Gaining Confidence in Distributed Systems

Gleb Naumovich,
Lori A. Clarke,
and Leon J. Osterweil
University of Massachusetts, Amherst
Computer Science Department
University of Massachusetts
Amherst, Massachusetts 01003
(413) 545-2013
{naumovic|clarke|ljo}@cs.umass.edu

Matthew B. Dwyer
Kansas State University, Manhattan
Department of Computing and
Information Sciences
Kansas State University
Manhattan, KS 66506
(913) 532-6350
dwyer@cis.ksu.edu

Introduction

Increasingly software systems are being written as distributed systems. The advent of Java and the World Wide Web will no doubt accelerate this trend. Distributed systems have many benefits, such as improved performance and flexibility. Unfortunately, it is much more difficult to test such systems. Distributed systems are inherently nondeterministic; executing the same program on the same inputs might produce different results depending on such things as the system load and scheduling algorithm. There has been considerable work on developing testing platforms for distributed systems, which primarily monitor behavior and support re-execution or post-mortem examination of intermediate results. As with all testing approaches, the results are only as good as the selected test data. The next input may expose an error in the system.

An alternative approach is to verify that certain properties must be guaranteed to hold for **any** execution of the system. This approach is independent of the choice of test data or any of the factors that could affect execution, such as the system load. Formal verification techniques have not been widely successful in the past primarily because they require a complete specification of the behavior, which is usually very difficult to write and often as prone to errors as the code itself. In addition, formal verification approaches are computationally expensive, and they also require considerable amount of expertise on the part of the user. We propose an alternative approach, which uses data flow analysis to verify, automatically and efficiently, user-specified properties of distributed systems.

With this approach, properties are usually not a full specification of the system's expected behavior but instead describe important requirements of the system, such as the elevator door can not open while the elevator is moving. Thus, instead of a test case which would exercise one path to show that the door can not be open when the elevator is moving, a property that states that this should not happen on any execution of the system is formulated. If it is verified, it is proven that no execution could cause the door to be open while the elevator is moving. If it is not verified, then a counter example is given which provides a trace through the program where the door is open while the elevator is moving.

This approach can be used throughout the software development lifecycle. Properties describing desirable behavior can be formulated as early as on the requirements stage. After the design of the system is completed, this approach can be used to verify whether the design conforms to the behaviors described by the set of properties. This allows errors to be detected early, rather than discovering them later during testing or even after the system has been deployed. This process can be repeated after the coding stage, enhancing the initial set of properties with additional ones reflecting the specific design and implementation choices.

Each property is proved separately so properties can be added to or removed from the property set as the system changes or more information is acquired. For example, if a system fails during execution, the analyst may have a hunch about the cause of that failure and can formulate a property to validate that hunch. If the property can be verified, then the analyst must consider alternative hunches. If the property is shown to be false, confirming the hunch, then the analyst can look at some counter examples, to help pinpoint the cause of the failure. After the analyst has "fixed" the system, the property should again be verified to assure that the analyst has found all the causes of the failure.

Using data flow analysis to statically detect errors in software systems was first proposed in the DAVE system [5], which could detect anomalous sequences of variable definitions and references in sequential FORTRAN programs. The CESAR system [6] extended this approach to general user-specified properties. The FLAVERS system (**FL**ow **A**nalysis for **VER**ifying **S**pecifications), which is the system described here, extends this approach to be applicable to distributed systems and provides a means for improving the accuracy of the results. FLAVERS has been used to verify properties of the benchmark programs that appear in the concurrency analysis literature [7], properties of network protocols [9] and high-level software architectures [8], and by industry to prove properties about Advanced Distributed Simulation software.

Technical Overview

FLAVERS [1] can be used to verify explicitly stated event sequence properties of concurrent or distributed systems. With this approach, an analyst defines a set of program events of interest and formulates properties to be checked as sequences of those events. Given a design or implementation of a concurrent or distributed system, annotated with these events, and a property to which that system should adhere, the technique determines whether the property holds on all system executions.

The results of an analysis may indicate that the property holds on **all** executions of the system, **no** executions of the system, or **some** executions of the system. The first two results are called *conclusive*, while the latter is called *inconclusive*. One of the reasons for an inconclusive result is that an error has been found, and thus the property does not hold on some executions of the system. Another reason, however, could be the potential imprecision of the model of the system that was used during the analysis. As is typically the case with static analyses, to assure that the results are always conservative, the model of the system overestimates the possible executable behaviors of the system. If the results of an analysis are inconclusive because the property holds on all real executions of the system but does not hold on some infeasible executions of the model, we call such results *spurious*.

