Simulating Error Propagation

Knowledge of error propagation results in safer software!

What is error propagation?

To some extent all software systems contains faults. Such problems can be alleviated by controlling the possible effects of a fault being executed. This is a typical method for creating safety critical systems. Usually, when a fault is executed there is a local effect which is of little consequence on a system level. If not dealt with immediatly, such a fault can propagate(spread) throughout the system and cause critical failure. Ensuring that failures such as these are properly isolated, and not allowed to propagate, is essential when developing critical systems.



It takes one domino to trip the rest. Source: jupiterimages

Why is it important?

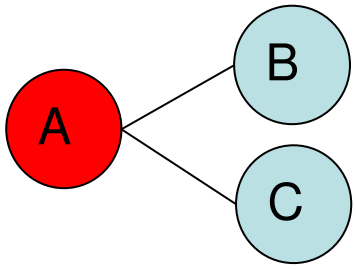
All around us we find technology dependant on safety critical software. Everything from nuclear powerplants and industry to planes and satelittes are controlled by software. If not for safety critical and dependable software the world would come to a halt. In worst case lifes would be lost.



Fancy one of these having a meltdown? Source: lightandmatter

Simulating errors

For our purposes a generic approach to error simulation is the best solution. It is both difficult and unecessary to simulate every error down to the last detail. Instead each error is designated a certain probability. All components of the software system we simulate can have several possible error modes, each with a prefixed chance of occuring. The system layout describes the connections between the components and injecting an error in one part of the system enables us to trace if, how and where the error propagates.



To properly prove the simulator we describe the system as a Markov system. At any given time ***t***, the state of component **A** can be derived from the states of all the other components in the system. Furthermore, the whole system can be described as the states of all components at a specified time. Now, we can deal with one and one component in the system at any given point in time. Using discrete time allows handling of one and one component. For instance, a fault has occurred in **A** (on the left). When deciding the state of **B** we have the knowledge of both **A** (which has occurred and was erroneous) and **C** (which has not yet occurred). Thus, it is possible to calculate the state of each component in the system, trace errors propagating through the system and find the state of the entire system.

This is a brief description of John E Simensen master thesis. It was supervised by associate professor R. Winther and Ph.D. student Monica Kristiansen. A part of the thesis was presented at Esrel 2007 in Stavanger, Norway.