Main Dossier
Application for Promotion/Tenure to Associate Professor

Abhik Roychoudhury
Assistant Professor
School of Computing
National University of Singapore
`abhik@comp.nus.edu.sg`
`http://www.comp.nus.edu.sg/∼abhik`

Contents

Chapter 1

Documents on Teaching

1.1 Teaching Philosophy

Effective teaching has often been associated with effective dissemination of knowledge leading to advanced understanding of a topic by the students. One can also extend this definition to include the notion of "conceptualization" – where the students not only understand a specific topic of a discipline, but also can extrapolate this understanding in other topics or in real-life problems. As an educator, I feel that effective teaching goes far beyond effective understanding and conceptualization of the students as learners. It involves **integration of the spirit of research into teaching and vice-versa**. What we mean by integration of research into teaching is a situation where the course instructor becomes a *serious learner* himself/herself and the teaching facilitates this process. So, by teaching a course, the instructor is spurred to conduct research on certain cutting edge topics which are later integrated into course(s).

As an example, let me refer to a graduate level module I have offered thrice. The title of this module is "Automated Software Validation". While teaching some traditional formal techniques for software validation, *some of the difficulties in using them for software debugging became clearer to me.* I came to fully realize that the user does not often have a clear specification of the "error" being checked – rather he/she only has an understanding that the behavior of a program for a particular input is unexpected. This inspired me to do research in semi-formal techniques on software fault localization with one of my Ph.D. students. Some of the understanding/experience gained from this effort has been integrated into a lecture in the third offering of the course.

The combined spirit of research by the teacher and students is, by no means, restricted to graduate courses. In the following, I outline some teaching methods which I employed in my *undergraduate teaching* for this purpose. It is worthwhile to mention here that in the last three years I have taught courses on formal verification or logic — topics which are relatively mathematical in nature. Of course an important task in the teaching of such courses is to relate these mathematical topics to the actual practice of computer system design. However, this relationship is often brought out in an extreme way (in textbooks or other teaching materials) — dramatic historical incidents which led to the spectacular failure of computer systems due to the lack of formal verification are presented to the students. This clearly attracts the students' attention, but this interest becomes difficult to retain, possibly because the historical incidents seem far removed. Instead, to motivate more mathematical topics like formal verification, I felt that the students need to *understand* the impact of formal verification in system design; they do *not* need to be surprised. So, in my *undergraduate course* on formal verification of embedded systems, I start with existing practices of verification/validation and how they are intrusive to the design process. I then discuss how a model-based formal approach can help the design cycle, without referring to historic disasters that occurred due to lack of formal verification. This results in a rather different pedagogical style, where the aim is to discuss the system design cycle with the students rather than surprising the students with the power of formal verification and logic. Once the learners appreciate the differences in the techniques, they develop a more long-lasting interest. This results in a learning process with more active participation from teacher/students — a goal which I always try to achieve in my classes. Furthermore, such classroom discussions have, from time to time, spurred my own research, the results of which have later been integrated into teaching.

As a concrete example, let me again refer to CS 4271, the undergraduate elective offered in the Computer Engineering programme at NUS School of Computing, which I have taught. The title of this module is "Critical Systems and their Verification", and it is meant to focus on formal techniques for design of embedded computing systems. When I started teaching this module, I found a lack of classroom examples through which I could elaborate some practical experiences in validating (substantial) parts of an embedded system. Around the same time, in the context of some research work in modeling reactive embedded systems I have been studying ARM's AMBA system-on-chip bus protocol. Working with a senior undergraduate student (whose final year project I was supervising), I figured that the AMBA protocol had some subtle corner cases leading to a deadlock which are hard to find using informal manual reasoning. This work was published in DATE 2003, a leading conference in embedded system/design automation. More importantly, it provided with a good hands-on case study which I could provide to my students in the CS 4271 course. It serves to illustrate exactly what role automated formal reasoning techniques can hope to play in practice for debugging subtle design errors. I have therefore, included this as a case study in later offerings of the CS4271 course.

In summary, my teaching methods are driven by an effort to cross-fertilize teaching and research. I have adopted these methods in classroom teaching as well as in other forms of teaching such as student supervision. As educators, I feel that **we should strive to break the walls between "teaching" and "research" in our own minds**. This can enhance a new spirit of excellence, allowing us to try and seek the best for our learners.

1.2 Teaching History

1.2.1 Courses Taught

Following are the modules I have been involved in during my six year stay at NUS.

