Research Statement

Venkatesh Choppella

May 5, 2009

Introduction

Directions for research My current research interests are programming languages and software systems. The study of programming languages helps discover the abstractions needed for building robust software systems. Under this rubric, my research explores the formal, algorithmic and implementational issues in several related areas: automated deduction, type systems for programming languages, the design of compilers for domain-specific scientific languages, and formal methods in software and information architectures. In the near future, I plan to explore ideas from formal methods and programming languages to two emerging areas: E-Governance and Systems Biology. Meanwhile, the efforts I have already put into the development of IT related projects will continue and expand.

Outline In the rest of this document, I outline problems and directions of research that I plan to explore in the future. Some of these are a continuation of the research in which I am already engaged. Others are directions that I consider interesting and promising.

Programming Languages

Type systems Type systems are formal systems of logic used for filtering out potential runtime errors in programs. Type inference is the process by which types in a program are reconstructed in the absence of type declarations. This is specially useful for typed, higher-order functional programming languages like ML and Haskell, which support extensions of what is known as the Hindley-Milner type regime. However, insufficient type error analysis and reporting is a major impediment in the engineering and adaptation of these otherwise powerful languages. Diagnosing type errors for polymorphic, higher-order languages is still an active resarch area and falls under the broader category of static program analysis.

Term Unification and Error reconstruction for Hindley-Milner There are several directions for further work in this area. A promising theoretical approach is to think of type error reconstruction as a dual of the type reconstruction problem. Error reconstruction, at least for Hindley-Milner , can then be tied to proof reconstruction in unification, which I formulated earlier [3; 4; 5]. An early attempt at exploring type error reconstruction through automatic generation of unification proofs for Hindley-Milner considered only the monomorphic case [6]. In work that is in progress, I am investigating an approach which builds a simulation relation between the type graph generated in the full (polymorphic) Hindley-Milner case with the type graph of an expanded version of the original program. The practical objective of this effort is to build a debugging environment, in which the programmer can supply an abstract input and run the type incorrect program until it generates a type error. Our approach is to tie the error reconstruction proofs on the type graph with data and control flow information to synthesize the abstract inputs that lead to the type error. The type debugging work is in collaboration with Guillaume Marceau, on leave from Brown University. Other prospective collaborators are members of the York University's functional programming group.

Semantic unification (unification modulo an equational theory) is a generalization of syntactic unification, which is unification modulo an empty theory. Semantic unification is used when implementing decision procedures for theorem provers. I am interested in the problem of automatic proof generation for unification modulo equational theory axioms for commutativity and associativity (C, A and AC unification). I wish to use the proof generation framework for syntactic unification developed earlier [4] as a starting point.

Scientific Computing

Tensor contraction engine From 2003 to 2006 I worked at and collaborated with researchers at the Oak Ridge National Laboratory and the Ohio State University on the Tensor Contraction Engine compiler (TCE) for high performance computing. The Tensor Contraction Engine is an optimizing compiler that transforms *ab initio* quantum mechanical computations expressed as tensor equations into high performance Fortran code. The work is documented in several publications [1; 2; 14]

Optimizations and pluggable algebras TCE supported operation minimization, loop fusion, data locality optimization, integration with constraint solvers, space-time tradeoffs between memory and recomputations, and optimum memory layout of intermediate arrays. Several interesting ideas remain to be explored, however. The data structures used in TCE to represent tensor expressions are first order structures with variable names embedded in them, which makes their internal manipulation quite cumbersome. I am working on a novel variable-free representation for tensor expressions along with a calculus of selection and projection operators [7]. The second, more general problem is to

build a framework for feeding domain specific knowledge to compilers rather than hardwiring that knowledge into them, as done in TCE. It should be possible to have domain specific "plugins". These plugins are essentially term rewriting rules for an equational algebra (symmetry group axioms, for example). Feed them into a portable core compiler would then yield a specialized domain specific compiler.

High productivity scientific computing In scientific computing research, code performance was an overriding concern. Today the attention is shifting from code performance to increasing programmer productivity. Software engineering practices are increasingly becoming integral to the way computational science is practiced. For an elaboration of this idea, see the essay by Gregory Wilson [18]. My interest in is in scientific computing is in two areas: generation of code, as in the TCE compiler discussed earlier, and the verification of this code. I am interested in making scientific computing code more reliable by using technologies from program verification.

Scientific programs are replete with array computations, which involve index arithmetic and updates of index variables in loops. Errors in index computations may cause a program to crash. But sometimes they could also alter the behaviour of the program in ways that makes it difficult to trace, like slower convergence, for example. I plan to rely on dependent type analysis and higher-order contracts [10] for type checking programs with array computations. Dependent types will be used to statically validate the correctness of array index manipulations whenever possible, otherwise the checking will be done at runtime, using contracts.

