Research Statement of Jesse Bingham

March 5, 2005

Computer hardware, and thus computer software, is becoming increasingly complex as facilitated by Moore's Law. With this swelling of complexity comes an increase in the number and subtlety of design bugs. Traditional simulation-based testing can only *increase confidence* in design correctness by catching bugs, and can hence allow the subtler bugs to persist into fabricated silicon (for hardware) or shipped binaries (for software). *Formal verification* strives to ameliorate this situation by *proving* correctness in the mathematical sense. Here the proposition that a *model* of the system adheres to its formal *specification* is viewed as a theorem that must be proved. A fundamental concern in the practical application of formal verification is the vast human resources required, namely expertise and time. To this end, techniques such as model checking, decision procedures, and automated theorem proving have emerged. These approaches, and combinations thereof, offload work onto the computer and reduce the likelihood of human error resulting in a flawed proof.

There are several overlapping hurdles that must be overcome before formal verification sees widespread use. First are the limits on *capacity* that arise in the automatic techniques. The *state explosion* problem, for example, is the Achilles heel of model checking. Also, many useful logics in formal verification suffer from high computational complexity or even undecidability, thus hindering decision procedures. As a result, often only very scaled down and simplified versions of the "real" system can be formally verified. Since scaling-down and simplifying can bury bugs, we are left with a similar problem as in simulation-based verification. Related is the separation between the verified model of the system implementation and the system implementation itself. Formally proving that the latter realizes the former would clearly help solve this problem. Finally, expressiveness is a concern; this is a question of whether the methods support verification of the properties desired by the user.

Research on formal verification spans a wide spectrum between theory and practice. Many papers in the area are admittedly interesting and intellectually pleasing, while lacking a solid foundation in reality. However, endeavors to make formal verification practical can have theoretical depth. As the following examples of my contributions and potential research directions attest, such endeavors are where I find my niche.

Memory Systems. Aside from my thesis work, I have had two opportunities to formally model and verify realistic memory systems. The first of these was an effort to verify safety and liveness of several variants of the *Token Coherence* protocol. The modeling was done using Lamport's *temporal logic of actions* (TLA), and several properties were model checked using the accompanying model checker, TLC. My second experience was during an internship at Microsoft Research in Silicon Valley. Here I modeled aspects of the memory system of Microsoft's XBox2 gaming console, also using TLA. An important result of this project was the discovery of a serious coherency bug. The project engineers indicated that their simulation-based testing would likely have been unable to expose this flaw.

Studying these and other systems has given me a sense of what real hardware protocols look like, which has informed some of the verification research presented in my doctoral dissertation and discussed below.

Verification of Shared Memory Models. One of the most difficult and important aspects of designing a shared memory multiprocessor is getting the memory system right. The natural specification of a shared memory system is the *memory model*, which defines how memory will appear to the programmer,

in terms of semantics of memory events such as reads, writes, and various synchronization primitives. Verifying the memory model directly is clearly superior to checking safety properties that are thought to entail memory model conformance, since the former is the true specification. Memory models are typically defined in terms of the existence of certain permutations of system behaviors, which are hard to express in classical formalisms such as temporal logic.

Sequential consistency (SC) is the archetypal memory model, and is a focus of my dissertation, in which a refinement of SC called *decisive SC* (DSC) is introduced. DSC demands a property observed in all "real" shared memory systems that is not inherent in the standard definition of SC. Intuitively, this property requires that each read event must inherit its value from a fixed write event. An extensive study of the complexity of various verification problems related to DSC is explored in my dissertation. A key result is that a certain bounded variant of DSC, that captures the behaviors of any conceivable protocol, is decidable, whereas SC itself is known to be undecidable. In support of this study, I developed a data structure called a *view window*, which summarizes the current reordering of events.

