SHORT NOTES

the left or right of the point. Each of the last two IF alternatives do this. The reason we must break the last case into two subcases is that if two polygons have some sides in common, the vertices need not occur in the same order (i.e. the order of their common vertices may be reversed). However, we want the same side and same point always to produce the same result when CROSS is called.

To determine if a ray from (x,Y) crosses a side from (xA,YA) to (xB,YB), given that YA ≠ YB, we can compute the X coordinate of the point where the side crosses a horizontal line through (X,Y) using the expression XA + (Y-B)(XA-YA)/(YB-YA). A crossing occurs if and only if X is less than this value. We know that XA ≠ XB, or one of the earlier cases would apply. We make certain that this test always produces the same result by requiring that XA < XB, and reversing the vertices if necessary.

If retrieval of information on the basis of location is rarely performed, or if only a small number of points is to be considered, then the above algorithm may be applied to each point in turn. If a large number of points is to be searched frequently, the points may be organized in a tree structure. A set of points and a rectangle which contains all of the points in the set are associated with each node of the tree. If the rectangle is entirely inside or outside the region of interest then so are all the points in the rectangle. Otherwise, we must consider a lower level node having smaller associate rectangles, and so forth, down to the level of individual points.

This approach easily generalizes to three (or more) dimensions, to regions with other than polygonal boundaries, and to other geometrical operations.

F. W. BURTON
School of Computing Studies
University of East Anglia
Norwich, NR4 7TJ
England

V. J. KOLLIAS
Athens Faculty of Agriculture
Laboratory of Soils and Agricultural Chemistry
Iera Odos, Votanikos
Athens, Greece

J. G. KOLLIAS
Department of Computer Sciences
National Technical University of Athens
9, Heroon Polytechnion Avenue
Zografou
Athens (621), Greece

† Part of this work was done while visiting Michigan Technological University.
‡ Present address: Department of Electrical Engineering and Computer Science, Box 104, University of Colorado at Denver, 1100 Fourteenth Street, Denver, Colorado 80202, USA.

References

4. S. Nordbeck and B. Rystedt, Computer cartography point in polygon points. BIT 7, 30-64 (1967).

Received April 1983

The Halting Problem Does Not Matter

Suppose that \( P \) is a program whose set of acceptable data \( \text{accept}(P) \) is recursive. Then there is another program \( P' \) with the same acceptable data, which cannot loop, and which behaves like \( P \) for all input data in \( \text{accept}(P) \).

A key measure of program quality is robustness. A program is said to be robust if it will behave sensibly, whatever its input. In particular, it should never under any circumstance go into an infinite loop. There are certain conceivable kinds of data which can only be distinguished and accepted by programs which might loop. Such a set of possible data is called recursively enumerable but not recursive. (This is the terminology of Ref. 1.) If anybody can construct a set of conceivable data which is not even recursively enumerable, then he has discovered a counterexample to Church's thesis. In practice, all useful programs accept a set of data which is not only recursively enumerable, but also actually recursive.

Proposition

Suppose that \( P \) is a program whose set of acceptable data \( \text{accept}(P) \) is recursive. Then there is another program \( P' \) with the same acceptable data:

\[
\text{accept}(P') = \text{accept}(P)
\]

and which cannot loop:

\[
\text{loop}(P') = \text{empty}
\]

and which behaves like \( P \) for all input data in \( \text{accept}(P) \):

\[
\text{if } P \text{ and } P' \text{ are both given the same word } w \text{ from } \text{accept}(P) \text{ as initial datum, then their outputs will be the same.}
\]

Proof

Throughout what follows, all programs will be regarded as Turing machines. The output of a program is the content of its tape on termination. The key assumption is that \( \text{accept}(P) \) is recursive. This means that there is a program \( Q \), and

\[
\text{accept}(Q) = \text{accept}(P)
\]

and \( Q \) cannot loop. The program \( P' \) is constructed from \( P \) and \( Q \).

Let \( P' \) have some input datum \( w \). This datum is a sequence of symbols stored at the left hand end of the machine's tape. First, \( P' \) copies this word one position to the right on the tape, and inserts a special character (say \( E \), not recognized by \( P \)) before it. Secondly, it inserts another special symbol (say \( F \), not recognized by \( P \) or \( Q \)) after the word, and makes a new copy of \( w \) to the right of \( F \). At the far end of this copy, \( P' \) inserts a third special symbol (say \( G \), not recognized by \( Q \)).

If \( w \) is the sequence

\[
w_1, w_2, w_3, \ldots, w_n
\]

then by this stage the tape appears as

\[
E \ w_1 \ w_2 \ \ldots \ w_n \ F \ w_1 \ w_2 \ \ldots \ w_n \ G
\]

Thirdly, the tape head is moved to the tape cell to the right of the one holding \( F \).

Next, \( P' \) performs all the operations of \( Q \). Just one modification is required in \( Q \). Whenever the tape head is situated over a cell containing \( G \), \( P' \) will

(a) over-write this cell with a blank character, and move the head to the right
(b) write \( G \), and move the head back left.

During this stage, the input will be rejected if and only if the original datum \( w \) would be rejected by \( Q \).

Where \( Q \) would halt, \( P' \) contains a subroutine which restores the tape to its initial state. This is possible because \( E \) and \( F \) still delimit a copy of \( w \), and \( G \) marks the extent of tape which has been over-written. At the end of the subroutine, which cannot fail or loop, \( P' \) enters the original program \( P \).

Q.E.D.

In practice, programmers worthy of the name always write their code on these lines. The halting problem does not concern them. It should be seen for what it is: a profound feature of mathematics, and a curiosity in the history of computing.

ALAN HUTCHINSON
Department of Computation
U. M. I. S. T.
Sackville Street
Manchester
UK

Reference