The accuracy improving approach of FLAVERS allows the analyst to add control and data flow information incrementally to the analysis to increase its precision. The information is added in the form of *constraints*, represented as finite state automata. For example, the analyst might decide that the results are inconclusive because the value of some variable is used as a sentinel in a conditional statement in the software system under analysis, and this conditional statement affects the precision of the results. Using the automated tools provided, the analyst can build an automaton representing the values of that variable. Another example of a constraint that is often necessary to increase the accuracy of the analysis of a distributed software is an automaton that represents the control flow of a single process in the distributed system.

After new constraints are constructed, they are incorporated in the next run of the analysis. Figure 1 illustrates the high-level structure of the accuracy improving data flow approach of FLAVERS. The dashed line from the inconclusive results to the set of constraint automata for the system shows the process of using previous analysis results to guide the selection of new constraints for improving the accuracy of the results.

To analyze non-trivial properties of concurrent systems, the analyst may need to go through the process of adding constraints several times; each time getting inconclusive results, realizing the cause of the inconclusiveness, and incorporating additional relevant information into the analysis. The advantage of this technique is that each attempt is relatively efficient and straightforward and, thus, an analyst can prove interesting properties easily and efficiently, compared to other techniques such as theorem proving and reachability analysis. Descriptions of the application of this technique appear in [1] and [9].

In addition to improving the analysis accuracy, the constraint mechanism can be used in other ways,

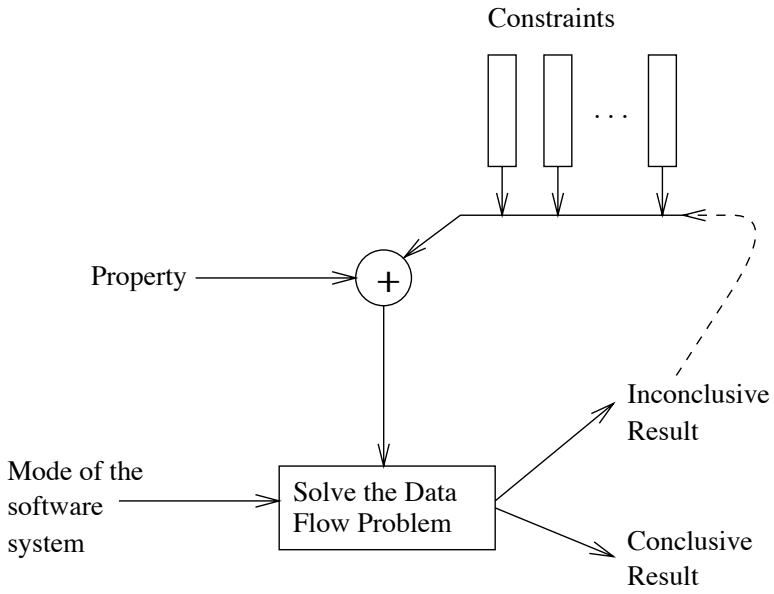

Figure 1: Accuracy Improving Data Flow Analysis

such as specifying behaviors of missing pieces of software and modeling the environment in which the system executes. Modeling behaviors of some not yet created parts of the system in the form of constraints allows the analysis process to start earlier, without waiting until all components of the system are built. Constraints mechanism can also be used to specify assumptions about the environment (e.g. hardware or human interactions) in which the system executes. For example, the behavior of a communication link connecting network entities can be modeled to provide certain restrictions that the hardware imposes, but which cannot be obtained from the software. For example, an assumption can be made that the link cannot lose more than a predefined fixed number of messages in a row [9].

Theoretical Basis

The incremental accuracy improving technique models the system under analysis with a *Trace Flow Graph (TFG)*, whose nodes are annotated with events of interest. The TFG represents the flow of information through the system. Interleavings of events located in different tasks are represented explicitly with a special kind of edge. Both properties and constraints are represented as *Finite State Automata (FSA)* over the set of the events of interest. During data flow analysis a set of state tuples is associated with each node of the TFG, where a tuple contains one state from each of the FSAs. A state tuple associated with a TFG node means that there is a trace through the TFG to this node that is constrained by the set of FSAs so that the FSAs are in their associated state when this node is reached. Data flow equations describe the appropriate propagation of the FSA state information through the nodes of the TFG. These equations can be defined so that the solution always converges to the Maximal Fixed Point [4].

Once the Maximal Fixed Point solution is obtained, the information about the states of the property automaton is extracted from the unique final node of the TFG graph, which represents the termination of the software system. If only accepting states of the property automaton are observed in the final node, then this implies that the property holds on **all** executions of the system. If no accepting states of the property automaton are observed, this implies that the property holds on **no** executions of the system. Finally, if both accepting and non-accepting states of the property automaton are present in the final node, this implies that the analysis is inconclusive, either because the property really holds on some but not other executions of the system, or because of spurious paths through the TFG. These paths are selectively shown to the analyst, who must decide if they represent a real error in the system or if more information must be provided via

constraints to eliminate these spurious paths in subsequent analysis runs.

