was applied. Thus, there are no changing signals (events) shown. To each good-gate, a number of bad-gates are attached in a linked-list structure (shown as tiny wiggly arrows.) Attached to the good AND gate are four bad-gates, a_0, b_0, c_0, *and* e_0. *Notice that at least one value around a bad-gate differs from the good-gate and the difference is caused by the corresponding fault. Fault lists for the other two gates are also shown in the figure. At the primary output g, any bad-gate whose output differs from that of the good-gate indicates fault detection. Thus, faults* a_0, c_0, e_0, *and* g_0 *are detected. With fault-dropping, we would have removed these faults from further consideration. In this exercise, however, we will not drop faults.*

Notice that our fault detection result agrees with that obtained by deductive fault simulation in Figure 5.18. But the fault lists there were shorter. In the deductive simulator the fault-list is for a signal and contains only the faults that affect (or are detected at) that signal. In the concurrent simulator the fault-list is for a gate and even faults that affect the inputs of that gate are included in the list. Fault-lists in a concurrent simulator are, therefore, comparatively longer. The advantage, though not as clear in logic simulation, is significant when more complex functional modules (memories and RTL or behavioral models) are simulated.

In Figure 5.20, we simulate a 1 to 0 ($1 \rightarrow 0$) *good-event at a. Examine the changes shown in the AND good-gate and its fault-list (associated bad-gates.) The top input of all except the* a_0 *bad-gate change. Only the good-gate output changes, producing a* $1 \rightarrow 0$ *event on signal e. After these evaluations, bad-gates* a_0 *and* e_0 *have identical signal values as the good-gate and hence they converge. They are removed from the fault-list. At this point, one good-event* $1 \rightarrow 0$ *on e and no bad-events have been generated. The OR gate is evaluated and produces a* $1 \rightarrow 0$ *good-event on g. Bad-gates are also evaluated but none generates any bad-event. After evaluation, bad-gates* a_0, c_0, e_0 *and* g_0 *converge to the good OR gate. These are removed from the fault-list. However, the processing of the* ($1 \rightarrow 0$) *good-event at a is not complete.*

As Figure 5.21 shows, changing of signal a activates fault a_1. *Thus, a diverging bad-gate labeled* a_1 *is inserted in the fault-list of the AND gate. Newly diverging gates are shown with lighter grey shading in Figure 5.21. Similarly, another bad-gate* e_1 *is also added. These two generate bad-events, which when processed at the OR gate produce further divergence of bad-gates* a_1 *and* e_1 *there. The change caused by the good-event at the OR gate, discussed above, results in the divergence of another bad-gate* g_1. *This completes the simulation.*

All bad-gates at the output g have a different output value than that of the good-gate. Therefore, detected faults are b_0, d_0, f_1, a_1, e_1, *and* g_1.

Our example illustrates only some features of the concurrent fault simulation algorithm. Useful techniques such as *multi-list traversal* (MLT) allow simulation of multiple output functions that may also have internal states. An interested reader should study the book by Ulrich *et al.* [684] to learn about the complete capabilities of this algorithm. Its significant advantages are efficiency (elimination of redundant computation) and modeling flexibility (fault simulation for anything that can be simulated.) Some notable concurrent fault simulators are the MARS