithms. As the line that sweeps the plane advances in the direction of the x-axis, the x-structure represents a queue of tasks to be accomplished. The y-structure represents the state of the current cross section.

The third data structure, the i-structure, is specific to our needs. It stores only the k' final intersection points and those endpoints of the original input polygons that lie in a region where the boolean expression is true.

The fourth data structure, the s-structure, represents the polygonal circuits of the input masks. The algorithm uses also the computation tree $T(E)$ of the boolean expression E .

The x-structure X contains the x-coordinates of the vertical edges of the input polygons. Every entry describes one of the four configuration types shown in Figure 3.4. Every edge of X has a pointer to its record in the s-structure. The x-structure can be implemented as a sorted linear list. Sorting the elements of X and storing them in increasing order takes time $O(N \log N)$.

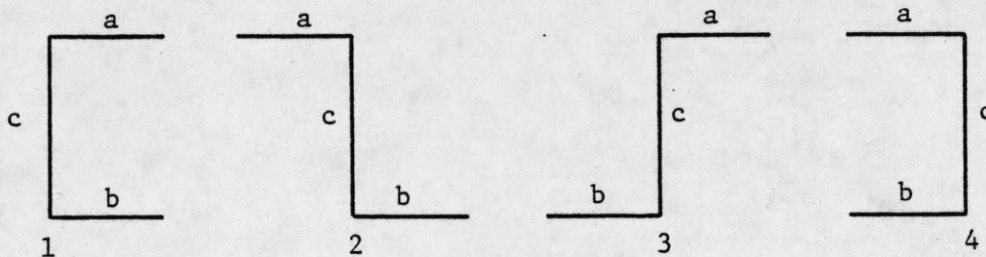


Figure 3.4. The four possible configuration types for a given value $x \in X$.

The y-structure Y contains the description of the current cross section. Y has an entry for each horizontal edge intersected by the sweep line. These entries are sorted according to increasing y-coordinates. Y must support operations INSERT and DELETE. This structure is the crucial component of the algorithm. In the next section we shall describe a particular implementation of Y , which allows the identification of the final intersection points during the execution of operations INSERT and DELETE.

The i-structure I provides the information about the intersection points and their incoming and outgoing edges, and the endpoints of the input polygons that lie in a region where the boolean expression is true.

It is initialized to be empty and terminates empty. Data structure I is a linear list. Every record of the queue represents an endpoint P of the output mask and has a pointer, IEDGE(P), to the record of the s-structure that represents the outgoing edge in the output mask. I supports the following operations: $I \leftarrow P$ adds point P to list I; $P \leftarrow I$ extracts point P from list I; IDELETE(P) deletes record P from I.

The s-structure S provides the information about the polygonal circuits of the input masks. A cyclic list is used to represent each polygonal circuit. Each record of the list represents an edge of the polygonal circuit, points to the following edge, SUCCESSOR(e), with respect to the orientation of the polygon, and to a list L_e of pointers to the records of I representing the final intersection points lying on edge e. The s-structure is shown in Figure 3.5. S supports the following operations: $L_{e_n} \leftarrow P$, $P \leftarrow L_{e_n}$, SUCCESSOR(e). $L_{e_n} \leftarrow P$ adds a record to the list L_{e_n} of edge e_n in a fixed position (called tail); this record is a pointer to the record of I that represents point P. $P \leftarrow L_{e_n}$ extracts the record from the head of L_{e_n} , if L_{e_n} is a queue; from the tail otherwise.

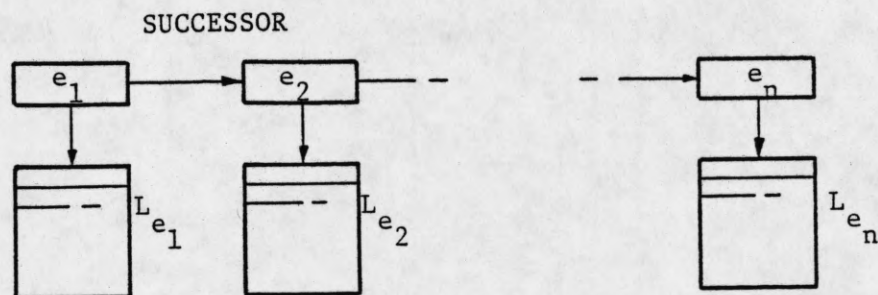


Figure 3.5. The s-structure.