- CS 4271 Critical Systems and their Verification
  (Semester 2 2001-02, Semester 1 2002-03, Semester 2 2003-04, 2004-05)
  Duties: Course Design (this was a new course), Lectures, Assessment, Projects
  Web-page: `http://www.comp.nus.edu.sg/~abhik/CS4271`
- CS 5219/6214 Automated Software Validation
  (Semester 1, 2003-2004, 2004-05, 2005-06)
  Duties: Course Design (this was a new course), Lectures, Assessment, Projects.
  Web-page: `http://www.comp.nus.edu.sg/~cs6214`
- CS 5201 Foundations of Theoretical Computer Science (since Semester 1 2005-06)
  Duties: This is the PhD qualifier examination in Theoretical CS for which I serve as the coordinator.
  Web-page: `http://www.comp.nus.edu.sg/~abhik/CS-QE/`
- CS 1102 Data Structures and Algorithms (Semester 2 2000-01)
  Duties: Lectures, Assessment
- CS 2104 Programming Language Concepts
  (Semester 1 2001-02, Semester 1 2002-03)
  Duties: Modification of course content, Lectures, Assessment
- CS 4272 Hardware Software Codesign (Semester 1 2006-07)
  Duties: Modification of course content, Lectures, Assessment, Lab set-up.
- Organization of Embedded Systems Seminar (Semester 1, 2003-2004)
  Duties: Giving general overview to students interested in Embedded Systems, organizing talks by students and faculty.

Among the above, I completely designed the CS4271 and CS5219/6214 courses. The module folders for these two courses are included in my teaching portfolio (attached). Following are some further details about these two courses.

- *CS 4271 Critical Systems and their Verification*
  It is a relatively new course offered by School of Computing. It is only one of the few electives currently offered for the Computer Engineering students. I was solely responsible for designing the entire course (curriculum, assessment scheme, lecture material etc). Since this course is associated with the Computer Engineering program, I made this course a hands-on course. During the lecture/consultation hours, I show the students some hand-on tricks in modeling and verification through

some online exercises (e.g. during the lecture itself we work out the modeling and verification of a substantial real-life protocol). Also, about one-third of the grade is based on performance of the students in a practical term project. The term project is done in groups of 2-3 people (bigger groups are required to take on bigger projects). A typical project involves formally specifying and verifying a substantial part of a real- life embedded system (e.g. a protocol or a hardware unit). To train the students appropriately for the project, I use the following schedule.

  - 1st month: I present a comprehensive list of potential projects to the students. Projects are fixed for each student.
  - 2nd month: Students finish the formal specification work and present an interim project report.
  - 3rd month: Most of the experimental work on formal verification is done; final submission.

- *CS 5219/6214: Automated Software Validation*
  This is another new course offered by School of Computing for graduate students. The purpose is expose students to sophisticated software debugging and program understanding techniques which go beyond traditional simulation and testing. I have designed the course to cover a wide variety of validation techniques which include various static checking techniques (theorem proving, model checking and static analysis). The techniques are discussed and explained in a concrete setting with lots of emphasis on a term project. Since this is a graduate course, the students here are required to perform the projects individually.

1.2.2 Student Supervision and Mentoring

Current Supervision Currently I am supervising the following six Ph.D. students. I am also guiding three undergraduate students for their final-year projects.

- Tao Wang, (Ph.D. student since 2002, expected to finish in 2007), Sole Supervision, *Dynamic Analysis Techniques for Software Fault Localization*
  Awarded **Microsoft Research Asia Fellowship** for his work in the year 2004-05.

- Ankit Goel, Ph.D. student since 2003, Sole supervision, *Interacting Process Classes: A model of computation for reactive embedded systems.*

- Vivy Suhendra, Ph.D. student since 2004, Co-supervised with Tulika Mitra, *Memory Optimizations for Developing Predictable Software for Real Time Embedded Systems*, Awarded **Microsoft Research Asia Fellowship** for her work in the year 2006-07.

- Liang Guo, Ph.D. student since 2005, Sole supervision, *Traceability in model driven embedded software development*

- Lei Ju, Ph.D. student since 2005, Co-supervised with Samarjit Chakraborty, *Model-driven timing analysis of embedded software.*

- Shanshan Liu, Ph.D. student since 2006, Sole Supervision.

Past Supervision So far, I have supervised one Ph.D. student, four M.Sc. students and several undergraduate students.

- Xianfeng Li, Ph.D. (graduated Dec 2005), Co-supervised with Tulika Mitra, *Micro-architectural modeling for Timing Analysis of Embedded Software.*
  First Employment: CS Department, Beijing University (China).

- Tuan-Anh Tran, M.Sc. (Graduated 2005), Co-supervised with P.S. Thiagarajan, *Generation of Protocol Converters from Scenario-based Specifications*
  First Employment: Friar Tuck Pte Ltd (Singapore).

- Qinghua Shen, M.Sc. (Graduated 2004), Co-supervised with Tulika Mitra, *Multi-threaded Java from Multi-processor Perspective*
  First Employment: Creative Technology Ltd (Singapore).

- Hemendra Singh Negi, M.Sc. (Graduated 2004), Co-supervised with Tulika Mitra, *Two Concrete Problems in Worst-Case Execution Time Analysis*
  First Employment: Mentor Graphics, New Delhi (India).

- Lei Xie, M.Sc. (Graduated 2003), Sole supervision, *Performance Impact of Multi-threaded Java Semantics on Multiprocessor Memory Consistency Models.*

- *Undergraduate Students* Guided several bright undergraduate students for their final year projects, some of which have led to research publications in established conferences (such as DATE'03). Details appear in my full CV.

1.2.3 Participation in thesis and oral examination committees

I have served as the thesis evaluator of several M.Sc. students from NUS, namely: Xu Na and Kamrul Hasan Talukder.

