Computer Science Research Proposal

Automated Statistical Abstraction of Programs

Michael J A Smith
<mjas2@cam.ac.uk>

March 1, 2005

The application of automated theorem proving to *program verification* is increasingly being used in real-world situations, most of which are concerned with proving safety properties of code. For example, the Berkeley Lazy Abstraction Software Verification Tool [4] (Blast), and Microsoft®'s SLAM Toolkit [1] are applied to the verification of safety properties in drivers. Such use entails considering all possible execution paths, so that a property can be verified to hold in all circumstances[1]. This approach therefore deals mainly with the processing of situations that do not commonly arise in practice, hence are not captured by conventional testing.

On the other hand, the issue of *modelling computer systems* is primarily concerned with statistical properties of a system, and therefore the common case execution paths. An example of such an application is seen in network modelling; that is to say inferring the statistical behaviour of data flows on a network. Currently, such modelling is done either by simulation, or by use of statistical frameworks such as queueing theory. While the latter may allow a sophisticated analysis of global properties, one has to be cautious of the assumptions made of the underlying physical system.

The challenge here lies in relating the model of a system to its actual implementation, and therefore in the interface between the 'bottom-up' approach taken by program verification, and the 'top-down' approach taken by behavioural modelling. The proposed research is to investigate the extent to which the two processes may be unified. For example, it should be possible, when provided with suitable data regarding the use of a procedure (such as the implementation of a queue), to derive statistical properties, such as the probability of a queue being full on a request to push more data onto it, given a distribution of arrival times. Conversely, one might like to generate approximations to the service time distribution of a queue, based on the implementation itself, in the context of normal operating parameters. There are two related problems here; that of verification that a queue corresponds (within specified bounds) to a service time distribution, and that of deriving the distribution itself.

Any work of this nature would be required to deal with the language used in real-world implementations; namely C. Despite the issues observed due to lack of type safety and pointer related mishaps, substantial work has already been undertaken to address these problems. Most notable is the CCured [8] system, developed at Berkeley, which performs transformations on C code, in order to enforce type-safety in existing programs. Although this deals with just a subset of the language, it is noted that the majority of code does not take advantage of the more esoteric capabilities of C, and therefore may be readily analysed. CCured performs its

[1] When dealing with such real-world systems, one can only give such a 'proof' under certain well-defined assumptions, such as a lack of concurrency, for example.

analyses and transformations using the C Intermediate Language (CIL) [7], which is a 'clean' subset of C, which is manipulated via an OCaml library.

Further to this, most code written in C has the advantage of containing many sequential assignments (not far from the algebraic normal form (ANF) used for simplifying code in functional languages), from which predicates on the variables may be inferred. Furthermore, the time costs of the code are manifest, given that it is much closer to actual assembler than is the case with higher-level languages. Such information can be used to generate useful verification conditions without the requirement of program annotation by the user. This has been used to great effect in the aforementioned Blast and SLAM systems. Blast generates and iteratively refines an abstract model such that either no error nodes are reachable, or there exists a path to an error state (which is given as a counterexample to the property being proven). A further system is the C Bounded Model Checker [3] (CBMC), one aim of which is to verify consistency between a specification written in a hardware description language (such as Verilog) and a prototype implemented in C.

There has already been substantial work in extending models and model checking to deal with stochastic or probabilistic properties. The work at Edinburgh on the Performance Evaluation Process Algebra [5] (PEPA) provides a process algebra that is able to express continuous-time Markov chains (CTMCs). This is an extension of Milner's Calculus of Communicating Systems (CCS) such that actions have a parameterised exponentially distributed duration, rather than being instantaneous. The complementary work at Birmingham on a Probabilistic Symbolic Model Checker [6] (PRISM) is an extension of Computation Tree Logic (CTL) to allow properties which express a probability of being true. This enables model checking on the basis of a statistical specification. Despite progress made in both areas of work, however, there remains the fundamental problem of relating a model to actual code (or more specifically, deriving the former from the latter), which is what this research aims to address.