The output mask. The output mask will be described by the o-structure. The o-structure O contains the polygonal circuits of the output mask. A cyclic list is used to represent each polygonal circuit. Each record of the list represents an endpoint of the polygonal circuit. Structure O supports the following operation in constant time: $O \leftarrow P$ adds point P to list O.

3.3 Operations on the Y-Structure.

Every entry of X describes one of the configuration types of Figure 3.4. We have to insert into and/or delete from Y (as appropriate) the ordinates of the horizontal edges a and b, and check whether the vertical edge c contains any endpoint of the output mask (final endpoints), determined by its intersections with the horizontal edges whose ordinates are between the ones of a and b. The technique we are about to describe will find and report the final endpoints, without processing the other intersections determined by c.

The Y-structure is implemented as a height balanced tree. The vertices of this tree will be called nodes, in order to differentiate them from the vertices of T(E), tree of the boolean expression E, already called vertices. Every node P of this tree contains two records: the first record, Y[P], contains the value of the ordinate of the horizontal edge represented by P; the second record, MASK[P], contains the name of the mask of that edge.

The insertion into and/or deletion from Y of the horizontal edges a and b defines two insertion/deletion paths in Y. Figure 3.6 illustrates an example of these paths.

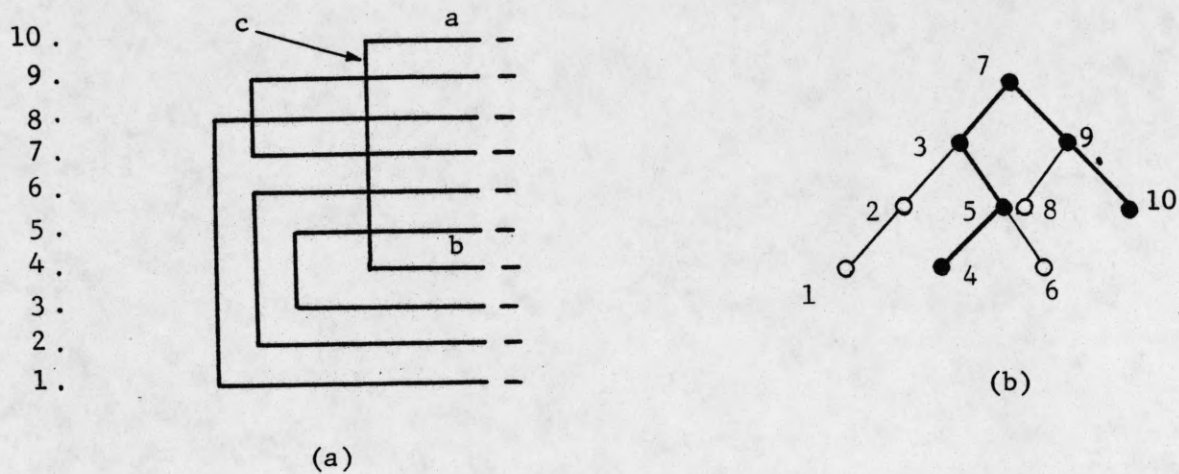


Figure 3.6. Insertion paths of a and b into Y. (a) planar configuration; (b) Y-structure. The paths are shown in heavy lines.

We note that some of the horizontal edges that intersect vertical edge c are represented by nodes of Y that belong to the insertion/deletion paths. In the example of Figure 3.6 these are the horizontal edges with ordinates 5, 7, and 9. The other horizontal edges that intersect vertical edge c are represented by nodes of Y that do not lie on the insertion/deletion paths. That is the case of the horizontal edges with ordinates 6 and 8 in Figure 3.6.

We present an algorithm that, by means of a First Sweep of the plane

1. reports the final endpoints determined by the intersection of the vertical edge c with the horizontal edges whose nodes lie on the insertion/deletion paths;

2. decides if the vertical edge c contains final endpoints determined by its intersections with the horizontal edges whose nodes do not lie on the insertion/deletion paths.

The information derived from point 2 will be used later during a Second Sweep of the plane (to be described in Section 3.5) for the report of those final endpoints.

3.4 First sweep

We note that the nodes of Y that represent the horizontal edges a and b in Y may appear one in the left and one in the right subtree of the root or both in the same subtree. An example of these two cases is illustrated in Figure 3.7.