I have been on the thesis committee of several PhD students from NUS, namely: Zhu Ping, Mihail Asavoae, Kathy Nguyen Dang, Corneliu Popeea, Sim Joon, Chen Chunqing, Sun Jun, Hamid Abdul Basit.

I have also served as the external Assessor of the following PhD thesis — "A formal framework for a service oriented multi-agent society" by Manas Ranjan Patra (University of Hyderabad, India).

1.3 Teaching Performance Indicators

Designing new modules I have proposed and designed one new graduate module called *Automated Software Validation* which covers different verification methods (model checking, theorem proving) for software validation. In addition, I have also designed the entire course contents of an undergraduate module (*Critical Systems and their Verification*) which part of the Computer Engineering programme at School of Computing. *Module folders for both of these courses are included in my teaching portfolio.*

Non-traditional Curriculum Development Activities The reputation of a department or university rests not only on its research output, but also what the future employers say about the students we produce. With this in mind, since 2002 I have been actively involved in formulating a new written PhD Qualifier Examination in Computer Science at NUS School of Computing. This involved (a) deciding on set of core topics for examination (b) deciding on type and level of difficulty of questions and (c) setting sample question papers. In addition, since a formal Qualifying Exam is new to NUS School of Computing, the entire process of setting up the written Qualifier Examination was an intense activity. It involved consultation/discussion with many colleagues regarding (a) why a written Qualifier Exam is needed, (b) what is the desired format/level of difficulty of the exam and (c) assessment policies and administration of the exam. Our written Qualifier Exam consists of two papers — one in Theory and the other in Computer Systems. Since the inception of the written Qualifier Exam in 2005, I have **served as the coordinator of the Theory paper in the Qualifier Examination**. The exam is offered twice a year. In each offering of the exam, my duties involve meeting students to clarify their queries about the exam, finding examiners to set questions and moderate the level of difficulty of the questions. I maintain a web-page for the Computer Science Theory Ph.D. qualifier which provides more details about the exam — `http://www.comp.nus.edu.sg/~abhik/CS-QE`

Student Supervision While supervising undergraduate and graduate students, I have given equal emphasis to developing technical excellence and developing strong communication skills. So far, one of my Ph.D. students (Xianfeng Li) has graduated and joined the Computer Science Department of Beijing University, the top university in China. **Two of my Ph.D. students have been awarded the prestigious Microsoft Research Asia fellowship** for their research work (Tao Wang — 2004-05, Vivy Suhendra — 2006-07). This fellowship is awarded to PhD students in Asia on a competitive basis.

Pedagogical Article on Teaching Apart from classroom teaching and student supervision, I have participated in workshops with educators from other universities to reflect on my experiences from teaching. *A summary of some of my teaching methods appears in a recent paper* entitled "Introducing Model Checking to Undergraduates". This paper was presented at the Formal Methods Education Workshop 2006 at Toronto. The

paper can be accessed from `http://www.comp.nus.edu.sg/~abhik/pdf/fm-ed06.pdf`
It is also included in my teaching portfolio (attached).

Use of novel software tools for teaching I have used software tools developed in my research for teaching undergraduate courses. The Chronos software performance estimation tool was developed as part of my research in 2002 - 2004. I am currently using it for laboratory assignments in the undergraduate course *Hardware Software Co-design*, an elective in our Computer Engineering programme.

Participation in Development Programmes In 2001, I completed the Professional Development Programme (PDP) offered by the Center for Development in Teaching and Learning (CDTL) to incoming NUS teaching staff.

1.4 Future Plans in Teaching

I plan to work on further improving my teaching in the future years. Concretely, I plan to teach new courses, fine-tune the PhD qualifier examination (of which I currently serve as a coordinator), and participate more actively in discussions/symposium on pedagogy in my area.

New Courses In the current semester, I am teaching a new course — *Hardware Software Codesign*, an undergraduate elective module in our Computer Engineering programme. The contents of this module is different in nature from courses in programming languages and formal verification that I have taught in the past. The Hardware Software Codesign module is more focused on system design, rather than formal reasoning/analysis. This is a new challenge for me, one from which I hope to learn as well as contribute.

PhD Qualifier Examination I plan to be involved in the fine-tuning of the Computer Science Department's PhD Qualifier Examination on Theory. I am currently coordinating this examination and I want to discuss more with other colleagues about the standard of questions, existing/requisite background of our incoming students and how we can try to supplement certain deficiencies in background knowledge.

Participation in Pedagogical Discussions Recently, I participated in the Formal Methods Education Workshop at Toronto. The aim was to discuss with fellow educators the issues in teaching formal methods and logic to Computer Science students. In future, I plan to spend more energies in this direction. I am keen to try out mechanisms for highlighting the value of formal methods (in system design) to Computer Science students.

Chapter 2

Documents on Research

2.1 Research Programme

Developing models and methods for building reliable software is of obvious importance. It is more so, with software controlling many devices of everyday use (such as handphones) and safety-critical applications (such as in automotive and avionics domains). My research focuses on developing reliable software by employing formal mathematical techniques at the various stages of the design life cycle — requirements, intermediate representations, and system implementations. The work includes formal analysis/validation of software functionality as well as timing behavior.