In light of this, a more interesting application of such a statistical perspective would be to flow- and congestion-control mechanisms in a network protocol stack. Instead of generating an abstract model for the purposes of a (safety) proof, one could abstract to a *statistical* model (for example, in the form of a stochastic process), when given well-defined parameters describing the data generation of the application and conditions on the network. In the case of congestion control, interesting properties include the average throughput, and the variation in data rate (jitter) of the connection. This abstraction could then undergo iterative refinement in a similar way to conventional approaches[2], except that it would be refined in terms of goodness of fit of the model, as opposed to the correspondence between paths to error states in the abstract and physical systems. Hence we are still comparing the predictions of the model with the behaviour seen in the code. The combination of a 'conventional' abstraction (using verification conditions derived from the code) and a statistical abstraction alongside it (or above it) could be a powerful notion in providing a stronger correlation between statistical modelling and implementations.

Current work in formalising network protocols includes the use of labelled transition system semantics[3][2], which may be verified against behaviour seen in test systems, using a theorem prover. However, this still requires substantial human input, both in terms of investigating the implementation's source code, and constructing test-cases to exhibit its behaviour. The argument for direct inference from the source code is that such static analysis could be more useful in understanding the interaction of the implementation with the network at a global level, at the sacrifice of lower-level details.

[2]There is a relationship here with code profiling, however here we do so with respect to generalised statistical models, rather than simply execution time.

[3]The author worked as a research intern on the Network Semantics project (see `http://www.cl.cam.ac.uk/users/pes20/Netsem`), during the summer of 2004.

The proposed research is to investigate the possibilities in this area, and the extent to which such 'statistical abstraction' can and cannot be done. It is felt that while positive results of such research could be of great value to the network community (for example, resulting in less reliance on simulation), the attempt at uniting the methodologies of program verification and modelling/model checking is a worthwhile investigation in itself.

During the first year of undertaking this research, I intend to develop a complete view of the work already undertaken, and to familiarise myself with the tools required to carry out such work. The initial stages will be concerned with developing the theoretical framework for statistical abstraction and refinement, and in determining the conditions under which a refinement will converge on a model with suitable error bounds. This will be tightly coupled with the subsequent task of actually implementing such techniques. By the end of the first year I intend to be able to infer simple statistical properties of basic queue implementations, and to have a clear idea as to how to combine this with the existing work on probabilistic model checking. To this end, the produced deliverable will be a report of such findings.

References

[1] T. Ball and S. K. Rajamani. The SLAM toolkit. In *Conference on Computer Aided Verification*, 2001.

[2] S. Bishop, M. Fairbairn, M. Norrish, P. Sewell, M. Smith, and K. Wansbrough. Rigorous specification and conformance testing techniques for network protocols, as applied to TCP, UDP, and sockets, 2005. Not yet published, see `http://www.cl.cam.ac.uk/users/pes20/Netsem/`.

[3] E. Clarke, D. Kroening, and F. Lerda. A tool for checking ANSI-C programs. In Kurt Jensen and Andreas Podelski, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 2988 of *Lecture Notes in Computer Science*, pages 168–176. Springer, 2004.

[4] T. A. Henzinger, R. Jhala, R. Majumdar, and G. Sutre. Software verification with Blast. In *Proceedings of the Tenth International Workshop on Model Checking of Software (SPIN), Lecture Notes in Computer Science 2648*, pages 235–239. Springer-Verlag, 2003.

[5] J. Hillston. Compositional Markovian modelling using a process algebra. In W.J. Stewart, editor, *Numerical Solution of Markov Chains*. Kluwer, 1995.

[6] M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. In P. Kemper, editor, *Proc. Tools Session of Aachen 2001 International Multiconference on Measurement, Modelling and Evaluation of Computer-Communication Systems*, pages 7–12, 2001.

[7] G. C. Necula, S. McPeak, S. P. Rahul, and W. Weimer. CIL: Intermediate language and tools for analysis and transformation of C programs. In *Proceedings of Conference on Compiler Construction*, pages 213–228, 2002.

[8] G. C. Necula, S. McPeak, and W. Weimer. CCured: Type-safe retrofitting of legacy code. In *Proceedings of the Principles of Programming Languages*, 2002.