TAYLOR83

Taylor, Richard N., "A General-Purpose Algorithm for Analyzing Concurrent Programs", Communications of the A.C.M., Volume 26, Number 5, May 1983, pp. 362-376. Verification and testing of software is an important component in the development of any software system. If the software involves concurrent applications, the verification process becomes even more difficult. Some of the major problems which must be addressed are:

ensuring the system will not deadlock ensuring no undesired parallelism occurs. Richard Taylor, in this paper, proposes an algorithm for verifying concurrent applications with respect to the aforementioned problems and others. Any analysis of concurrent programs must be concerned with the syncronization structures of a program and to this end, the purpose of Taylor's algorithm may be stated in one of two ways: to determine all possible concurrency states and how they relate to one another OR to generate the complete concurrency history of the program. In doing this, all possible rendezvous are determined, all possible infinite waits are detected and parallel action information is obtained. The algorithm uses the approach of static analysis to determine the concurrency histories (i.e., analysis is performed on a model of the program without requiring test executions). As with any analysis tool, this algorithm must satisfy some basic characteristics (characteristics which determine the tools utility). As listed by Taylor, these are accuracy of the results: no potential error situation may be missed by the algorithm, adequate and sufficient result representation: the user must be able to decide, by looking at the results, whether or not the report actually indicates a real software error or is just spurious efficiency. Unfortunately, an inherent problem with all static analysis approaches and hence with Taylor's algorithm is that it has difficulty dealing with subscripted structures. This is due to the fact that the analysis program is incapable of determining the particular identity of an object (eg. difficulty with differentiating between elements of an array: this is crucial in many situations where array elements may be used as part of a rendezvous, etc.). It can be said that deadlock detection may be the most important requirement of this type of algorithm. The algorithm proposed by Taylor actually takes this a step farther with the ability to detect infinite waits that may arise; this being the detection of situations where a task may become permanently blocked (not necessarily deadlock, but it does include the deadlock case). The input requirements for Taylor's algorithm are an elegant product of the compilation process and are: program call graph: indicates subprogram invocation structure, program scope information: indicates nesting or hierarchical structure of a programs constituents, annotated program flowgraph: mark each node indicating the type of statement the node represents, the program objects the statement contains, etc..

Detailed discussions are presented in how each of the above input components is used in the program analysis. The algorithm itself is not totally acceptable with respect to execution time. It appears, because of all that it attempts to analyze, that the algorithm is exponential, although for certain classes, the complexity is more efficient. Taylor does, however, explore various techniques for speeding up the analysis of some programs. Overall, the algorithm can be regarded as a success. It does provide answers to the analysis problems mentioned earlier and the answers provided are as accurate as can be obtained from static tools without resorting to symbolic execution techniques. Concurrency is an important issue in any language design and any research into the development of programs such as Richard Taylor's is definitely important. The results presented by Taylor can unfortunately only be applied to systems using the rendezvous mechanism and further research into other algorithms or investigating the possibility of extrapolating Taylor's algorithm to other approaches is needed.

Possible bibliography entries for this article include:

1. concurrency