Jubi Taneja, Ph.D.

jubitaneja@gmail.com linkedin.com/in/jubitaneja github.com/jubitaneja
www.jubitaneja.com Research Engineer, Microsoft Research

RESEARCH INTERESTS

- AI for Software Engineering, AI for Code Generation, Compilers, Compiler Optimization, Formal Verification

EXPERIENCE

- **Research Engineer at Microsoft Research** July 2021 - Present

  PTX Kernel Equivalence Checker

  Building a PTX-level equivalence checker that uses symbolic execution to formally verify whether optimized GPU kernels—generated by LLMs or handwritten by compiler experts—preserve the semantics of their unoptimized counterparts. This work advances trustworthy AI-assisted codegen by enabling functional equivalence checking at the assembly level.

  LLM-Vectorizer: LLM-based Verified Loop Vectorizer

  Developed a framework that prompts large language models (LLMs) to rewrite scalar loops into vectorized form, and then formally verifies the transformation using SMT solvers.

  Paper: `https://dl.acm.org/doi/10.1145/3696443.3708929`

  Accera: Compiler to accelerate AI workloads

  Contributed to Accera, a domain-specific compiler designed to optimize compute-intensive algorithms for both CPU and GPU targets. The system lowers a Python-based scheduling DSL to MLIR, where a series of transformations and lowerings are applied before generating LLVM IR and machine code. I extended the DSL with features to express loop scheduling optimizations and implemented custom transformations across the MLIR and LLVM IR stages to enable high-performance code generation for diverse hardware backends.

  Repo: `https://github.com/microsoft/Accera`

- **Graduate Research Assistant at University of Utah** Aug 2013 - May 2021

  Dissertation: Improving Compiler Construction Using Formal Methods
  Programming languages are becoming more high-level and processors are moving to- wards more specialization. This trend has increased the gap between high-level program- ming languages and low-level processors. It is the responsibility of a compiler to bridge the gap between the two. The key contribution of this dissertation is to use formal techniques to build parts of the compiler to make them more effective and more correct. Static analyses compute properties of programs that are true in all executions. Compilers use these properties to justify optimizations such as dead code elimination. Each static analysis in a compiler should be as precise as possible while remaining sound and being sufficiently fast. Unsound static analyses typically lead to miscompilations, whereas imprecisions typically lead to missed optimizations. Neither kind of bug is easy to track down. My contributions include a collection of algorithms that use an Satisfiability Modulo Theories (SMT) solver to compute sound and maximally precise static analysis results for fragments of code in the LLVM intermediate representation, enabling automated testing of the corresponding static analysis code in LLVM. Some of the abstract domains that we target have convenient properties enabling efficient search for the most precise result, but one of them (integer ranges) appears to lack these properties; in this case we compute the most precise result using a CEGIS (Counterexample Guided Inductive Synthesis) loop. Our test cases include all of the LLVM IR generated while compiling SPEC CPU 2017. I found many imprecisions, some of which have resulted in patches being made to the LLVM compiler. My work is important because the correctness and efficiency of a large amount of

real-world code in a number of programming languages depends on the soundness and precision of static analyses in LLVM. Peephole optimizations are hard to get right as they involve many corner cases. They are hard to extend because developers are not using domain specific language to define the rewrite rules, automatically generate and verify them for soundness. My contributions include designing an automatic peephole optimizer that is driven by a superoptimizer. It makes use of a superoptimizer's intermediate representation to define the peepholes and transform them to another IR and it guarantees correctness as these optimizations are verified by a SMT solver.

PDF: https://github.com/jubitaneja/phd-dissertation

Testing LLVM's Static Analyses for Precision and Soundness
Developed a framework that uses SMT solvers to test the precision and soundness of static analyses in production compilers like LLVM. The system generates ground truth analysis results offline—too slow for production use but ideal for validating existing compiler analyses. This approach helped uncover numerous precision issues and rediscovered previously fixed soundness bugs that had regressed. The work bridges formal methods with practical compiler development, enabling more trustworthy and optimizable code.
Code: https://github.com/jubitaneja/souper-cgo20-artifact
Souper: Superoptimizer for LLVM IR
Souper is a synthesis-based superoptimizer for LLVM IR that automatically discovers optimal, verified peephole optimizations using SMT solvers. It encodes code fragments as logical formulas, searches for equivalent but more efficient replacements, and guarantees correctness through formal verification. The project advances compiler automation by improving optimization quality beyond what is hand-written in LLVM.
Code: https://github.com/google/souper

- **Research Intern at Mozilla** May 2018 - Aug 2018
  Automatic Generation of a PeepHole Optimizer
  Worked on using the superoptimization technique to automatically generate a peephole optimizer for the **Cranelift JIT compiler**, which compiles WebAssembly to machine code for x86 and ARM target architecture. Mentor: Nick Fitzgerald and Dan Gohman.

