MC Compiler Extension Review
Gabe Black

This paper was about using a language called Metal to write extensions for a compiler to catch domain specific programming errors. The Metal language is similar to the language used for yacc and generates a set of states and state transitions. When a piece of source code is run through the compiler, it is eventually passed to the extension a user has provided. For each execution path in the source, the extension finds what states the program will transition through, changing state each time it finds a state transition identified by special patterns in the source code. If the code moves into an illegal state, the extension flags this as an error.

One thing this paper did well was to provide many examples of how the system can and has been used to identify errors in real systems. Also, it was interesting to see how it could be used to find potential optimizations in code. Also, many of the examples included specific pieces of code that contained flaws, and the Metal code for the extension that found them. The number and variety of errors the system found was quite impressive, and it seems like this tool has potential for a bright and productive future.

One thing I didn't like so much was that the paper used too many examples. After a certain point, the examples didn't really add to my understanding of the system and became more of a chore to make my way through. It would have been much better to show unique ways the system could be used to accomplish distinct tasks with different goals. One step in the right direction was the explaining how the system could be used to find optimizations in highly optimized code.

Many times throughout the paper, the authors mentioned that xg++ was built on top of a C++ compiler, but almost all of the examples the described were system level code written in C. The authors commented that there were many false positives because of C++'s more strict typing system and extensions that were made to g++ that are illegal in C. It seems to me that it would have been much more sensible to work from a C compiler directly, rather than try to retrofit a C++ compiler and then work with C code.

Another lacking I saw in the paper was that they didn't describe in much detail the process an individual might go through to use this system. The Metal language could have been explained in more depth, and a specific walkthrough of a typical run of an extension would have helped me to understand the system from a practical point of view.