# Midwest Verification Day 2011

**Analysis of CPS Control Systems**
Jason Biatek, University of Minnesota

**Rewriting approach to type assignment**
Peng Fu, University of Iowa

In the paper "type preservation as a confluence problem", a novel approach is proposed to deal with type preservation, i.e. develop an equivalent rewrite system to simulate typing. Then the type preservation property in type theory becomes a confluence problem in rewriting system. In this talk, I will show the differences of the correspond rewrite systems between curry style and church style type systems. Concluding with some of the problems we encounter when we are trying to develop an appropriate rewrite system for curry style typing.

**A Proof Checker for Equational Reasoning in a Lazy Functional Language**
Adam Procter, University of Missouri

This talk will outline the design of MProver, a practical proof checker for a pure, lazy functional language. MProver is designed to support machine-verified equational reasoning about purely functional programs. The main distinguishing feature of MProver is its support for reasoning over possibly diverging computations and infinite data structures, and the addition of Haskell-style type classes enables a modular style of verification that closely follows familiar functional programming idioms. A prototype implementation of MProver is currently being developed.

**Guardol: A Language and Environment for Reasoning about Guard Applications**
Hung Pham

**A formally verified SSA-based middle-end**
Delphine Demange, INRISA

Recent work by Leroy et al. have lead to the first realistic C compiler whose correctness proof has been machine-checked. Such a work requires to carefully select the middle-end optimizations, trading off the efficiency of the generated code with the tractability of the optimization correctness proofs. SSA form is an intermediate language that is now widely used in modern compilers : the strong invariants it captures allows for writing simpler and faster optimizers. So far, several formal semantics have been proposed for SSA, but it is still believed that mechanically proving a SSA-based compiler would demand too much proof effort compared with the performance gain brought by SSA-based optimizations. In this work we provide a simple and intuitive formal semantics for the SSA form of the CompCert intermediate language RTL. We use the Coq proof assistant to prove correct the conversion from RTL to SSA using translation validation; we also provide a formalization of a GVN-based CSE optimization and prove correct the translation from SSA back to RTL.

**AMIBE: an Imperative Programming Language with First Class Continuations**
Yuting Wang, University of Minnesota

A continuation represents the future of an execution. It is often used as an intermediate representation(IR) to compile functional programming languages, make control flow explicit and full beta-reduction(function inlining) possible. Continuations are also a language feature that gives user the ability to completely control the execution control flow(first class continuation). Efficient implementation of first class continuation is important for languages that need non-determinism and backtracking(e.g., Comet).

We present a prototype imperative programming language with first class continuation – AMIBE. AMIBE uses the LLVM compiler infrastructure which is attractive for its optimizing tools and overall modern organization. However, LLVM does not support the implementation of continuation via a direct manipulation of the system stack. To move the execution state out of the system stack into a separate AMIBE stack, AMIBE adopts the Continuation Passing Style compilation technique(CPS). With CPS, states on the system stack are never reused since functions never return. Portable implementation for first class continuation becomes possible because the compiler only needs to save and restore the AMIBE stack which it fully controls. In CPS, function calls are tail calls. By exploiting the optimization for tail calls in LLVM, function calls are reduced to jumps, so that the system stack never grows on calls. AMIBE programs are
first compiled into an AMIBE IR closely related to LLVM IR, then transformed into CPS form. Finally the AMIBE IR in CPS is translated into LLVM IR. The performance of the optimizing compiler based on LLVM and CPS is compared against a naïve just-in-time compiler based on GNU lightning and currently used by Comet.

**Separation of Proof and Program**
Harley Eades, University of Iowa

This talk summarizes the design of a new dependently-typed core language called Separation of Proof and Program (Sep3). Sep3 consists of two disjoint fragments: a logical fragment and a programmatic fragment. The logical fragment is a higher-order type theory used to reason about programs written in the programmatic fragment. While the programmatic fragment is a general purpose programming language with general recursion. We will describe both fragments and several novel features of the language. Some of the features that will be discussed are how proofs may include arbitrary programs, a new unary predicate used to establish the termination of a program, and a built in ordering that may be used by inductive proofs.