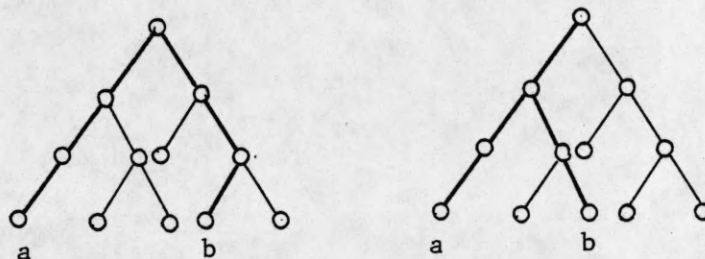


Figure 3.7. Disposition of the nodes of the horizontal edges a and b in Y .

The insertion/deletion paths have one or more nodes in common. These are the nodes of the path from the root to the first common ancestor of nodes a and b . The first common ancestor u of nodes a and b is the only node of this path that represents a horizontal edge intersected by c . The other nodes representing horizontal edges intersected by c are: the nodes that lie together with their leftson on the path from u to a , and the nodes of the right subtrees of these nodes; the nodes that lie together with their rightson on the path from u to b and the nodes of the left subtrees of these nodes.

Figure 3.8 illustrates an example: the nodes representing horizontal edges lying on the insertion/deletion paths and intersected by vertical edge c are shown solid.

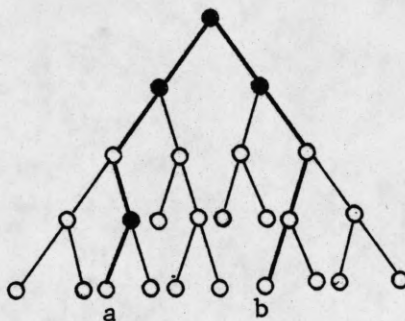


Figure 3.8. Insertion/deletion paths of the horizontal edges a and b . The nodes belonging to these paths and representing horizontal edges intersected by the vertical edge c are marked.

The first problem we want to solve is the recognition of final endpoints among the intersection points determined by the vertical edge c and the horizontal edges just described. We recall that an intersection point is a final endpoint if it verifies the boolean expression E . The masks that enclose the intersection point are the variables of E that are set to 1 in $T(E)$. The other variables are set to 0. If this assignment of variables verifies E , we have found a final endpoint. The masks that enclose an intersection point can be determined if we know the number of horizontal edges of each mask that lie below the intersection point. When an odd number of horizontal edges of mask A_j lies below the intersection point

at its abscissa, that intersection point is enclosed by A_j . In fact, the regions of A_j are bounded (recall our convention for complemented masks), and so any vertical line ℓ cuts an even number of horizontal edges of A_j . As the interiors of the polygons of A_j do not overlap, we have a segment of A_j between the first and second intersections of the horizontal edges of A_j with ℓ , and, generally, between the intersections of an odd and an even numbered horizontal edge of A_j with ℓ (we start from the bottom). Figure 3.9 illustrates an example.

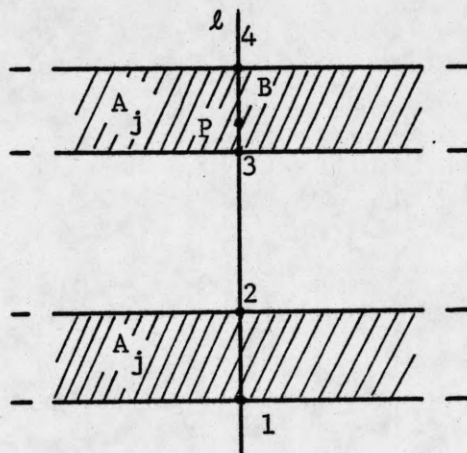


Figure 3.9. Point P is enclosed by mask A_j .

Thus, the information that we need is the parity of the number of horizontal edges of each mask lying below an intersection point. This information can be derived from the Y-structure if we attach a linear list, the Lower Mask List, $LM(P)$, to each of its nodes P. Each record of this list is a pointer to a leaf of $T(E)$, the tree of the boolean expression E. There is a pointer to each variable of E (i.e., a mask) that has an odd number of horizontal edges in the left subtree of that node. As the maximum number

of different variables in the boolean expression E is $\frac{h+1}{2}$, this is also the maximum number of records of each Lower Mask List. The records of the LM-list are ordered according to a depth-first-search-visit of $T(E)$. If a variable appears more than once in E , pointers exist only to its first appearance. Figure 3.10 illustrates an example of the LM lists. Whenever an intersection point (defined by the vertical edge c and a