Tool Summary

FLAVERS consists of a number of tools for creating the artifacts necessary for the analysis, for running the analysis, and for examining the results of the analysis. A graphic user interface guides the user through the various stages of the analysis.

Before the analysis can be run, the software system has to be annotated with events of interest and the property (or properties) to be verified must be specified. FLAVERS employs automated language processing tools [2, 3] to build an abstract syntax tree (AST) for the system and a control flow graph (CFG) for each task in the system. Both ASTs and CFGs are used to create the TFG, the model of the system that is used during the data flow analysis. Properties are specified in the form of quantified regular expressions (QREs) [1]. This specification consists of the alphabet of the property, which is a set of the events that are referenced in the property, an indicator of whether the property should hold on all or on no executions of the system, and a regular expression that describes the event sequences. After the analyst writes a property in the QRE language, it is submitted to FLAVERS for a syntactic check and a translation into a FSA form. All the internal artifacts, e.g., ASTs, CFGs, TFG, and FSAs, are available for viewing in a graphical form, although analysts do not have to be aware of them to be able to analyze software systems effectively. FLAVERS includes a number of automated tools to facilitate building constraints. Currently the automated creation of *Task Automata*, which model all possible orders of events allowed by the control flow in a single task, and *Variable Automata*, which model the execution behavior of selected variables in the system, are supported.

To assist the analyst in determining the cause of an inconclusive result, FLAVERS displays example traces through the system on which the property is violated. The analyst has to decide whether these executions are real, which means that the property really does not hold on some executions of the system, or whether they are infeasible. Usually, given an infeasible execution, it is easy to understand why this execution cannot occur in practice and to come up with the necessary constraints that forbid this execution, and perhaps related sets of executions, from being explored by the data flow analysis in subsequent runs.

FLAVERS is a promising tool for static analysis of distributed systems. It has been shown to be successful in proving properties of well-known benchmarks programs commonly used by the concurrent static analysis community, such as the Dining Philosophers, Readers-Writers, and Gas Station problems. In addition there has been some positive industrial experience applying FLAVERS to real software systems. Our belief that FLAVERS is scalable and thus applicable to large-scale, distributed software applications is based on the observation that interesting and important properties often use only a handful of events in a software system. Once these events are known the TFG model can be reduced significantly. Work on applying FLAVERS to large-scale distributed software is currently underway. As distributed systems become more prevalent, tools such as FLAVERS will be imperative for reasoning about these systems and helping to assurance their reliability.

References

[1] Matthew Dwyer and Lori Clarke. Data Flow Analysis for Verifying Properties of Concurrent Programs. In *ACM SIGSOFT'94 Software Engineering Notes, Proc. Second ACM SIGSOFT Symposium on Foundations of Software Engineering*, pages 62–75, v. 19, n. 5, 1994.

[2] Peri L. Tarr. Language Processing Toolset Prerelease Notes. *Arcadia Document, UM–91–01, University of Massachusetts, 1991.*

[3] Peri L. Tarr and Lori A. Clarke. PLEIADES: An Object Management System for Software Engineering Environments. In *Proc. of the First ACM SIGSOFT Symposium on Foundations of Software Engineering*, pages 56–70, Los Angeles, CA, December 1993.

[4] M. S. Hecht. Flow Analysis of Computer Programs. North-Holland, New York, 1977.

[5] Leon J. Osterweil and L.D. Fosdick. DAVE — A Validation, Error Detection, and Documentation System for Fortran Programs. In *Software Practice and Experience*, pages 473–486, v. 6, n. 4, October 1976.

[6] Kurt M. Olender and Leon J. Osterweil. Interprocedural Static Analysis of Sequencing Constraints. *ACM Transactions on Software Engineering and Methodology*, pages 21–52, v. 1, n. 1, January 1992.

[7] A.T. Chamillard, Lori A. Clarke, and George A. Avrunin. Experimental Design for Comparing Static Concurrency Analysis Techniques. Technical Report 96–84, Department of Computer and Information Science, University of Massachusetts, Amherst, MA 01002, August 1996.

[8] Gleb Naumovich, George A. Avrunin, Lori A. Clarke, and Leon J. Osterweil. Applying Static Analysis to Software Architectures Technical Report 97–08, Department of Computer and Information Science, University of Massachusetts, Amherst, MA 01002, January 1997.

[9] Gleb Naumovich, Lori A. Clarke, and Leon J. Osterweil. Verification of Communication Protocols Using Data Flow Analysis. In *Proceedings of SIGSOFT'96: Fourth Symposium on the Foundations of Software Engineering*, pages 93–105, San Francisco CA, October 1996.