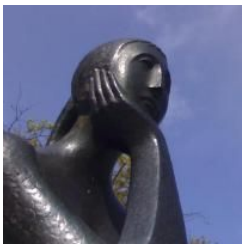


CUNY PONDER

The **Pr**O**g**ramming la**N**guages an**D** software **E**ngineering **R**esearch Lab

THE GRADUATE CENTER
CITY UNIVERSITY OF NEW YORK

HUNTER

CUNY PONDER, situated at Hunter College, works at the intersection of **software engineering** and **programming languages**. Our lab **develops**, **assess**, and **disseminates** techniques for automating critical software engineering tasks, including (static and/or dynamic) **program analysis**, **software security analysis**, and **software evolution and maintenance**. Our work typically results in algorithms and associated tools that analyze and/or transform (e.g., refactor) large and complex programs for improved **modularity**, **comprehension**, **maintainability**, **safety**, **security**, and **performance**. The techniques span multiple subfields of theory and application such as **programming languages**, **type theory**, (front-end) **compilers**, **data science**, **informational retrieval**, and **mathematical logic**.

Professor

Raffi Khatchadourian

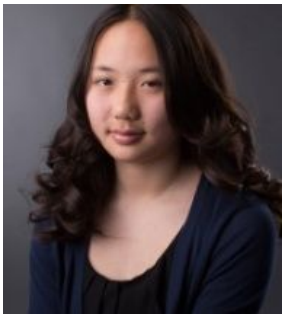
We are funded by Amazon Web Services (**AWS**), Women in Technology and Entrepreneurship in New York (**WiTNY**), the Japan Society for the Promotion of Science (**JSPS**), and the CUNY Diversity Projects Development Fund (**DPDF**). Our graduates work at Google, New York Times, TD Ameritrade, New York Foundling, J.P. Morgan, and AD/FIN. Our publications have appeared in **top software engineering conference** and have won **distinguished paper awards**.

Get in touch at ponder@hunter.cuny.edu and **find out more at** http://ponder-lab.github.io.

Students

Yiming Tang

Allan Spektor



Annie Wang

Krishna Desai

Oren Friedman

David Morant

Walter Rada

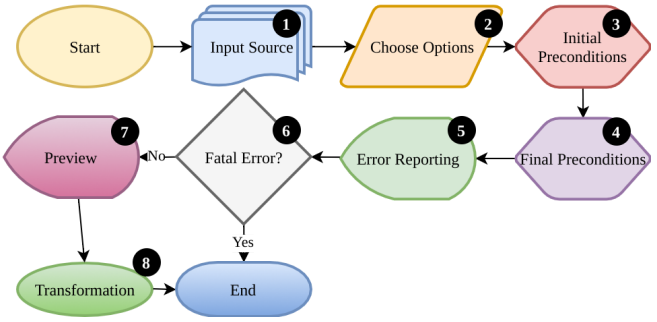
Olivia Moore

Md. Arefin

Projects

Safe Automated Refactoring for Intelligent Parallelization of Java 8 Streams

The Java 8 Stream API sets forth a promising new programming model that incorporates functional-like, MapReduce-style features into a mainstream programming language. However, using streams efficiently may involve subtle considerations. For example, although streams enable developers to run their code in parallel with little alteration, it is often not obvious if such code runs more efficiently this way. In fact, under certain conditions, running stream code in parallel can be less efficient than running it sequentially. Moreover, it can be unclear if running sequential stream code in parallel is safe and interference-free due to possible lambda expression side-effects. This project involves an automated refactoring approach that assists developers in writing optimal stream client code in a semantics-preserving fashion. The approach, based on a novel data ordering and typestate analysis, consists of refactorings that include preconditions and transformations for automatically determining when it is safe and possibly advantageous to convert a sequential stream to parallel and improve upon already parallel streams. The approach is implemented as a plug-in to the popular Eclipse IDE utilizing both WALA and SAFE.

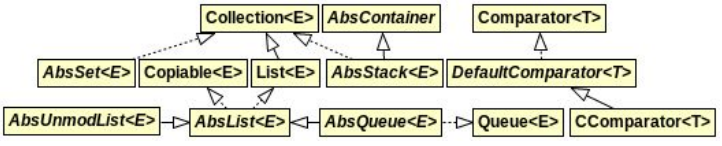


Automatic Migration of Legacy Java Method Implementations to Interfaces

The skeletal implementation pattern is a software design pattern consisting of defining an abstract class that provides a partial interface implementation. However, since Java allows only single class inheritance, if implementers decide to extend a skeletal implementation, they will not be allowed to extend any other class. Also, discovering the skeletal implementation may require a global analysis. Java 8 enhanced interfaces alleviate these problems by allowing interfaces to contain (default) method implementations, which implementers inherit. Java classes are then free to extend a different class, and a separate abstract class is no longer needed; developers considering implementing an interface need only examine the interface itself. Both of these benefits improve software modularity. This project involves developing a refactoring algorithm that assists developers in taking advantage of the enhanced interface feature for their legacy Java software.

Automated Refactoring of Legacy Java Software to Enumerated Types

Modern Java languages introduce several new features that offer significant improvements over older Java technology. In this project, we consider the new enum construct, which provides language support for enumerated types. Prior to recent Java languages, programmers needed to employ various patterns (e.g., the weak enum pattern) to compensate for the absence of enumerated types in Java. Unfortunately, these compensation patterns lack several highly-desirable properties of the enum construct, most notably, type safety. We present a novel fully-automated approach for transforming legacy Java code to use the new enumeration construct. This semantics-preserving approach increases type safety, produces code that is easier to comprehend, removes unnecessary complexity, and eliminates brittleness problems due to separate compilation. At the core of the proposed approach is an interprocedural type inference algorithm which tracks the flow of enumerated values. The algorithm was implemented as an open source, publicly available Eclipse plug-in and evaluated experimentally on 17 large Java benchmarks. Our results indicate that analysis cost is practical and the algorithm can successfully refactor a substantial number of fields to enumerated types.



Pointcut Change Prediction

Pointcut fragility is a well-documented problem in Aspect-Oriented Programming; changes to the base-code (the non-aspect part of a program) can lead to join points incorrectly falling in or out of the scope of pointcuts. Deciding which pointcuts have broken due to changes made to the base-code is a daunting task, especially in large and complex systems. This project consists of an automated approach that recommends pointcuts that are likely to require modification due to a particular base-code change, as well as ones that do not. Our hypothesis is that join points selected by a pointcut exhibit common structural characteristics. Patterns describing such commonality are used to recommend pointcuts that have potentially broken to the developer.