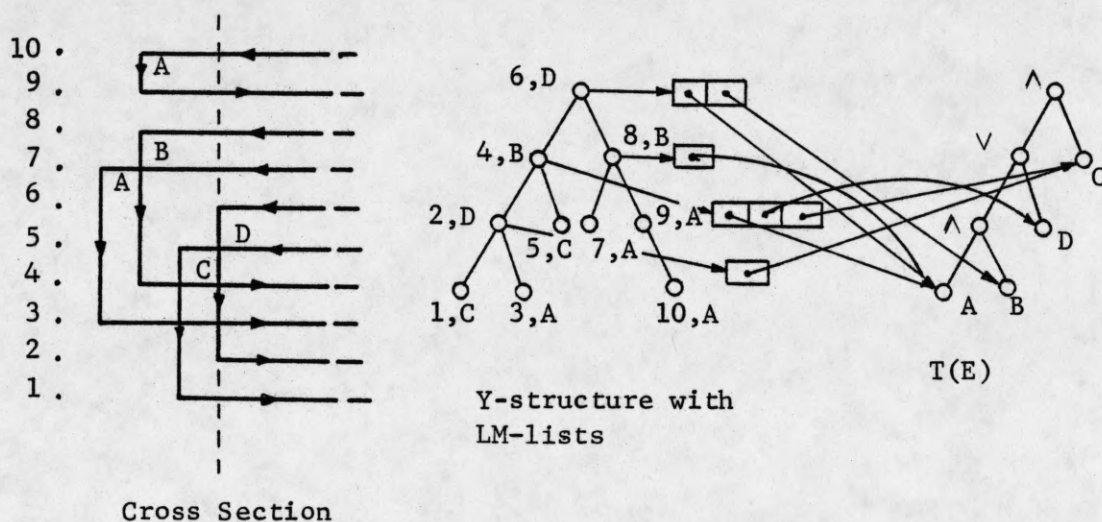


Figure 3.10. Illustration of the LM lists.

horizontal edge whose corresponding node P in Y lies on the insertion/deletion paths) has to be checked for the masks enclosing it, we make use of the LM-lists. We note that $LM(P)$ is not always sufficient for our needs. That is shown in Figure 3.11. In fact $LM(P_1)$ does not describe all the horizontal edges that lie below the intersection point defined by

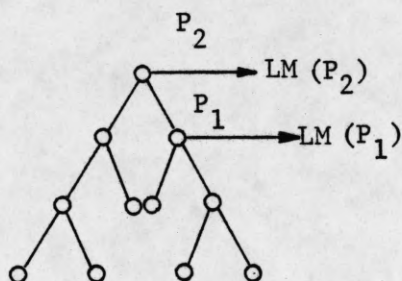


Figure 3.11. The horizontal edges below P_1 depend on $LM(P_1)$ and $LM(P_2)$.

the vertical edge c and the horizontal edge represented by P_1 . We note that we need to compute a symmetric difference (in the set-theoretic sense) of the LM-lists of nodes P_1 and P_2 (i.e., we merge the two lists with the added condition that, any time the same record appears in both lists, that record is deleted). This corresponds to the use of the LM-list as the rank field in Knuth's [8] rank tree.

Once the masks that enclose an intersection point are known, we can check the verification of the boolean expression E . This process requires time proportional to the length h of E . The update of the LM-lists is performed during the traversal of the nodes of the insertion/deletion paths. We insert into or delete from the LM-lists of these nodes the pointer to the variable of

$T(E)$ that represents the mask inserted/deleted. It requires time proportional to the length of the LM-list of each node. The maximum number of nodes traversed during an insertion/deletion process is $O(\log N)$, when Y has $O(N)$ entries. Thus, the recognition of the final endpoints determined by the vertical edge c and the horizontal edges whose representatives lie on the insertion/deletion paths, and the update of the LM lists of the nodes on the insertion/deletion paths, are performed in time $O(h \log N)$.

Every final endpoint has an incoming and outgoing edge in the output mask. Figure 3.12 illustrates two possible configurations. Case (a) corresponds to the intersection of the regions determined by the two input

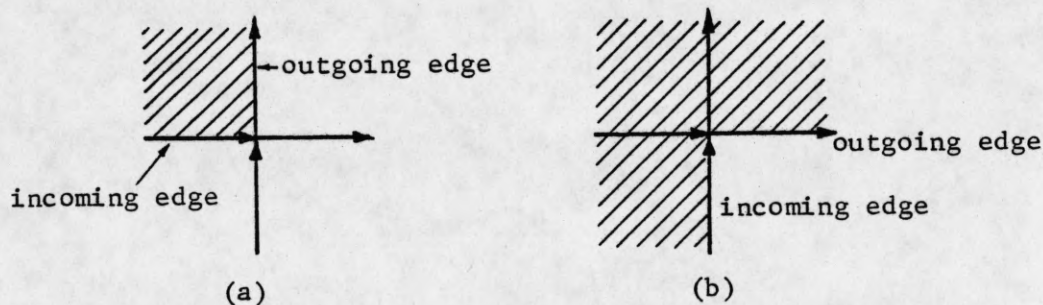


Figure 3.12. Final intersection points.

