5605

Project Title: SAT: A Scalable Analysis Toolkit
PI: Prof. Alexander Aiken
Period: 8/1/98-7/31/2001
Institution: University of California, Berkeley
Address: UC Berkeley, Berkeley, CA 94720
Contract Number: NAG2-1210

# Final Report (Summary of Research)

The SAT project aimed to demonstrate that it is feasible and useful to statically detect software bugs in very large systems. The technical focus of the project was on a relatively new class of constraint-based techniques for analysis software, where the desired facts about programs (e.g., the presence of a particular bug) are phrased as constraint problems to be solved.

At the beginning of this project, the most successful forms of formal software analysis were limited forms of automatic theorem proving (as exemplified by the analyses used in language type systems and optimizing compilers), semi-automatic theorem proving for full verification, and model checking. With a few notable exceptions these approaches had not been demonstrated to scale to software systems of even 50,000 lines of code.

Realistic approaches to large-scale software analysis cannot hope to make every conceivable formal method scale. Thus, the SAT approach is to mix different methods in one application by using coarse and fast but still adequate methods at the largest scales, and reserving the use of more precise but also more expensive methods at smaller scales for critical aspects (that is, aspects critical to the analysis problem under consideration) of a software system. The principled method proposed for combining a heterogeneous collection of formal systems with different scalability characteristics is mixed constraints. This idea had been used previously in small-scale applications with encouraging results: using mostly coarse methods and narrowly targeted precise methods, useful information (meaning the discovery of bugs in real programs) was obtained with excellent scalability.

The SAT system is constraint-based, meaning that analysis are formulated as systems of constraints generated from the program text. Constraint resolution (i.e., solving the constraints) computes the desired information. Mixed constraints combine multiple constraint languages in a single application. Individual constraint languages are treated as generic "building blocks" that can be combined in a structured way to create new analysis for specific problems. The framework is extensible, meaning that new constraint languages can be added that conform to a standard interface.

SAT very successfully demonstrated the idea of applying constraint-based analysis to very large programs. Using alias analysis of C programs as a paradigmatic example---alias analysis is critical to many software engineering tools, particularly for the accurate detection of bugs in software---we demonstrated in a series of papers techniques for extending the application of alias analysis from programs of a few thousand lines up to, eventually, 440,000 lines of C code. This represented an improvement of two orders of magnitude and was for two years far and away the most effective form of alias analysis known. The upsurge in interest in scalable alias analysis in the research community was, we believe, inspired by the progress we made, and has subsequently led to the development of many new techniques, all constraint-based and kin to the techniques we presented.

The stated goal of the project was to analyze a 1,000,000 line program. While we came close, we never did achieve this goal, because we discovered that there are no 1,000,000 line programs in the public domain! The largest available programs are about half that size, and that is where we stopped: we eventually ran out of programs, not out of scalability of our techniques. Much larger programs are available in industry, of course, but these are proprietary and not easy to obtain for research purposes outside of the corporations that own them.

In the area of detecting bugs in software, the project developed three distinct systems. The first was Carillon, a tool for detecting year 2000 errors in C programs. This system was constructed mainly to validate the hypothesis that the mixed-constraint toolkit underlying SAT provided an effective tool for rapidly developing new program analyses. This turned out to be true: Carillon was done in 1 man month by a single programmer who knew nothing about Year 2000 problems or SAT when he started.

The second effort was a tool for detecting buffer overruns in C programs. Buffer overruns, or the ability in C for programs to write past the end of an array, are a major security problem today, as many buffer overruns can be exploited to taking control of a machine. Many, if not most, computer viruses are based on exploiting buffer overruns. In this work, we developed a constraint-based method for predicting the size of an array and the total size used by the program; by comparing the two we could potentially identify buffer overruns. This worked surprisingly well in practice, as we found exploitable security holes in very widely used software such as the Linux networking packages.

In the third piece of work, we identified a very broad class of properties that could be checked by allowing programmers to add their own type qualifiers to a language's type system. For example, a certain routine may require that all of the lists it is passed as arguments be sorted. By adding qualifiers sorted/unsorted to the type system, and noting that the sorting function produces results of type "sorted list", it becomes possible to check this kind of invariant. This has opened up a whole field of inquiry for us, as we investigate what kinds of type qualifiers will help us to analyze what sorts of program properties.

Attached is a list of publications produced by the project as well as URLs of distributed software systems.

**Software**

BANE: A Constraint-Based Analysis Toolkit
bane.cs.berkeley.edu

Carillon: A Tool for Detecting Y2K Bugs in C Programs
bane.cs.berkeley.edu/carillon

**Publications**

J. Foster, M. Faehndrich, and A. Aiken. Polymorphic versus Monomorphic Flow-Insensitive Points-to Analysis for C, The 2000 Static Analysis Symposium, pages 175-198, June 2000.

D. Wagner, J. Foster, E. Brewer, and A. Aiken. A First Step Towards Automated Detection of Buffer Overrun Vulnerabilities. Proceedings of the 2000 Network and Distributed Systems Security Conference, pages 3-17, February 2000.

B. Liblit and A. Aiken. Type Systems for Distributed Data Structures, ACM Symposium on Principles of Programming Languages, pages 199-213, January 2000.

Z. Su, M. Faehndrich, and A. Aiken. Projection Merging: Reducing Redundancies in Inclusion Constraint Graphs, ACM Symposium on Principles of Programming Languages, pages 81-95, January 2000.

J. Foster, M. Faehndrich, and A. Aiken. A Theory of Type Qualifiers. Proceedings of the SIGPLAN Conference on Programming Language Design and Implementation, pages 192-203, Atlanta, Georgia, June 1999 (with J. Foster and M. Faehndrich).

M. Faehndrich, J. Foster, and Z. Su, and A. Aiken. Partial Online Cycle Elimination in Inclusion Constraint Graphs. In Proceedings of the SIGPLAN Conference on Programming Language Design and Implementation, pages 85-96, Montreal, Canada, June 1998.