Formal methods

Verification of ER models I am interested in using formal methods: specification languages, theorem proving, model checking, and other lightweight formal methods for validating various types of systems. In earlier work, I used the Prototype Verification System to validate Entity Relationship models used in database design [8]. Schema level implementations are verified against abstract data definitions and constraints specificied at the ER level. I plan to use two other approaches to simplify and automate the verification process: graph simulation, and counter-example generation techniques using the Alloy verification platform [11].

S/W Engg. approach to network design A more recent interest of mine is information and network security. I spent the summer of 2008 designing my institute's network, system and services. We had to rely heavily on extensively software engineering practices (version control, issue tracking, build scripts, etc.) for designing and maintaining the network. In our approach, a network is defined using a central specification. Configuration files and scripts for all

machines on the network are then synthesized from this specification and deployed on the respective machines. Unlike standard software distributions, deployment of these scripts means *distinct, but related* pieces of configurations need to reside on machines in the networks. This makes the deployment process trickier. For our work in this area, I plan to draw from ideas from Model Driven Architecture (MDA)-based Software Engineering [12]. MDA encourages a model-based, transformational approach to generating software. In addition to MDA, I plan to specify E-R models of in a domain specific language defined in PLT Scheme.

Static verification of network configurations Static verification of the network configuration can greatly reduce the time needed to debug a network. Perhaps the best example is that of the firewall, arguably the most crucial in a network from the standpoint of security. In work that I plan to do, firewall rules, routing tables and policies will be specified as Alloy rules generated from the Scheme specification. Alloy will then generate counter-examples that witness violations and inconsistencies in the design.

Architectures for E-Governance

Community Information Systems This is a more recent interest, spawned by my involvement in teaching and building Web applications. My research agenda here is to undertand the architecture of open, generic community information systems in a way that they can be customized to generate verifiably correct applications. I have recently started looking at the architecture of Pantoto [9], a light-weight, open source information management system. My objective is to make Pantoto a testbed platform for further research into enterprise information architectures for communities. The potential of such systems has clear implications for direct democracy and E-Governance, which I have outlined in a recent proposal [16]. There are important verification issues here: how does a system demonstrate that it conforms to properties like transparency and traceability? How are those properties even defined? E-Governance is a vast area; there are many other interesting problems to consider: data models, formalization of work-flows and information dynamics, security, deployment, localization, user-interfaces, techno-cultural issues, to name a few. Many of these issues are interrelated, and inputs from social and other sciences, and engineering discplines will be needed to address them.

Computational Biology

π-calculus models for systems biology I co-taught an elective course on computational biology last year as an opportunity to start learning about it. Two subdomains of computational biology interest me. The first, bio-informatics,

is concerned with algorithmic solutions for problems dealing with the structure of bio-molecules. The second, systems biology, studies how to model the dynamics of cellular and subcellular processes. In the past several years, there has been considerable interest in applying programming language calculi to model these processes. Towards this end, the stochastic π-calculus has been used with some success [15]. My goal is to study π-calculus and other formalisms and work with biologists and chemists to build programming models of cellular phenomena. I have recently started work with Professor M. S. Gopinathan, whose earlier work used delayed differential equation to model cell division [17]. In very preliminary work, we are exploring how cell division could instead be modeled using stochastic π-calculus.

References

[1] Alexander Auer, Gerald Baumgartner, David E. Bernholdt, Alina Bibireata, Venkatesh Choppella, Daniel Cociorva, Xiaoyang Gao, Robert Harrison, Sriram Krishanmoorthy, Sandhya Krishnan, Chi-Chung Lam, Marcel Nooijen, Russell Pitzer, J. Ramanujam, P. Sadayappan, and Alexander Sibiryakov. Automatic code generation for many-body electronic structure methods: The Tensor Contraction Engine. *Molecular Physics*, 104(2):211–228, January 2006. Invited paper for R. J. Bartlett Festschrift. [pdf].

[2] Gerald Baumgartner, Alexander Auer, David E. Bernholdt, Alina Bibireata, Venkatesh Choppella, Daniel Cociorva, Xiaoyang Gao, Robert Harrison, So Hirata, Sriram Krishanmoorthy, Sandhya Krishnan, Chi-Chung Lam, Marcel Nooijen, Russell Pitzer, J. Ramanujam, P. Sadayappan, and Alexander Sibiryakov. Synthesis if high-performance parallel programs for a class of ab initio quantum chemistry models. *Proc. of the IEEE*, 93(2):276–292, February 2005. Invited Paper. [pdf].