edges, case (b) to their union. We will call a final endpoint of the first kind a left-turn endpoint, while the second is a right-turn endpoint. For every final endpoint we have to determine if it is a left- or right-turn endpoint. We note that a left-turn endpoint corresponds to a situation where the masks bounded by the two input edges have to be simultaneously present in order to verify the boolean expression. Instead, a right-turn endpoint corresponds to a situation where the presence of any of the two masks bounded by the

input edges is sufficient to the verification of the boolean expression. Thus, if the LM-lists lead to the recognition of a final endpoint, we can determine if that point is a left- or right-turn endpoint. We check the verification of the boolean expression when we assume that the mask that is bounded by one of the two input edges is set to 0 in $T(E)$. If E is still verified, we have a right-turn endpoint, otherwise a left one. This check is performed once for each final endpoint and does not change the time performance previously stated.

The second problem we have to solve is the recognition of final endpoints among the intersection points determined by the vertical edge c and the horizontal edges that do not lie on the insertion/deletion paths. These horizontal edges are represented by the nodes of some subtrees of Y . These are the right subtrees of the nodes that, together with their leftson, lie along the path from the leftson of the first common ancestor of nodes a and b to node a , and the left subtrees of the nodes that, together with their rightson, lie along the path from the right son of the first common ancestor of nodes a and b to node b . Figure 3.13 illustrates an example.

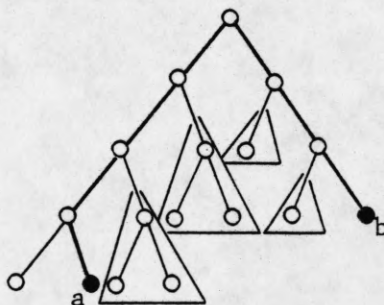


Figure 3.13. Subtrees of the horizontal edges intersected by c .

We wish to avoid visiting these subtrees, or we would process all the intersection points and not only the final ones. We note that every subtree determines a vertical span s . This is a vertical segment whose endpoints have the abscissa of the vertical edge c and the ordinates of the horizontal edges represented by the leftmost and rightmost leaves of the subtree. Figure 3.14 illustrates an example.

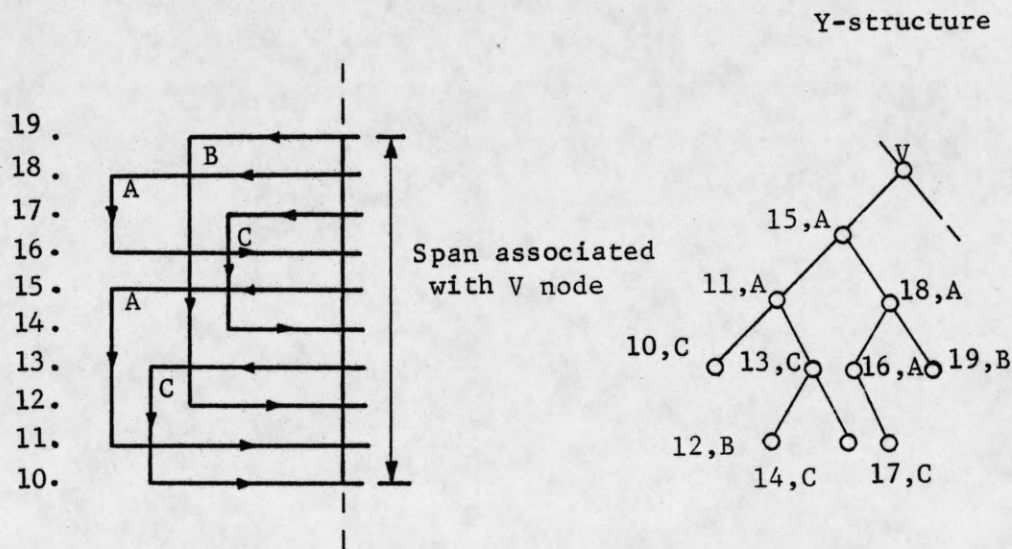


Figure 3.14. The vertical span of a subtree.