**Reasoning about general recursive functions in SepPP**
Garrin Kimmell, University of Iowa

SepPP (separation of proof and program) is a dependently typed programming language which supports the definition of, and reasoning about, general recursive programs. In this talk, we give a brief overview of reasoning about such programs within the language. Furthermore, as SepPP is designed as a core language, it places a significant burden on the programmer to fully annotate proofs. We describe a small collection of methods for reducing this burden, making proofs in SepPP more succinct.

**A Counterexample-Based Approach for Quantifier Instantiation in SMT**
Andrew Reynolds, University of Iowa

Satisfiability Modulo Theories (SMT) solvers have recently emerged as powerful tools for handling satisifiability problems with respect to the theories of certain data types. A current challenge for SMT remains dealing with satisfiability problems that contain universal quantifiers. Recent approaches have relied on techniques such as E-matching for quantifier instantiation. A weakness of this approach is its ability to recognize the relevance of instantiations in a precise way. In this talk, we present a new approach to quantifier instantiation in SMT, which draws upon the strengths of Model-Based Quantifier Instantiation. Our preliminary work shows that a more directed method for quantifier instantiation in SMT can be implemented based on information deduced when finding a satisfying assignment for symbolic counterexamples to quantified formulas. We provide the general idea of the method and briefly discuss its on-going implementation in the CVC4 solver.

**An Executable Formal Semantics of C with Applications**
Chucky Ellison, University of Illinois

This paper describes an executable formal semantics of C. Being executable, the semantics has been thoroughly tested against the GCC torture test suite and successfully passes 99.2% of 776 test programs. It is the most complete and thoroughly tested formal definition of C to date. The semantics yields an interpreter, debugger, state space search tool, and model checker "for free". The semantics is shown capable of automatically finding program errors, both statically and at runtime. It is also used to enumerate nondeterministic behavior.

**Invited Talk**
Grigore Rosu, Univesity of Illinois

**Building StarExec: A Cross Community Logic Solving Service**
Tyler Jensen and Clifton Palmer, University of Iowa

Automated theorem proving has become very powerful over the years with distinct communities developing around specialized logics. As of now, these communities have been largely independent, resulting in communication barriers and isolated infrastructures. We present StarExec: a cross-community computing service under development that will allow logic communities to manage their benchmark libraries, execute solvers on a large cluster and analyze results, all through an online interface. This talk will cover a brief overview of the system in terms of functionality, architecture and design, as well as problems encountered and progress made.

**Towards Oracle Creation Support**
Gregory Gay, University of Minnesota

In the testing process, a key artifact is the test oracle, which is used to determine whether an application under test executes correctly. Research has shown that the choice of test oracle can significantly impact the effectiveness of the testing process—in particular, demonstrating that careful selection of the variables monitored by the test oracle, termed the oracle data set, may yield benefits in testing. Despite the prevalence of tools that support the selection of testing inputs, few researchers have looked at *oracle creation support*.

In this work, we propose an oracle creation support method, which automatically selects the oracle data set for expected value test oracles. This approach is based on the use of mutation analysis to rank

variables in terms of fault-finding effectiveness, thus automating the selection of the oracle data. Experiments over four real-world case examples demonstrate that our method may be cost-effective approach for producing small, effective oracle data that—in our experiments—outperform the current industrial best practice by orders of magnitude.

**Matching Logic Verification using the K Framework**
Andrei Stefanescu

Matching logic is a new logic designed to state and reason about structural properties over program configurations. Syntactically, it introduces a new first-order formula construct, called a pattern, which is a configuration term, possibly containing variables. Semantically, its models are actually concrete program configurations, where a configuration satisfies a pattern iff it matches it. MatchC is a matching logic verifier for a deterministic fragment of C implemented in the K framework. In this paper we discuss: (1) the architecture of the current a version of the verifier, with emphasis on the components implemented in K; (2) our practical experience with MatchC; and (3) the design a new version of the verifier based on the recently proposed deduction system of matching logic, again with emphasis on the K part.