  Code: https://github.com/jubitaneja/cranelift-peepholes

- **Senior Software Engineer at Samsung R&D Institute,** India Sept 2011 - Aug 2013

  Worked in Linux Kernel Development Team. The project aimed at development of partial prelink technique to reduce the launch time of web browsing kits in Smart TV. Partial Prelink retains the design goals of Prelink without the need to update all dependent executable and shared libraries.

- **Junior Research Fellow at IIT Bombay,** India June 2010 - July 2011

  Worked on extending a generic dataflow analyzer (gdfa) in **GCC** that performs static analysis on programs on the basis of textual specifications. Advisor: Prof. Uday Khedker.

  Code: https://www.cse.iitb.ac.in/grc/index.php?page=gdfa

- **Research Intern at IIT Bombay,** India June 2009 - Nov 2009

  Worked on incremental compiler construction and development of specification based code generator generator (CGG). Advisor: Prof. Uday Khedker.

CONFERENCES/MANUSCRIPTS

- LLM-Vectorizer: LLM-based Verified Loop Vectorizer. Jubi Taneja, Avery Laird, Cong Yan, Madan Musuvathi, and Shuvendu Lahiri. CGO 2025. https://dl.acm.org/doi/10.1145/3696443.3708929

- ClassInvGen: Class-Invariant Synthesis using Large Language Models. Chuyue Sun, Viraj Agashe, Saikat Chakraborty, Jubi Taneja, Clark Barrett, David Dill, Xiaokang Qiu, and Shuvendu Lahiri. SAIV 2025. https://arxiv.org/pdf/2502.18917

- Testing Static Analyses for Precision and Soundness. Jubi Taneja, Zhengyang Liu, and John Regehr. CGO 2020. **Distinguished Paper Award** and **Best Student Presentation Award**. https://dl.acm.org/doi/10.1145/3368826.3377927
- Souper: A Synthesizing Superoptimizer. Raimondas Sasnauskas, Yang Chen, Peter Collingbourne, Jeroen Ketema, Jubi Taneja, and John Regehr. https://arxiv.org/abs/1711.04422 (2017).
- Improving Compiler Construction using Formal Methods. Jubi Taneja. The University of Utah. Jan 2022. https://dl.acm.org/doi/abs/10.5555/AAI30241036

INVITED/CONFERENCE RESEARCH TALKS

- LLM-Vectorizer: LLM-based Verified Loop Vectorizer Invited talk at UC Santa Cruz, Uber Technologies, PNW PLSE 2024, CGO 2025, LATHC 2025.
- Improving Compiler Construction using Formal Methods, invited talk at UC Berkeley Programming Systems Seminar group, and Microsoft Research. June 2020. Video: https://www.youtube.com/watch?v=de8Ak0nY1hA
- Exploiting and Improving LLVM's Dataflow Analyses using a Superoptimizer, Student Research Competition at LLVM Developers' Meeting, San Jose, CA, USA. October 2017. Video: https://www.youtube.com/watch?v=WyMTa2_yNHQ&t=526s

EDUCATION

Aug 2013 - May 2021	**University of Utah, Salt Lake City, Utah** PhD Student (Computer Science), GPA: 3.97 **Advisor: Prof. John Regehr**
July 2006 - May 2010	**University College of Engineering, Punjabi University, India** Bachelor of Technology (Computer Science), Score: 89.74% **(Gold Medal)**

TECHNICAL SKILLS

- Programming: Python, C++, Rust

ACHIEVEMENTS

- Received the Distinguished Paper Award and Best Student Presentation Award for the paper presented at CGO 2020.
- Received the University Gold Medal for standing first in order of merit in the CS undergraduate program.

SERVICES

- SIGPLAN-M Long-Term Mentor (2021-Present). Featured in PL Perspectives: https://blog.sigplan.org/2021/01/05/introducing-sigplan-m/
- Co-lead of Women in Research and Moms in Research community at Microsoft Research.
- Co-host of Women in Compilers and Tools Meetup Series, SF Bay Area, 2021 - 2024.
- PC member at PLDI 2025, LCTES 2025, CGO 2023, Euro LLVM 2023, ICCQ 2021, LLVM Developers' Meeting 2021, LLVM Dev Meeting 2020.
- Artifact Evaluation Chair of CGO 2022, CGO 2021.
- Student Research Competition (SRC) Chair at CGO 2026, PLDI 2023, PLDI 2022.

- SRC Judge at PLDI 2021.
- Panel member for Women in Compiler & Tools (WiCT), CGO 2020.
- Artifact Evaluation Reviewer, PLDI 2016.
- Student Volunteer, PLDI 2015-2016.
- Graduate Student Advisory Committee (GradSAC) Member, School of Computing, University of Utah, 2014-2016.