Each point of the vertical span verifies a set of subexpressions of E . If we keep a representation of these variables and subexpressions at the root of every subtree (as a list attached to the root node), we know the subexpressions which are verified somewhere in its span without visiting its subtree. This information can be provided by the Left and Right Subtree Mask Lists that we attach to each node of Y . $LSM(P)$ and $RSM(P)$ denote the Left and Right Subtree Mask lists of node P , respectively. The $LSM(P)$

and RSM(P) lists are linear lists. Each record is a pointer to a vertex of T(E) representing a boolean subexpression verified at at least one interval (a segment) contained in the vertical span of the left or right subtree of P, and whose ancestors in T(E) are not pointed to from any other record of LSM(P) or RSM(P). The maximum number of records of LSM or RSM is $\frac{h-1}{2}$, which corresponds to the worst case configuration of T(E) illustrated in Figure 3.15.

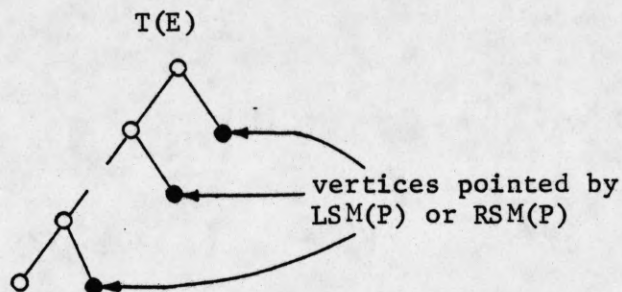


Figure 3.15. Worst case memory requirement for RSM or LSM.

In addition, the vertical span of P may be enclosed by one or more masks, as it is shown in the example of Figure 3.16. It is important to recognize such situations as they can modify each subexpression verified by the subtree considered. The masks that enclose the vertical span of P are identified if we consider the list of the masks that enclose P (the root of the subtree that represents the vertical span). This list, which is derived from the LM-list of P and the LM-lists of the ancestors of P (as described above), is a collection of pointers to the leaves of T(E). Some of these leaves may be pointed to also from the LSM or RSM lists of P, and others have their

$A \wedge B$; (ii) mask D encloses the span. The first item is obtained by means of the LSM-list of node 15, the second is derived from the LM-lists of node 15 and of its ancestor, node 10. The vertices of $T(E)$ pointed from these lists are set to 1 and their verification leads to the verification of the boolean expression E .

Once we know the variables and subexpressions verified in the vertical span and the masks that totally enclose it, the check for the verification of the boolean expression requires time proportional to its length, i.e., $O(h)$. Thus, at each node P of the insertion/deletion paths we check, by means of its LM-list and the ones of its ancestors, if it represents a horizontal edge that intersects c in a final endpoint, and we check if the appropriate subtree of P , whose nodes represent horizontal edges intersected by c , has final endpoints in its vertical span. This final check is performed by means of the LSM- or RSM-list of node P and the list of the masks that enclose P . Edge c is marked at this stage when this situation occurs. The report and insertion into the I-list of the final endpoints, determined by the intersection of c with the horizontal edges represented by the nodes lying on the insertion/deletion paths, and the recognition of the existence of final endpoints determined by c in the vertical spans, represented by the subtrees of nodes of the insertion/deletion paths, are completed in time $O(h \log N)$.

3.5 Second Sweep

This portion of the algorithm performs a second sweep of the plane. The role of axes x and y is interchanged. U and W are the equivalents of

X and Y. The four possible configurations for each entry of U consist of two vertical edges and a horizontal one. The vertical edges are inserted into W and intersected by the horizontal ones. The only difference between the First Sweep and the Second Sweep is that the abscissae of the vertical edges marked during the execution of Firstsweep are stored in the LSM and RSM lists with the pointers to the subexpression of T(E) they verify. Thus, any time a final intersection point is determined by a horizontal edge and a vertical edge belonging to a subtree not traversed in the insertion/deletion process, we know the edges that define that final endpoint.

The running time is equal to the one of First Sweep plus the time required for the report of the final intersections determined by vertical edges represented by nodes not in the insertion/deletion paths. Thus, an upper bound to the running time is $O(N h \log N + k')$, where k' is the number of final endpoints. The increase in space use is described in the next section.

3.6 Memory use of the Y-Structure

The memory use of the y-structure depends on the number of records stored in the lists attached to each node of Y. We already know that list LM may have a maximum of $\frac{h+1}{2}$ records, and lists LSM and RSM may have $\frac{h-1}{2}$ each, at most.