2.1.1 Overall Goals

Reliable software/system development is a key challenge in computing and information sciences. Indeed a 2003 conference series on Grand Challenges in Computer Science and Engineering organized by the Computing Research Association (CRA) lists dependable system development as one of the five key challenges for the future. However, the problem in this area is one of scale and applicability — the techniques for developing reliable software often do not scale up to real-life systems. In the last six years, I have conducted research on building dependable embedded systems via formal modeling, analysis and debugging. The overarching goal is to develop techniques which are scalable and give increased confidence about system behavior at different levels of abstraction — models, software and assembly code.

Specifically, in the last six years, I have concentrated on the following three directions.

- Requirements for Embedded System Modeling and Design
- Validation and debugging of software functionality
- Platform-aware timing analysis of embedded software

The work on requirements modeling focuses on behavioral system requirements at the early stages of (embedded) system design. The work on software debugging proceeds by analyzing a program or a program trace at the level of generic programming languages like C or Java. The focus is on functionality validation. The work on software timing analysis proceeds at a lower level since the underlying hardware behavior is also taken into account while analyzing the execution time of (embedded) software.

The three directions cut across the various layers of abstraction involved in system design. This has also helped me understand better the problems that arise in software/systems validation. In the following, I outline the significant research accomplishments.

2.1.2 Major Accomplishments

At the system requirements, my work has primarily focused on developing formal requirements on system *behavior*. In particular, we have developed formal executable models for high-level design of distributed embedded systems.

Scenario-based System Requirements modeling In this area, I have concentrated on developing behavioral requirement specifications of reactive embedded systems. Our modeling notations are compatible with the well-known Unified Modeling Language or UML. Specifically, I have studied modeling and simulation of distributed embedded systems with many similar interacting components — common in application domains like automotive control, avionics, telecommunication and transportation [1]. We have developed symbolic methods which can simulate and analyze such systems containing large number of *similar* interacting components with minimal loss of precision. The key contribution is to develop (a) formal models which emphasize inter-component communication (scenario-based system descriptions) and (b) symbolic execution semantics to efficiently execute those models. Thus, we exploit the behavioral similarity of components within a system to support efficient execution of our system models (for purposes of test generation and behavioral simulation). Our symbolic simulator has been used to debug realistic designs of controllers from avionics and automotive domains. In particular, we have used it to model/analyze communication protocols in automotive control systems such as the Media Oriented Systems Transport (MOST) protocol used by communicating media devices in a car network.

Apart from developing novel scenario-based design models, we have also studied related issues in (a) model synthesis [2], (b) model-based test generation, and (c) the connection between formal behavioral design models (such as Sequence Diagrams or Live Sequence Charts) and software analysis (in particular dynamic analysis methods) [3, 7].

The system requirements are useful in the early stages of embedded system design. In reality, it is not feasible to perform all of the analysis/validation at the requirements/model level simply because the requirements of a system evolve over time. Consequently, it is also important to study analysis/validation methods for software debugging, a topic that I have followed up in my research.

Analysis Methods for Software Debugging In this area, I have studied various problems involving program comprehension/debugging. Since debugging often proceeds by trying out test cases of a program, it is important to study debugging methods which proceed by analyzing the program trace(s) or execution run(s). In particular, with my Ph.D. student Tao Wang, I have developed methods and tools for dynamic slicing [10]. Dynamic slicing is a generic technique for program comprehension which highlights parts of a program responsible for an observable "bug". It forms the core of many program development tasks such as understanding the cause of software performance degradation and software verification. To the best of our knowledge, our *Jslice* tool is the only tool for dynamic slicing for the Java programming language. Recently, we have also investigated debugging methods which select and compare execution runs of a program for software fault localization [4].

Part of my research concentrates on software model checking or lightweight theorem proving techniques. In particular, I have studied the role of programming language memory models for software model checking [5, 8]. In my Ph.D. work, I had studied formal techniques for proving invariant properties of parameterized protocols/software — where the number of processes/threads is unbounded. We developed an automated inductive proof technique which combines model checking and lightweight induction. In this work, the protocol/program's behavior is encoded as a logic program. This logic program is *automatically transformed* so that the recursive structure of the transformed program provides the schema/plan for the induction proofs.

In reality, validation methods for software systems should not be restricted to checking functionality properties alone. The correctness of many computing systems depends on the timing of its constituent operations. This is particularly so for embedded systems where many systems are also real-time, that is, they operate under tight timing constraints imposed by the environment they operate in. In my research, I have developed analysis methods/tools for estimating the execution times of programs. The results from this research forms the core of system level timing analysis and validation methods.

Software Timing Analysis In this area, I have studied static program analysis techniques for providing upper bounds on the execution time of a given program on a given platform. A key practical difficulty in developing such analysis methods is the abundance of performance-enhancing micro-architectural features in today's processors. Consequently, the analysis for estimating the maximum execution time of software cannot be blind to the hardware; the timing effects of the underlying processor need to be taken into account. We have developed static analysis methods for estimating a program's execution time in presence of processor micro-architectural features like out-of-order pipelines, cache and branch prediction [6]. The main difficulty arises from the large number of possible system states when the micro-architectural features are taken into account. We propose a novel event-based fixed-point analysis method which avoids such state space explosion and still provides accurate execution time estimates.

