Many Variable Fault Isolation: The Boolean Exposition of a Classic Interview Problem

Travis Ayres
trayres@gmail.com

September 6, 2015

Abstract

This paper arose as a result of a question during a technical interview, and is a classic: "You are given a faulty system composed of three parts; you can replace any item in the system with a known working part. Which component is faulty?" However, a particularly vexing take on this classic question is to present the candidate with logically inconsistent test results in order to see how the candidate thinks; wishing to avoid this version of the question, we encode the system as a Boolean equation and solved for the faulty component. This paper discusses the basic question as a demonstration, generalizes the concept to any number of components and multiple faults, and includes a discussion of the results.

The Problem Statement

A system composed of N components $C_1, C_2, C_3, \ldots, C_{N-1}, C_N$ has a fault and is inoperable. Each component can be replaced with a known working substitute; when all faulty components are replaced, the system operates correctly. Find the faulty component(s).

Encoding the Problem

The system is comprised of components that are either original or replacements and the result is that the system operates or not; we use a Boolean functional mapping of input state and output state as follows: Represent the system under consideration as a state vector of the format $C_1, C_2, \ldots, C_N$ with 0 (False) indicating an original component and 1 (True) representing a swapped component. For the output, let us choose 0 to encode a faulty (or inoperable) system, and 1 to encode a correctly functioning (operable) system.

An Example: The Single-Fault, 3-Component System

To motivate the discussion, consider the single-fault, 3-component system. By construction we know the state variable for the original system with no swapped components is $0, 0, 0$, and we are given that the output is 0. Let's consider the case where the third component, C_3, is faulty. The truth table with the encoding above and the givens is:

C_1	C_2	C_3	*Result*	
0	0	0	**0**	← By construction
0	0	1	**1**	
0	1	0	**0**	
0	1	1	**1**	
1	0	0	**0**	
1	0	1	**1**	
1	1	0	**0**	
1	1	1	**1**	

Writing this in canonical Sum of Products (SOP) format, we have the equation:

$$C_1'C_2'C_3 + C_1'C_2C_3 + C_1C_2'C_3 + C_1C_2C_3$$

By factoring terms, we immediately see an interesting pattern emerge:

$$(C_1'C_2' + C_1'C_2 + C_1C_2' + C_1C_2)C_3$$

$$(C_1'(C_2' + C_2) + C_1(C_2' + C_2))C_3$$

This can be reduced to the term of the component that caused the system to be inoperable (C_3) but more importantly, it leads to an intuitive understanding of the equations.

The Intuitive Interpretation

Looking back at the Truth Table, we see that the equation reduces to exactly the component we need to replace; stated another way, common to all functioning units is the replacement of the malfunctioning component! We can go one step farther: a component that is already working can be replaced or not replaced; as both components are functioning, swapping them makes no difference. Components that are initially faulty, however, MUST be replaced for the system to operate, so they appear without their complemented form in the system equation, and will not reduce out.

Generalization to Arbitrary Number of Components and Faults

For a system to function, each component must be functional; in equation form, we have $C_1 \wedge C_2 \wedge \ldots C_{N-1} \wedge C_N$. Components that were already functional can be replaced or not: they show up in the final equation as $(C_M + C'_M)$, which is why only the faulty items are left in the final equation. We see how this generalizes to an arbitrary number of components and faults!

Assume we have a system composed of parts $C_0, C_1, C_2, \ldots, C_{N-1}, C_N$. Adding a component C_{N+1} to the system leaves us with two possibilities: C_{N+1} is not faulty, and therefore does not effect the system equation, or it is faulty, must be replaced, and ends up in the system equation. We see from this that we can add any number of components to the system and the final system equation will determine if they are faulty.

As a side note, we mention that items that don't effect the functionality of the system (say, its color) will not show up in the system equation, unless specifically encoded for (a component painted incorrectly must be replaced).

Conclusion

We've taken a classic interview question and provided some discussion of how to encode and solve for the faulty component, and seen how this generalizes to an arbitrary number of functional components and multiple faults. I hope this overview was in some way useful or interesting; if it can be improved in any way, please email me and I'll update it. Thanks for reading!