We claim that list LSM (or RSM) may reach the condition of maximum memory occupancy only if it pertains to a node P of Y, which has a left or right subtree with at least $\frac{h-1}{2}$ nodes. In fact the existence of $\frac{h-1}{2}$ records in the LSM or RSM list requires the verification of $\frac{h-1}{2}$ vertices of T(E) as in the example of Figure 3.15. The vertices are verified only

if the nodes in the left or right subtree of P represent (field MASK[P] of the node) the masks corresponding to these vertices.

A node P of Y whose three lists are in their maximum memory occupancy configuration is called a saturated node. Since to have $(h-1)/2$ nodes in the subtree rooted at P is a necessary condition for P to be saturated, an upper bound to the number of saturated nodes is obtained by assuming as saturated each node whose subtree contains $(h-1)/2$ nodes. We now calculate this number. The saturated node subtree is conventionally a subtree of Y rooted at the root of Y and such that each of its leaves has at least $(h-1)/2$ nodes in its subtree. Let m be the number of leaves of the saturated node subtree. q is the number of nodes of Y. We have:

$$q = 2m - 1 + m\left(\frac{h-1}{2}\right)$$

and

$$m = 2 \frac{q+1}{h+3} \tag{3.1}$$

The maximum number of nodes of the saturated subtree is derived from (3.1) when we consider Y at its maximum size, i.e., $q = N$. We have $2m-1 = 4 \frac{N+1}{h+3} - 1$. Each of these nodes uses $O(h)$ memory locations. The total memory use of the saturated subtree is $O(N)$.

The non-saturated subtrees are the subtrees of Y rooted at a leaf of the saturated node subtree. Each of these subtrees has a maximum of $h-1$ nodes. Expression (3.1) is the number of non-saturated subtrees in Y. The height of a non-saturated subtree is at most $1.44 \log_2(h-1)$ (it is a height balanced tree). The conditions of maximum memory occupancy correspond to the case of no repetition of the same mask among the nodes of a nonsaturated subtree. In fact, in case of no repetition the pointer to the variable of T(E) specified by the record MASK[P] of each node of the

subtree appears in all the lists of the ancestors of that node in the nonsaturated subtree. Each node has at most $1.44 \log_2(h-1)$ ancestors in the nonsaturated subtree, and so an upper bound to the memory use of each nonsaturated subtree is $O(h \log h)$. If we multiply this result by (3.1) we have the total memory use of the nonsaturated subtrees: $O(N \log h)$.

The names of the k'' vertical edges marked during the execution of First Sweep ($k'' \leq N/2$) have to be stored in the LSM and RSM lists of the W -structure during the execution of Second Sweep. They need to be stored once for each vertex of $T(E)$ they verify. In fact, we store the pointers and all the names of the corresponding vertical edges containing final intersection points in the appropriate RSM-lists of the ancestors of the leftmost leaf of W and in the appropriate LSM-lists of the ancestors of the rightmost leaf of W . Thus, when we traverse one of these nodes we have a complete description of its right or left subtree, respectively. When we traverse one of these subtrees, we just need to store the changes in the boolean subexpression verified by each of the k'' vertical edges possibly represented in that subtree, as it is shown in the example of Figure 3.18.

Any of the k'' vertical edges can verify no more than h subexpressions (i.e., the nodes of $T(E)$) and so each of them occupies memory $O(h)$. Thus, the total memory use for storing the k'' vertical edges marked in First Sweep is $O(h k'')$.

The total memory use of Algorithm BOOLEAN MASK 2 is the sum of the memory use of the y -structure, $O(N \log h + k'' h)$, and of the endpoints of the output mask, k' . Thus, we have $O(N \log h + k'' h + k')$ total memory use.

