

STATEMENT ON RESEARCH

Timothy Richards

timothy.richards@trincoll.edu

My research goal is to improve the speed, portability, adaptability, and safety of systems software. This includes pushing the envelope in adaptive retargetable compilation, application of hardware specific optimizations, automatic generation and verification of compiler backends, and agile compile-time and run-time environments. I envision software that can be automatically retargeted to new architectures easily, verified for correctness using formal specifications, and tuned automatically to optimize machine code for specific environments based on the targeted architecture, its unique set of features, and the executing operating system.

As computer architectures migrate toward highly parallel and heterogenous processors in a networked environment the retargetability of system software becomes increasingly problematic. These machines vary widely in the number of useable cores and in their performance and capabilities. To take advantage of the performance potential, new optimizations and substantial re-tuning of code are required with each successive processor generation. Currently, system software can't adapt at the pace necessary to keep up with these changes making the design, exploration, and usability of new architectures and computer systems difficult. This will have great technological and economic consequences as the time to market increases. My research focuses on techniques that will enable system software to adapt more easily, quickly, correctly, and automatically.

My research approach is to draw upon a broad range of knowledge to synthesize solutions. In the systems area, I look toward programming language and compiler techniques, computer architecture, and networking for observing, understanding, and solving problems related to system software. I emphasize artificial intelligence and machine learning methods to generate solutions that require elements of search and adaptation. I use formal specifications to provide correctness guarantees on generated code and optimizations. In the future, I am interested in using genetic algorithms and formal specifications to improve security using compilation techniques that generate system code that is robust. The combination and cross pollination of ideas from different disciplines is an exciting and fruitful way to explore new representations, approaches, and solutions to solve problems related to the compilation and generation of system software.

My earliest work as a graduate student focused on the co-evolution of compilers and simulators to support architectural exploration and evaluation. The goal was to survey the current state of the art in compiler and simulator construction and propose research directions for building them automatically from a common infrastructure. The results of this work led to a number of research avenues and the creation of CoGenT, a new research project focused on the co-generation of system tools. I contributed to several aspects of CoGenT including the design of machine- and compiler-oriented description languages, techniques and software for generating a variety of system tools automatically, and several grants that have funded my work during my graduate career.

I explored the application of term rewriting systems using machine related axioms to prove the semantic equivalence of compiler intermediate representation (IR) and target machine instruction sequences. This work stimulated my thesis that focused on new techniques for generation instruction selectors automatically from machine descriptions. Specifically, my dissertation explores the idea of generating an instruction selector using strategies that are independent of the compiler framework and target machine. To accomplish this, I introduced a universal description language (CISL) that is capable of describing both the compiler IR and instructions for the target machine. The descriptions of the IR and target are then used as input to a heuristic search procedure that uses term rewriting and exploits algebraic properties of the CISL language to match IR operations to sequences of target instructions. In the end, search produces a set of compiler independent instruction selector patterns that are used to generate the instruction selector component of a specific compiler framework. This approach has been successful in generating instruction selectors for a number of compiler/target combinations.

The automatic generation of compiler instruction selectors is an important step toward producing system software that is more portable, however, a significant amount of work remains in generating a complete compiler backend automatically (e.g., register allocation, scheduler, peephole optimization). My near-term research goals are to continue work on compiler and architecture independent solutions for generating compiler backends automatically and easily. At the same time I am interested in researching compilation techniques for dynamic languages in a networked environment (e.g., client and server-side JavaScript). In particular, I am interested in ways to improve the runtime performance of just-in-time (JIT) compilers, performance of the generated code, safety, security, and power consumption of the compiled code, and the adaptability of the JIT system to changes in the operating environment and underlying architecture using formal specifications, automated techniques, and machine learning. My long-term goal is to produce fast and reliable system software that is easily retargeted from one platform to the next using techniques that I have developed and will continue to expand in the future. This research will also be used as fuel to improve techniques for teaching students the complex nature of the software-hardware interface and the difficulties related to safety, security, and power at this level.