**versat: A Verified Modern SAT Solver**
Dukki Oe, University of Iowa

"versat" is a formally verified SAT solver incorporating the essential features of modern SAT solvers, including clause learning, watched literals, optimized conflict analysis, non-chronological backtracking, and backjumping. Unlike previous related work on SAT-solver verification, versat is implemented using efficient low-level data structures like mutable arrays for clauses and other solver state, and machine integers for literals. The implementation and proofs are written in GURU, a verified-programming language. The soundness of versat is statically verified: whenever the solver reports a set of input clauses unsatisfiable, then there exists a resolution proof of the empty clause from those input clauses. This proof is not constructed at run-time. Rather, our verification confirms statically that it exists, for all formulas versat reports unsatisfiable. An empirical evaluation shows that versat can solve and certify SAT problems on the modern scale.

**Verifying Consistency Between Multiple Models**
Steve Vestal, Adventium Labs

When engineering complex systems like planes, trains, and automobiles, many models will be developed in many languages by many engineers at many levels of abstraction and used for many purposes. A common source of design defects is inconsistencies between all these models. For example, a change made to a solid model (captured in a 3D CAD modeling environment) to improve structural properties may cause changes in mass properties that make the solid model inconsistent with a dynamical systems plant model and controller (captured in a DAE modeling environment). We would like theories, methods, and tools that would allow us to specify and verify consistency properties between multiple models. A theoretical issue is that the different languages are based on different semantic models. A practical issue is that formally specifying and verifying inter-model consistency is sufficiently novel that there is little experience with the sorts of properties that we would like to be able to specify and verify. This brief talk will present a few examples of multi-model consistency problems, will describe a small exercise that used

SMT technology to verify a multi-model consistency property, and will outline an approach that might lead to concepts that could form a basis for sounder theoretical foundations.

**An incremental and parallel k-induction model checker**
Temesghen Kahsai, University of Iowa

In this talk, we present an incremental and parallel model checking architecture to verify safety properties of synchronous systems. The architecture, which is strictly message-based, is designed to minimize synchronization delays and easily accommodate the incorporation of automatic invariant generators to enhance the general verification procedure. In particular, we developed an additive invariant generator that feeds invariants to the model checking algorithm as soon as they are produced. This architecture allows the verification of multiple properties in an incremental fashion. Specifically, the outcome of a property -- valid or invalid -- is communicated to the user as soon as the result is known. Moreover, verified valid properties are added as invariants in the model checking procedure to aid the verification of the remaining properties.

**Efficient run time monitoring for test scenarios**
Ian de Silva, University of Minnesota

Numerous activities require low-overhead monitoring of applications running on their target platform. One example of this is evaluating test suit conformance to a coverage criterion. Typically, developers instrument the application under test (AUT) with monitors to evaluate adherence to a criterion, but these perform poorly and alter the source code. In this talk, I will present a possible technique for low-overhead monitoring of AUTs utilizing He et al.'s programmable extraction logic for instruction-level event capturing on a multi-core system [1]. In this architecture, the monitor runs on a different core from the AUT and is provided with the event data. We use this to monitor the test suite execution for six sample applications, generated from Simulink/Stateflow models, for adherence to the Modified Condition/Decision Coverage (MC/DC) test criteria while measuring performance as it compares to both instrumented and unmonitored versions.

**Efficient Taint Analysis using Heterogeneous Multiprocessors**
Antonia Zhai, University of Minnesota

Availability of on-chip resources in multi-core processors have witnessed a steady increase in recent years, enabling programmers to extract performance by exploiting parallelism at different levels. This has resulted in a significant increase in the amount of parallel code being written. However, the inherent difficulty in writing parallel code brings its own set of problems along. Data race conditions and the associated correctness and security risks is one of the most important among them. Traditionally various software and hardware schemes have been proposed and implemented to detect data race conditions. However, each of the schemes have their own limitations. Software schemes are significantly slow and ineffective for real time race detection. Hardware schemes provide much better performance, nonetheless, are costly to design, verify and implement.

Industry and academia research are very active on an emergent processor architecture that combines the power of both CPU and GPU on the same silicon die. Some of these designs, like AMD Fusion and Intel Sandy Bridge, are already in market. Such architectures provide us with ample hardware resources, in the form of GPU threads, to perform race detection without costly stand-alone hardware. In this project, we propose to utilize GPU threads to perform data race detection for parallel applications running on the

CPU. By implementing a transparent race detection scheme using available on-chip hardware, we aim to significantly reduce the correctness and security risks inherent to parallel applications, without impacting their performance.

**META II: Multi-Domain Analysis of Software System Architectural Models**
Mike Whalen, University of Minnesota