Application: Transitive Closures in Medicine and Engineering

Transitive and reflexive closures are especially important in computer science. For example, suppose computers are connected to each other in a network, with each computer connected directly to a small number of other computers. Information can be passed directly from one computer to another over a connection between them. The transitive and reflexive closure of *IsConnectedTo* is *CanAccess*. This relation gives the limit of how far information from one machine may be passed along to others. The examples that follow show how the transitive closure idea leads to better understanding in fields as diverse as medicine and chip testing.

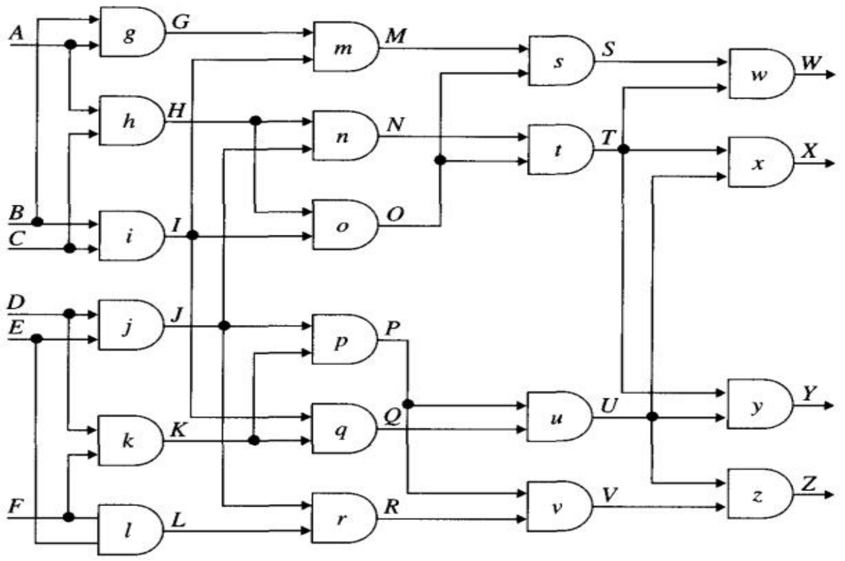
Artificial Intelligence

Many artificial intelligence applications can be phrased in terms of some (simulated) person making inferences based on some initial data. One kind of application is the expert system, in which designers try to encode the knowledge that an expert would use in approaching a problem. Suppose, for example, an expert system is used to suggest to a physician certain tests that should be run. The system might say, for example, that if the patient's weight is more than 25 percent over the recommended level to check for high cholesterol. (Drs. X, Y, and Z all told the designers of the expert system that is what they do, so it must be a reasonable rule.) And if the patient eats a high-fat diet, there should be a check for cholesterol. (Drs. X, W, and Q all said they do that.) And if there is a check of the cholesterol level, there should also be a check for high triglycerides (suggested by several other doctors.) If there is a test for triglycerides, there should also be a test for something else, and so on. This series of "rules" is stored in the program called the *expert system*. The doctor enters that the patient has a body weight 30 percent over his recommended weight and this fact triggers a series of inferences: check cholesterol level; check triglycerides level, and so on. This is a transitive closure operation: including one test triggered including another, which triggered including another, . . . , until nothing else was triggered.

Often, the rules are rather more complicated, such as "if the patient's weight is 15 percent over the recommended weight *and* the patient is diabetic, then do a cholesterol test." This is a more complicated sort of closure operation, but the idea is similar.

Testing Circuits

Here, we picture a combinational electric circuit:



Current flows from left to right, so there are six input lines, A through F, and four output lines, W through Z. There are 20 gates, g through z. For convenience, we picture them all as and-gates, but the intention is that they might implement some AND gates, some OR gates, and some NOT gates. Define two relations between lines and gates, one "saying" that a line is an input to a gate and the other that a line is an output of a gate. The large dots indicate that a line is split, being an input for several gates, such as A is

Input	
A	g
A	h
B	g
B	i
C	h
C	i
D	j
D	k
⋮	⋮

Output	
g	G
h	H
i	I
j	J
k	K
l	L
m	M
n	N
o	O
⋮	⋮

input for both gates g and h. Otherwise, when two lines cross, such as the output line of h and the output line of i, it just means that when the circuit is fabricated, these two lines will follow this path but will not touch.

The circuit manufacturer would want to check that each gate is functioning correctly. For example, if all the lines carry 0's and 1's (designers use 1 and 0 instead of *TRUE* and *FALSE*), gate o might be "stuck at 0", that is, it might always output a 0, no matter what its input is. The manufacturer would then like to have a "test vector" for that: a set of inputs to distinguish whether gate o is stuck at 0. The first part of choosing such a test vector is to determine which output lines could be affected if gate o is malfunctioning. In this case, lines W, X, and Y could be affected. Line Z cannot be, however, since no output from gate o flows, directly or indirectly, into gate z.

The relation of one line directly influencing another is *Output* $\circ$ *Input.* The relation of directly or indirectly influencing another line—through *any number* of intermediate lines—is thus (*Output* $\circ$ *Input*)*. The question above is to find all output lines where (*o, some output line*) $\in$ (*Output* $\circ$ *Input*)* $\circ$ *Output.*

Of course, now that designers have narrowed down which lines might be affected by a malfunction at gate o, they must go on to determine how to produce a single input that will identify the stuck-at-0 fault. However, we cannot do that without knowing what the individual gates are.