[3] V. Choppella and C. T. Haynes. Source-tracking Unification. In Franz Baader, editor, *Proceedings of 19th International Conference on Automated Deduction, CADE-19, Miami Beach, USA*, number 2741 in Lecture Notes in Artificial Intelligence, pages 458–472. Springer, 2003. Superceded by [4]. [pdf].

[4] V. Choppella and C. T. Haynes. Sourcetracking unification (revised and extended version). *Information and Computation*, 201(2):121–159, September 2005. Invited Submission. Supercedes [3]. [pdf].

[5] Venkatesh Choppella. *Unification Source-tracking with Application to Diagnosis of Type Inference*. Ph.D. thesis, Indiana University, August 2002. URL `http://www.cs.indiana.edu/cgi-bin/techreports/TRNNN.cgi?trnum=TR566`. IUCS Tech Report TR566. [pdf].

[6] Venkatesh Choppella. Polymorphic type reconstruction using type equations. In Greg Michaelson Phil Trinder and Recardo Pe na, editors, *Implementation of Functional Languages: 15th International Workshop, IFL 2003, Edinburgh, UK*, volume 3145 of *Lecture Notes in Computer Science*, pages 53–68. Springer Verlag, December 2004. ISBN: 3-540-23727-5. [pdf].

[7] Venkatesh Choppella, Atanas Rountev, Gerald Baumgartner, and P. Sadayappan. Variable-free canonical forms for tensor contraction expressions. Under preparation, 2008.

[8] Venkatesh Choppella, Arijit Sengupta, Ed Robertson, and Steven Johnson. Prelimary explorations in specifying and verifying entity-relationship models in pvs. In Natarajan Shankar and John Rushby, editors, *Proceedings of AFM'07: Second ACM workshop on Automated Formal Methods*, pages 1–10. ACM Press, November 2007. [pdf].

[9] T B Dinesh. Pantoto open source content management system. `code.google.com/p/pantoto-lite`, 2008. Last visited 2008-10-12.

[10] Robert Bruce Findler and Matthias Felleisen. Contracts for higher-order functions. In *Int'l Conf. on Functional Programming (ICFP'02)*, pages 48–59. October 2002. URL `citeseer.ist.psu.edu/findler02contracts.html`.

[11] Daniel Jackson. *Software Abstractions: Logic, Language and Analysis*. MIT Press, 2006.

[12] Anneke Kleppe, Jos Warmer, and Wim Bast. *MDA Explained. The model driven architecture: practice and promise*. Object Technology Series. Addison-Wesley, 2003.

[13] S. Krishnan, S. Krishnamoorthy, G. Baumgartner, C-C. Lam, J. Ramanujam, P. Sadayappan, and V. Choppella. Efficient synthesis of out-of-core algorithms using a nonlinear optimization solver. In *Proc. International Parallel and Distributed Processing Symposium (IPDPS 2004), Albuquerque, New Mexico, USA*. IEEE Computer Society, April 2004. ISBN: 0-7695-2132-0. [pdf].

[14] Sandhya Krishnan, Sriram Krishnamoorthy, Gerald Baumgartner, Chi-Chung Lam, J. Ramanujam, P. Sadayappan, and Venkatesh Choppella. Efficient synthesis of out-of-core algorithms using a nonlinear optimization solver. *Journal of Parallel and Distributed Computing*, 66:659–673, 2006. Invited Submission. Supercedes [13]. [pdf].

[15] Aviv Regev and Ehud Shapiro. The π-calulus as an abstraction for biomolecular systems. In Gabriel Clobanu and Grzegorz Rozenberg, editors, *Modelling in Molecular Biology*, pages 219–266. Springer, 2004.

[16] K. R. Srivathsan, Venkatesh Choppella, Andreas Auer, Fernando Mendez, and Thomaskutty Sebastian. Direct democracy at the micro-level in india: The design and implementation of a civic participation platform, March 2008. Research proposal submitted to the Indo-Swiss Joint Research Programme. [pdf].

[17] J Srividhya and M. S. Gopinathan. A simple time delay model for eukaryotic cell cycle. *Journal of Theoretical Biology*, 241(33):617–627, 2006. Elsevier.

[18] Gregory V. Wilson. Where's the real bottleneck in scientific computing? *American Scientist*, 94(1), January-February 2006. http://www.americanscientist.org/issues/pub/wheres-the-real-bottleneck-in-scientific-computing last visited 2008-10-12.