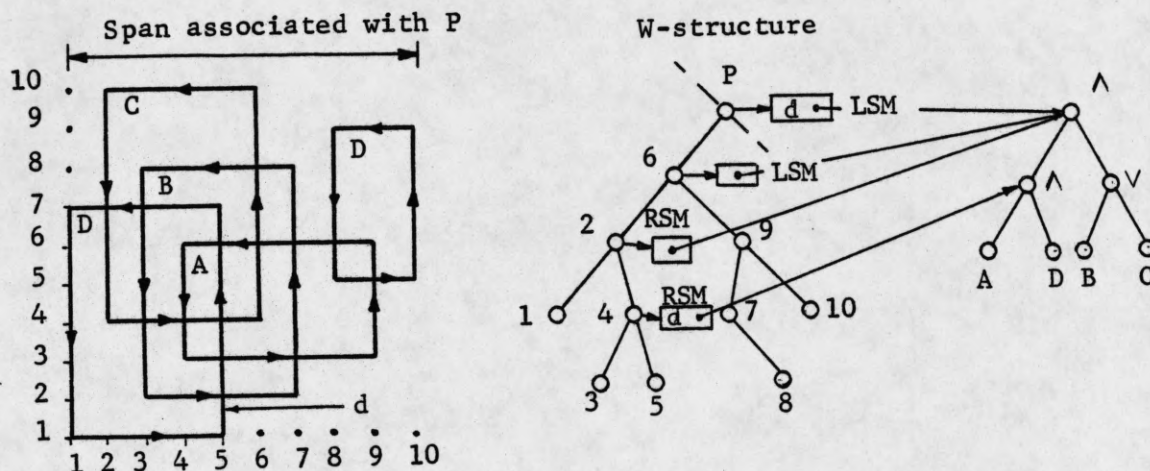


Figure 3.18. Example of memorization of one of the k'' vertical edges.

3.7 Final remarks.

Procedures First Sweep and Second Sweep recognize the endpoints of the output mask and their incoming and outgoing edges. Algorithm BOUNDARIES, described in Section 2.2 of the preceding chapter, uses these endpoints and outputs the polygonal circuit of the output mask in time $O(N + k')$, using memory $O(N + k')$. Thus, we finally have:

Theorem: Algorithm BOOLEAN MASK 2 runs in time $O(N h \log N + k')$ using memory $O(N \log h + k''h + k')$.

APPENDIX - CASE ANALYSIS OF THE INTERSECTION POINT.

This appendix is devoted to the complete listing of all the configurations of the intersection and union of two masks whose polygons intersect in a point P. The intersection point P, along with the two masks that define it, is illustrated in the first column of Table A.1. The second column illustrates the incoming and outgoing edges of P in the intersection mask, the third one the incoming and outgoing edges of P in the union mask.

Table A.1. Description of the intersection and union masks.

Description of the intersection point	Description of mask intersection	Description of mask union

Table A.1 (continued)

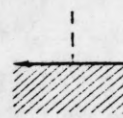
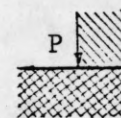
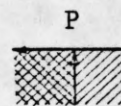
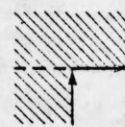
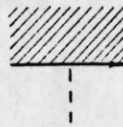
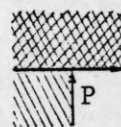
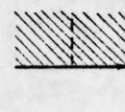
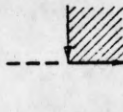
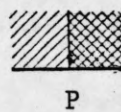
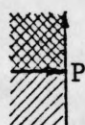
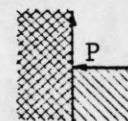
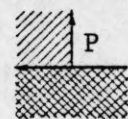
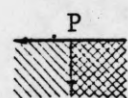
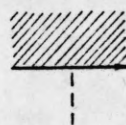
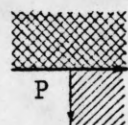
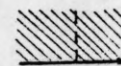
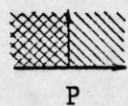
Description of the
intersection pointDescription of
mask intersectionDescription of
mask union

Table A.1 (continued)

Description of the
intersection pointDescription of
mask intersectionDescription of
mask union

REFERENCES

1. C. K. Yap, "Fast algorithms for boolean operations on rectilinear regions," Courant Institute, February, 1982.
2. U. Lauther, "An $O(N \log N)$ algorithm for boolean mask operations," Proc. 18th DA-Conf., Nashville, 1981.
3. H. S. Baird, "Design of a family of algorithms for large scale integrated circuit mask artwork analysis," M.S. Thesis, Dept. of Computer Science, Rutgers University, May 1976.
4. P. Losleben, K. Thompson, "Topological analysis for VLSI circuits," Proc. 16th DA-Conf., San Diego, 1979.
5. U. Lauther, "Simple but fast algorithms for connectivity extraction and comparison in cell based VLSI designs," Proc. ECCTD 80, Warsaw, 1980.
6. J. L. Bentley and T. Ottmann, "Algorithms for reporting and counting geometric intersections," IEEE Trans. Comput., vol. C-28, pp. 643-647, Sept. 1979.
7. W. J. Paul, R. E. Tarjan and J. R. Celoni, "Space bounds for a game on graphs," Eighth Annual Symp. on Theory of Computing, Hershey, PA, pp. 149-160, May 1976.
8. D. E. Knuth, The Art of Computer Programming, Vol. 3, Addison-Wesley, 1973.