An overview of techniques aimed at automatically generating oracles from tests[*]

Luke Chircop[1], Christian Colombo[1], Adrian Francalanza[1], Mark Micallef[1], and Gordon J Pace[1]

Department of Computer Science, University of Malta

While significant testing is carried out throughout software development lifecycles, given the size and complexity of modern software systems, the possibility of post-deployment bugs remains a real concern. Although users would typically report any encountered bugs themselves, such bugs could tarnish the company's reputation and expose vulnerabilities within the system which could be exploited, potentially costing the company a fortune.

One way of reducing this risk is to include monitors with the running software such that any occurring bugs are immediately detected, reported, and potentially mitigated automatically. A significant problem with such a technique, however, is that existing tools require the software engineers to specify the properties to be checked in a formal language. This typically involves training personnel to use the tool and the language, as well as having to dedicate extra time to write the properties themselves. This adds up to the already expensive development process that software companies are constantly trying to optimize hence deterring the interest in including the use of runtime monitoring.

The use of monitors at runtime can be of great benefit to software companies. Therefore, we set out to identify how this approach can be packaged in a manner that makes it more viable for companies to consider using it. During the development of software, a collection of tests are usually defined and executed to assert that within their confines and inputs, the software operates correctly. All of these tests contain valuable information regarding how software is expected to behave. Given this readily available source of knowledge, we asked ourselves whether a technique could be devised to extract this information and automatically generate runtime monitors that can be deployed at runtime.

Proposed solution

Extracting valuable information from existing tests is not as trivial as one might think. Depending on the tests that are considered, there might be situations were not all of the required information is statically available. In such cases, the tests would need to be exercised to be able to log system traces which can then be used to infer models of good and expected behaviour. These models can then be used to generate monitors that verify that behaviour. To be able to extract information from tests and automatically generate monitors, a five step process is currently being considered:

1. The system and set of available tests are gathered and the points of interest are identified. These points of interest include method calls/returns, computational steps and will be used to gather behavioural characteristics of the system being tested.
2. The points of interest identified are then instrumented with loggers capturing behaviour and system state.
3. The instrumented system is then executed with the provided tests allowing the loggers to dump traces of observed execution.

[*] Project GOMTA financed by the Malta Council for Science & Technology through the National Research & Innovation Programme 2013

4. The dumped traces are then to be processed to generate models of expected good behaviour. Doing so may require two models one modelling the possible execution flow that the software can take and the other stating what the state of the software at particular instances should be.
5. The inferred models are then used to automatically generate runtime monitors able to verify such behaviour at runtime.

State of the Art

While to the best our knowledge there is very little work that attempts to generate monitors from tests automatically, a related body of work was identified that attempts to extract models from regression tests to pinpoint the probable causes of regression test failures in upgraded versions. This can be observed in [1–5], where a three stage technique is used.

Since in regression testing it is assumed that tests should pass when run in both the base and upgraded version of a program, the first step tries to pinpoint the probable cause of failure, by comparing the two versions to identify the points that may be influenced by the changes (taking into consideration the changed functions and neighbouring functions that reference them) based on those assumptions. Loggers are then instrumented inside the base version and the test cases are executed to dump system traces which include the state of local and global variables at every point of interest and also the sequence of events executed. These trace dumps, are then processed to generate two types of models; variable assertions representing variable state at each point and finite state automata models representing expected behaviour.

Once these models are inferred, the final stage kicks in. Here, the upgraded version is instrumented with the same loggers and the tests that failed are executed to dump event traces. These traces are then run through with the inferred models to pinpoint the probable cause of failure.

Conclusion

The related work that has been presented is of great interest to us since it provides a process that is able to extract information from existing tests and automatically generate models of expected behaviour. A challenge that is being foreseen if such a process is to be considered revolves around the fact that since our aim is to monitor live system executions, the upgraded version would not exist for the process to identify possible locations of interest. Instead, the tests would need to be executed whilst a logger keeps track of all the events that executed. Another factor that has to be taken into consideration, is the fact that there can be cases of characteristic tests (e.g. boundary tests) that when modelled might not include all the possible inputs accepted by a function which could result in a number of false positives being reported.

References

1. A. Babenko, L. Mariani, and F. Pastore. Ava: Automated interpretation of dynamically detected anomalies. In *Proceedings of the Eighteenth International Symposium on Software Testing and Analysis*, ISSTA '09, pages 237–248, New York, NY, USA, 2009. ACM.
2. L. Mariani and F. Pastore. Automated identification of failure causes in system logs. In *Software Reliability Engineering, 2008. ISSRE 2008. 19th International Symposium on*, pages 117–126, Nov 2008.
3. L. Mariani, F. Pastore, and M. Pezze. Dynamic analysis for diagnosing integration faults. *IEEE Trans. Softw. Eng.*, 37(4):486–508, July 2011.
4. F. Pastore, L. Mariani, A. Goffi, M. Oriol, and M. Wahler. Dynamic analysis of upgrades in c/c++ software. In *Proceedings of the 2012 IEEE 23rd International Symposium on Software Reliability Engineering*, ISSRE '12, pages 91–100, Washington, DC, USA, 2012. IEEE Computer Society.
5. F. Pastore, L. Mariani, A. E. J. Hyvärinen, G. Fedyukovich, N. Sharygina, S. Sehestedt, and A. Muhammad. Verification-aided regression testing. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, ISSTA 2014, pages 37–48, New York, NY, USA, 2014. ACM.