2.2 Publications and other Research Outcomes

2.2.1 Full List of Research Outcomes

Publications My work has been published in prestigious journals and conferences such as ACM Transactions on Programming Languages and Systems (TOPLAS), Real-time Systems Journal, ACM/IEEE International Conference on Software Engineering (ICSE), Intl. Symposium on Formal Methods (FM), Intl. Conf. on Computer Aided Verification (CAV), ACM Design Automation Conference (DAC) and IEEE Real-time Systems Symposium (RTSS). *We note here, that conferences form the primary publication venues in Computer Science; they are peer-reviewed, highly regarded and extremely competitive (with only 10-20% of papers accepted).*

A full list of my publications from the NUS Publication System is attached. The publication list also appears in my *detailed CV* (attached) along with the acceptance rates (ratio of paper submissions to accepted papers) for the conference papers.

Patent My work on deductive proof methods has led to a *US Patent* (US patent 6343372, awarded January 2002). It deals with the synthesis of verified-correct program abstraction methods from the correctness proofs of transformations between concrete and abstract programs.

Tools developed As part of my research, me and my students have released several software analysis tools for use in research, teaching and development. *One of them is already being used for lab assignments in a fourth-year undergraduate course.* In particular, the following are mature program analysis tools for software written in general-purpose programming languages like C or Java.

- *Jslice*: Jslice is a dynamic slicing tool for Java programs. Given a program and an input, it can explain the cause of "unexpected behaviors" in the program's execution by highlighting portions of the program responsible for those behaviors. Jslice is available as open-source software from `http://jslice.sourceforge.net/` It has been distributed with the widely-used Kaffe Virtual Machine at the request of Kaffe developers.

- *Chronos*: Chronos is a tool for estimating the maximum execution time of any C program on a given processor platform. It proceeds by static analysis of the program source code while taking the underlying processor hardware into consideration. Chronos is available as open-source software from `http://www.comp.nus.edu.sg/~rpembed/chronos`

Both of the above tools have been released recently (2005-06), so we do not have detailed statistics of their usage. However, we have been approached by researchers and developers for the usage of these tools. In particular, for the JSlice tool, we were approached by developers of the Kaffe Virtual machine for integrating and distributing our work with Kaffe. Currently, JSlice is distributed along with the widely-used *Kaffe Virtual Machine* for Java.

2.2.2 Five Selected Publications

Following are five major publications resulting from my research work in the past six years. These publications demonstrate how I have contributed in the related fields of formal system modeling (ICSE 2006), software analysis (ICSE 2004, RTSS 2004), and software verification (ICSE 2002, CAV 2001).

- *Interacting Process Classes, ICSE 2006.* [1]
  This work develops a novel modeling and simulation framework for distributed embedded systems with many similar interacting processes. The main contribution is to support time and memory efficient simulation of designs while allowing combination of rich modeling constructs compatible with the Unified Modeling Language (UML). The applications of this work lie in the design of distributed embedded systems with behaviorally similar processes such as a traffic controller controlling many cars/trains/aircrafts.

- *Using Compressed Bytecode Traces for Slicing Java programs, ICSE 2004* [10]
  This work develops methods for debugging Java programs based on analysis of program execution runs. The research has led to the *Jslice* dynamic slicing tool for Java programs — the first of its kind. This is useful, since dynamic slicing is a popular technique which forms the core of many program development/analysis methods. However, slicing tools for real-life programming languages (like Java) are relatively hard to come by. Our research in this paper aims to fill this gap.

- *Modeling Out-of-order processors for WCET Analysis, Real-Time Systems Journal, 2006* [6]
  This work develops static analysis methods for tightly estimating the maximum execution time of a program when the program is executing on a processor with advanced micro-architectural features. The work had led to the *Chronos* Worst-case Execution time estimation tool for C programs, which can be used for scheduling and performance estimation of real-time/embedded software. The typical applications are in safety-critical embedded software from application domains like automotive and avionics. A detailed version of the paper has appeared in the *Real-time System Journal* in 2006.

- *A Memory Model Sensitive Checker for C#, FM 2006* [5]
  Reasoning about concurrent program behaviors is hard. In this work, we study how to accurately capture all possible behaviors of a program in a modern multi-threaded programming language such as C#. We show that current verification techniques (mistakenly) assume a strong execution model called "Sequential Consistency", which may result in a verified correct program exhibiting violations of the property verified! We then discuss how the semantics of multi-threading can be accurately described and incorporated into program verification. Even though the results are for C#, the ideas are generic and can be applied to other programming languages such as Java (a related work is [8]).

- *Automated inductive verification of Parameterized Protocols, CAV 2001* [9]
  Many distributed systems can be described as unbounded collections of similar processes. In this work, we prove properties about such systems by employing a mix of automated methods (search) and mathematical induction. The main conceptual novelty comes from automating these induction proofs by gleaning the induction schema directly from the system description; logic program manipulation techniques are used for this purpose. A large part of this research was done during my Ph.D. work at SUNY Stony Brook.

