Delta Debugging

Delta debugging is an extremely useful algorithm, and good delta debugging tools are practically required in order to do meaningful random testing on complex systems. Unfortunately, traditional delta debugging reduces tests too much for any goal not related to simple success or failure, discarding all behavior not related to the failure of a test. Even more critically, traditional delta debugging cannot be applied to successful tests at all. The assumption has always been that delta debugging is a debugging algorithm, only useful for reducing failures. However, the best way to understand ddmin-like algorithms is that they reduce the size of a generic cause (e.g. a test case, a thread schedule, etc.) while ensuring that it still causes some fixed effect related to the essence of a test case: ddmin is a special-case of cause reduction.‡ The core value of delta debugging can be understood in a simple proposition: given two test cases that achieve the same purpose, the smaller of the two test cases will typically execute more quickly and be easier to understand and mechanically analyze. Delta debugging is valuable because, given two test cases that both serve the same purpose, the smaller of the two is almost always preferable. There is no reason why the essence of a test case should be limited to fault detection.

For developers, the occurrence of a scenario similar to this happens all too often: a developer codes a piece of functionality that works. Then, over a period, multiple changes to the program source files are made and that piece of functionality stops working. The cause of the regression could be any of the changes that were made since the time the functionality was last known to work.

To isolate the cause of the regression, the developer begins the debugging process. Generally, debugging is a manual process where the developer must walk through the code while trying to keep track of variables and function calls. Sure, there are debuggers that can help you keep track of variables, the call stack, watch certain blocks of code, and execute the code step by step, however, debugging is still mainly a manual process.

Based on our experience using the Delta Debugging. This helped us optimize the debugging of some unknown complex source code easier. Reducing failure inducing inputs helps us understand how the inputs affect the program. We initially struggled to work with delta debugger. The concept was clear that it was using binary search as a core logic to divide the program into smaller parts and solving the problem is much easier in terms of complexity.