Scott McPeak

Research Statement

My main research interest is in tools and techniques to improve software quality. In this statement I describe my past involvement in several research projects whose goal was to improve software quality or improve the infrastructure for conducting software quality research, and conclude with plans for future work in this area.

CCured

CCured [1,2,3,4] is a source-to-source translator that instruments C programs with run-time checks that ensure memory safety. Because of an aggressive initial static analysis, CCured's checks are typically very inexpensive, slowing an application by between 20 and 60% (in comparison, most memory checkers for C slow applications by at least an order of magnitude). The project was led by Prof. George Necula and involved half a dozen graduate students and a few undergraduates.

My role on the project included porting applications to CCured, implementing and maintaining the run-time library, designing the mechanism by which CCured's static analysis soundly manages padding and alignment within structures despite the fact that CCured affects structure layout, and interfacing CCured with the Edison Design Group (EDG) C++ Parser [5], an effort that culminated with the successful translation of SafeTP [6], a moderate (around 40,000 lines) sized C++ program. (SafeTP is a secure FTP client and server proxy I co-wrote with Dan Bonachea.)

Our group's efforts on CCured have made it more than a mere "paper prototype", as a number of other groups around the world are currently evaluating or using CCured (and its CIL [3] infrastructure) for various purposes. Among other applications, we have demonstrated its ability to reasonably easily instrument network server programs, making it realistic to put an end to security vulnerabilities due to buffer overflows in C code.

Elkhound

Elkhound [7] is a Generalized LR (GLR) parser generator. The GLR parsing algorithm tolerates nondeterministic and even ambiguous grammars by in effect forking the parse stack when an LR conflict is encountered. The Elkhound implementation is remarkable for two reasons. First, it is a factor of 10 faster than other GLR implementations for grammars that are close to LALR, making it practical for production use. Second, it uses a redesigned worklist algorithm that guarantees reductions and merges are performed strictly bottom up, which enables its use in a parser *generator* (other GLR parsers must build parse trees). Elkhound and Elsa (below) were primarily written by me, with guidance from Prof. Necula.

Elsa

Elsa [8] is a C++ parser built on top of Elkhound. Its novel features include a very flexible language extension mechanism, good performance, and the ability to translate away some of the more difficult C++ features such as templates and implicit function calls (constructors, conversions, etc.). While it has not yet achieved full conformance with the language standard, it is able to parse a number of large and important C++ applications such as Mozilla and Qt.

My interest in the Elkhound and Elsa projects came from the lack of a good, general-purpose parsing framework and C++ parser for research use. C++ is an important and widely-used language in practice, but automatic parsing techniques (such as ordinary LALR) were useless for it, and existing parsers were either proprietary (e.g., EDG), making it hard to share research results, or difficult to extend (e.g., gcc), making it hard to experiment with new language ideas. Elkhound has significantly expanded the territory of languages amenable to automatic parsing techniques, and Elsa makes C++ tractable for researchers in academia.

Verification of Pointer-Intensive Programs

Most of my current research effort is in the direction of formally verifying correctness properties of programs that use pointers in nontrivial ways. While the fundamental techniques of software verification are over 35 years old, applying them to data structures with complicated sharing relationships has remained an obstacle. The field needs practical solutions for pointers.

Prof. Necula and I have developed a system for reasoning about pointer-based data structures based on what we call "local equality axioms". The technique is sufficiently expressive to capture the important properties of many well-known data structures, and in particular is able to relate the *shape* of such structures to the *contents* of the data within them. Furthermore, the description language is tractable, as a large fragment is provably decidable and much of what remains is solvable in practice with a few simple heuristics. We have applied the technique to verify the correctness of both textbook data structure kernels, and some real-world programs such as Linux device drivers. Our paper on this topic is currently undergoing peer review, and is available upon request.

Future Work

I believe the notion of practical formal verification is becoming more and more realistic. The community's understanding of how to reason about data structures (e.g., [9,10]) has advanced significantly in the past five years, and in combination with ongoing work on system modularity we may finally be able to build systems whose burden of use is only marginally higher than that of a static type system, but with benefits closer to those of exhaustive testing. My plan is to continue working on improving the applicability of verification systems.

References

1. CCured: Type-Safe Retrofitting of Legacy Software
   George C. Necula, Jeremy Condit, Matthew Harren, Scott McPeak, Westley Weimer
   In ACM Transactions on Programming Languages and Systems (TOPLAS), to appear, 2004
2. CCured in the Real World
   Jeremy Condit, Mathew Harren, Scott McPeak, George C. Necula, Westley Weimer
   In Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation (PLDI03), June 2003.
3. CIL: Intermediate Language and Tools for Analysis and Transformation of C Programs
   George C. Necula, Scott McPeak, S. P. Rahul, Westley Weimer.
   In Proceedings of Conference on Compiler Construction (CC02), March 2002.
4. CCured: Type-Safe Retrofitting of Legacy Code
   George C. Necula, Scott McPeak, Westley Weimer.
   In Proceedings of the 29th ACM Symposium on Principles of Programming Languages (POPL02), January 2002.
5. http://www.edg.com
6. http://safetp.cs.berkeley.edu
7. Elkhound: A Fast, Practical GLR Parser Generator
   Scott McPeak, George C. Necula.
   In Proceedings of Conference on Compiler Constructor (CC04), April 2004.
8. http://www.cs.berkeley.edu/~smcpeak/elkhound/sources/elsa
9. Anders Möller and Michael Schwartzbach
   The Pointer Assertion Logic Engine
   PLDI 2001.
10. Shmuel Sagiv, Thomas W. Reps and Reinhard Wilhelm
    Parametric Shape Analysis via 3-Valued Logic
    TOPLAS 24(3), 2002.