2.2.3 Contributions in co-authored publications

All my co-authored publications involve substantial contributions from me and my co-authors. Whenever a student co-author is involved, his/her name typically appears first in the list of authors.

2.3 Research Performance Indicators

Publications are one important measure of research contributions. Apart from publications, impact of research can be evidenced through other means — such as citations and invited talks. Here I outline some of these indicators.

2.3.1 Research Citations

In the following, I present a brief analysis of my research citations. All citation data has been extracted from Google Scholar. For each paper, *the self-citations have been excluded.*

Total Citations: The total number of citations (excluding self-citations) is around 250.

Top cited papers: Following are the papers co-authored by me which have the maximum number of citations. The papers in ICSE 2002, CODES+ISSS 2003, and DATE 2003 were written during my stay at NUS

- Logic Programming and Model Checking, PLILP/ALP 1998, 35 citations
- Accurate Estimation of Cache-related Pre-emption Delay, CODES+ISSS 2003, 21 citations
- Specifying Multithreaded Java Semantics for Program Verification, ICSE 2002, 20 citations
- XMC: A Logic Programming based Verification Toolset, CAV 2000, 19 citations
- Using formal techniques to debug the AMBA system-on-chip bus protocol, DATE 2003, 18 citations

Nature of citations Apart from the raw citation numbers, the context/fashion in which a paper is cited is also indicative of how the research result is received or used by other researchers. In this context, I present some recent citations of one of my works done at NUS. In particular, my work on program verification by considering the Java memory model has been useful in the context of understanding program behaviors in the presence of complex memory consistency models. Here are some citations giving detailed description of the work.

- "Roychoudhury and Mitra [25] suggested a formal specification for the Java memory model (JMM) using guarded commands, and apply their specification when analyzing multithreaded Java programs. Their work is focused on the JMM but it can be tuned to other weak memory consistency models. Our approach can be extended to support their formal specification."

  — "Scaling Model Checking of Dataraces using Dynamic Information",
  Shacham, Sagiv and Schuster, PPoPP 2005.

- "The Java Memory Model [Gosling et al. 1996; Roychoudhury and Mitra 2002] is equivalent to entry consistency because locks are associated with objects,."

  —- "A Unified Theory of Shared Memory Consistency",
  Steinke and Nutt, Journal of ACM, 2004.

- "Roychoudhury and Mitra suggest that there are three approaches to tackle this problem [30] i.e. They opt for the third approach because . However, we can still settle on the first approach, "

  — "Predictable Memory Utilization in the Ravenscar-Java profile",
  Kwon, Wellings, King, IEEE International Symposium on Object-Oriented Real-Time Distributed Computing ISORC-2003.

2.3.2 Invited Papers and Presentations

For my work on UML-based system requirements modeling and analysis, I was most recently **invited to write/present a paper** at the *International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA) 2006* [Track on Highly Reliable Software]. ISoLA primarily looks at practical applications of formal verification methods; it forms a meeting ground for researchers and users of formal methods to get together and study adoption of formal methods into practice. I have also given several other invited presentations of my work on UML-based requirements modeling, which includes the following venues.

- **Invited full-day tutorial** at the Intl. Conf. on Petri Nets and Other Models of Concurrency (ICATPN) 2005, USA (jointly with P.S. Thiagarajan).
- Workshop on Predictable Software Component Assembly, University of Manchester, UK, September 2005.
- Workshop on Formal Methods for Design and Analysis of Software, Microsoft Research, October 2005.
- Invited by IEEE chapter of IIT Kharagpur (India) for a presentation — January 2006.
- Presentation at United Nations University - International Institute of Software Technology (UNU-IIST), Macau, June 2006.

For my work on timing analysis of embedded software, I have been **invited to write a book chapter** in the Second Edition of the widely-used *Compiler Design Handbook* (jointly with Tulika Mitra). I gave an invited presentation on this topic at the University of Florida, Gainesville in June 2005. I have also given contributed tutorials on this topic at the Intl. Conf. on Formal Engineering Methods (ICFEM) 2005, and IEEE Intl. Conf. on VLSI Design (VLSI) 2007.

For my PhD work on combining lightweight inductive proofs with automated reasoning, I was **invited to the *International Conference on Logic Programming* (ICLP) 2002 for giving a tutorial** (see `http://floc02.diku.dk/ICLP/` for details). I was one of only four invited tutorial speakers at ICLP 2002 (one speaker for each day of the conference). The focus of this tutorial was to discuss how logic program evaluation/transformation can be flexibly used to automatically reason about distributed systems. I was also **invited to write a chapter** in the book "*Program Development in Computational Logic*" (published 2004), which summarizes the last ten years of development in this field.