I am interested in extending this work to other memory models. I believe that definition of a similar structure to view windows could be employed in their verification. Such an approach would have the user augment a system description so that each state is associated with such a structure, and then a model checker or theorem prover would be used to check that the augmentation legally witnesses the memory model.

Also of importance in memory system verification is linkage of a high level system description to lower level descriptions, i.e. gate or RTL. I hope to look at ways to use contemporary theorem proving and decision procedure technology to formally bridge this gap.

Parameterized Verification. One method of dealing with the capacity problem is by *parameterization*. Most systems are parameterized along several dimensions, e.g. bit-width of a data path circuit, number of processors or addresses in a shared-memory system, depth of queues in a chip-level protocol, number of linked-data structure nodes in a heap manipulating program, etc. Let $S(n)$ be such a system instantiated with parameter value n. A parameterized verification method is one that uniformly verifies correctness of $S(n)$ for all naturals n. In some sense, parameterized verification, when possible, can be viewed as a solution to the capacity problem; the instance $S(128)$ might be well beyond what any conceivable finite state model checker could handle, however the successful application of parameterized verification implies correctness of $S(128)$.

My doctoral dissertation makes two contributions in the realm of parameterized verification. First, I present an abstraction-based framework to soundly verify (a subset of) DSC of a shared memory system, parameterized over both addresses and data values. The technique is successfully applied to two nontrivial systems. In the future, I would like to investigate the possibility of using a similar approach to verify other memory models. The second contribution is in regard to the verification of so-called *well-structured transition systems* (WSTS), which, simply put, are a class of parameterized systems with certain properties that allow for decidability of safety properties. In my dissertation, I show how binary decision diagrams can be employed to verify these systems, hence allowing for an (empirically) compact representation of state spaces. This technique is a major improvement over previous algorithms when applied to problems with large local state. Many systems cannot be expressed as WSTS; for example those with *conjunctively guarded transitions* violated well-structuredness. So that my approach can be applied to such systems, I have developed a sound and automatic reduction which, in effect, removes conjunctive guards. This method allowed for the parameterized verification of German's cache coherence protocol, which is a widely-circulated challenge problem for parameterized verification. I am interested in extending this approach to handle broader classes of systems, including linked-data structure programs, as discussed below.

Static Analysis. My approach to safety property verification of WSTS, mentioned previously, can very roughly be explained as follows: analyze the reachable states of $S(1)$, then $S(2)$, then $S(3)$, etc.,

until either a bug is found, or a certain convergence condition is reached. This convergence condition guarantees that if no bug has been found thus far, then there are none in any larger system instances.

I would like to try to find applications of this paradigm in software verification, in particular programs that manipulate linked data structures through pointers. The intuition here is that, if the program functions correctly for, say, 10 nodes, then it would appear highly unlikely that it would fail for some number of nodes ≥ 11. During a forthcoming visit to Microsoft Research's Software Productivity Tools group, I plan on considering the verification problem for such programs from this perspective. We will attempt to find useful combinations of language syntax, specification logic, and abstractions that facilitate theoretical results analogous to my convergence condition for WSTS.

SAT Solving. Many fundamental problems in formal verification can be compiled into instances of the *Boolean satisfiability* (SAT) problem; such problems include *bounded model checking* and *combinational equivalence*. Hence efficient SAT solvers play an important role in the field. Recently, I have done some work on a new pruning technique called *B-cubing*. Our implementation of this algorithm has proved to be competitive with the predominant SAT solver ZChaff II. Previously, I have also undertaken a project to use specially modified compiled circuit simulation for semi-formal bounded model checking.

In the future, I hope to find myself in an industrial lab, working in collaboration with other formal verification researchers. I will seek problems similar in flavor to those considered in my previous work: practically important, and hard enough to require theoretical depth to solve. I see myself as a generalist within the domain of formal verification, and am always eager to expand my knowledge of this topic as well as related areas such as Computer Architecture, Logic, Programming Languages, Theoretical Computer Science, and Mathematics.