2.3.3 Research Grants

Apart from several grants from within the faculty (School of Computing) and the university (NUS URC projects), I have attracted substantial funding from outside the university. My research has been supported quite steadily by government agencies as well as industry. *A full list of research grants appears in my detailed CV.* In the following, I highlight the following three research grants supported by industry, government and special initiatives; I have served as the Principal Investigator of these three grants. In addition to these three grants, I have also been the PI of several faculty-level internal grants which are not mentioned here.

- My research on software validation and debugging methods has been supported by the following **Public Sector research grant from Agency of Science Technology and Research (A*STAR)**. Apart from publications in prestigious venues, the output from this research project includes some large-scale software productivity tools which can be useful in research and development.

  Tools and Techniques for Model-based Software Debugging
  PI: Abhik Roychoudhury, Co-PI: P.S. Thiagarajan,
  2004-2007, Budget: S$ 361,648

- As far as **industry funding** is concerned, I recently received a small research grant from Microsoft. The work done in the project led to increased understanding of "corner-case" behaviors of C# programs and the development of a prototype verifier at the bytecode level [5]. Following are the details of the grant.

  Correctness and Performance Issues in the CLI Memory Model
  PI: Abhik Roychoudhury, Co-PI: Tulika Mitra, Weng-Fai Wong
  2005-2006, Budget: S$ 24,000

- I have served as the Principal Investigator (PI) of a **multi-disciplinary research grant** supported by the NUS Infocomm and Infotech Initiative (ICITI). The project was a collaboration with NUS Science Faculty.

  Efficient Design Space Exploration of Embedded Systems
  PI: Abhik Roychoudhury
  Co-PI: Biman Chakraborty (NUS Dept of Statistics)
  Tulika Mitra (NUS Dept of Computer Science)
  2003 -2006, Budget: S$ 75,000

In addition to the above projects where I served as the Principal Investigator (PI), I have been a Co-PI of **major projects from the Agency of Science Technology and Research (A*STAR)**. This includes the following two large projects funded under A*STAR's Embedded and Hybrid Systems Programme.

- *EASEL: Engineering Architectures and Software for the Embedded Landscape*, 2006-2009, Budget: **S$** 1.4 **million**

- *Formal Design Techniques for Reactive Embedded Systems*, 2003-2006, Budget: S$ 429,000

2.3.4 Conference Committees and Paper Reviewing

- Served on the Program Committee of following conferences
  - Intl. Conf. on Formal Engineering Methods (ICFEM) — 2003, 2005, 2006
  - Intl. Symp. on Logic based Program Synthesis and Transformation (LOPSTR) — 2004, 2005
  - ACM Intl. Symp. on Applied Computing (SAC) 2006
- Started the International Workshop on Software Verification and Validation (SVV) 2003/04/05/06, a new workshop on software validation.
- Track Chair for Software Engineering in the International Conference on Distributed Computing & Internet Technology (ICDCIT) 2004, Proceedings published as LNCS 3347, Springer Verlag.
- Reviewer of papers in many international journals
  - ACM Transactions on Architecture and Code Optimization (TACO), ACM Transactions on Embedded Computing Systems (TECS), ACM Transactions on Programming Languages and Systems (TOPLAS), Formal Aspects of Computing Journal (FACJ), Theory and Practice of Logic Programming (TPLP)

  and conferences
  - CONCUR, FM, FST& TCS, ICALP, ICLP, LCTES, POPL, PLDI, RTSS, VMCAI.

2.4 Future Plans in Research

My current as well as future work concerns models and analysis for embedded software development. In safety-critical application domains such as avionics, automotive, healthcare monitoring, and command-control systems — developing reliable bug-free embedded software is of utmost concern. Consequently, there is increased focus on model-driven software development as well as software certification for embedded systems. In my current research, I have covered various modeling and analysis techniques which can aid the development of embedded systems for safety critical application domains. This includes various combinations of — symbolic execution, model checking, static program analysis, and dynamic analysis. More importantly, I have looked at the application of these methods at various stages of embedded system design — requirements, design models and hardware/software implementations.

In future, I plan to build on my current work by bringing tighter connections between the different stages of model-based design for embedded systems. Currently, model based design flows proceed from (informal) requirements to (semi-formal) design models; subsequently, the design models are converted to implementations in a (semi)-automated fashion. There is now an enhanced understanding on the importance of propagating both functional and extra-functional constraints (such as timing constraints) from requirements level to the implementation level. Central to this work is the concept of model transformations — where requirement models get translated to design models which further get translated to implementation models. Indeed, this issue is being taken seriously even in industrial tools. In the following, I discuss some long term plans which relate to this research direction.

In the longer term, I am studying the possibility of developing *bi-directional links* between requirements, design models and implementations in the context of embedded system design. Currently, system design focuses on generating design models from requirements, and system implementation from design models. However, as most practitioners will agree, it is often impossible to get a complete set of system requirements at one go. Consequently, it is difficult to generate correct-by-construction designs and implementation at one go. This makes it difficult (or even impossible) to perform all of the validation at the model level and none at the software level. Now, when we perform testing/debugging of the software implementations, if functional errors are found — it very hard to relate back the error to the model/requirement level. Currently, there exist number of commercial tools for (semi)-automatically generating software implementations from system models specified in the Unified Modeling Language (UML). However, the association between models and code is still largely a forward one. Even for state diagrams (which form the key executable part of UML) many of the model elements cannot be retrieved once C/C++/Java code has been generated. Such backwards communication between software and models can be crucial for communicating changes/fixes in the software to the embedded system designer or even other stakeholders. This is particularly important from a pragmatic point of view since in reality embedded system design (currently) involves lot of low-level optimizations at the implementation level.

Associations between Code and Design Models In future, I plan to study the issue of *backward association between system implementations and design models.* We need to study whether $software \rightarrow model$ backward associations can be seamlessly integrated into UML design tools, particularly those which proceed by (forward) transformation of design models. Clearly, we also need to study the nature of models suitable for such backward associations – whether we take UML state diagrams or richer models with UML-compatible notations. The ability to relate code elements with model elements must also be seamlessly integrated with various code-level debugging methods — model checking, slicing, fault localization or even performance debugging (using execution-time analysis). This will be an important step in supporting model-driven embedded software development for safety-critical applications.

Moreover, in the context of embedded system designs, one could study *extra-functional artifacts* — such as critical paths in the design model (possibly given as StateCharts) which contribute to the maximum execution/response time of the corresponding software implementation. From the model-level performance analysis, we can generate representative critical paths or system scenarios which can be fed to software level performance analyzers. Seamlessly integrating model-level and lower-level analyzers to provide timing guarantees of embedded control software is an important problem, one which I plan to investigate.

Associations between Design Models and Requirements Finally, as a more ambitious goal, we can try to relate design models to (informal) requirements thereby completing the backward link between software, designs and requirements. For many safety critical domains, the informal requirements are given as a collection of temporal properties. It has been observed (by researchers working in Live Sequence Charts or LSCs) that such requirements can be readily visualized as LSCs. We feel that it might be important to relate LSCs (which are completely scenario-based) with hybrid design models which provide a mix of state based and scenario-based descriptions (such as our recent work [1, 2]). The hybrid design models can then be used for test/code generation while still maintaining a link (in terms to scenarios) to the informal requirements which are visualized as Live Sequence Charts.

Chapter 3

Documents on Service

Service to University/Faculty I have served as the Assistant Professor Representative in the School of Computing Executive Committee in the year 2002-03. I have also served as a member of the School of Computing Graduate Studies Committee since 2003. As a member of this committee, I have been involved in evaluating and interviewing graduate applicants from South Asia. I have also participated in two visits to Indian universities for recruiting top students into our graduate programme. The details of these visits are as follows.

- Visit to Indian Institute of Technology (IIT) Mumbai — November 2002.
- Visit to Indian Institute of Technology (IIT) Guwahati, Bengal Engineering & Science University and Jadavpur University — March 2005.

Service to International Academic Community I have started a new workshop focussing on software validation, the International Workshop on Software Verification and Validation. It has been held for four consecutive years. In 2006, it was co-located with the Federated Logic Conferences (FLoC) at Seattle. I have also served as a program committee member of a number of conferences, and a reviewer for many journals/conferences. Details of these services are listed in the *Documents on Research* chapter (see Page 18) and hence these are not repeated here.

Service to Profession/Industry I have been a member of the Scientific Committee of National Olympiad in Informatics (NOI) for three consecutive years – 2002 - 2004. NOI is a creative problem solving / programming competition for High School / JC Students. Selected candidates from NOI represent Singapore in the International Olympiad in Informatics" (IOI). As a member of the NOI Scientific Committee, I was involved in formulating questions and model answers for the competition.

Bibliography

[1] A. Goel, S. Meng, A. Roychoudhury, and P.S. Thiagarajan. Interacting process classes. In *International Conference on Software Engineering (ICSE)*, 2006.

[2] A. Goel and A. Roychoudhury. Model synthesis: from informal requirements to formal scenario-based models. In *Intl. Symp. on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA)*, 2006.

[3] A. Goel, A. Roychoudhury, and T. Mitra. Compactly representing parallel program executions. In *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2003.

[4] L. Guo, A. Roychoudhury, and T. Wang. Accurately choosing execution runs for software fault localization. In *Compiler Construction (CC)*, 2006.

[5] T.Q. Huynh and A. Roychoudhury. A memory model sensitive checker for c#. In *International Conference on Formal Methods (FM)*, 2006.

[6] X. Li, A. Roychoudhury, and T. Mitra. Modeling out-of-order processors for WCET analysis. *Real-time Systems Journal*, 2006.

[7] A. Roychoudhury. Depiction and playout of multi-threaded program executions. In *IEEE/ACM Conference on Automated Software Engineering (ASE)*, 2003.

[8] A. Roychoudhury and T. Mitra. Specifying multithreaded Java semantics for program verification. In *Intl. Conf. on Software Engineering (ICSE)*, 2002.

[9] A. Roychoudhury and I.V. Ramakrishnan. Automated inductive verification of parameterized protocols. In *Computer Aided Verification (CAV)*, 2001.

[10] T. Wang and A. Roychoudhury. Using compressed bytecode traces for slicing Java programs. In *ACM/IEEE International Conference on Software Engineering (ICSE)*, 2004.