CS 842: *Advanced Topics in* Programming Language Design and Implementation

PROF STEPHEN M. WATT

Course Objectives

- Learn about programming language concepts that appear in both tried-and-true and new-and-hot languages.
- Develop a clear understanding of how programming languages are implemented and how key features work.
- To understand the issues and tradeoffs that are made in designing new programming languages, such as Rust and Typescript, or in extending popular languages such as Python and C++.

- You will be learning from the Prof and from each other in a seminar setting.

Professor Intro

Name: Stephen Watt

Background:

- IBM Research (Yorktown Heights) 12 years
- Prof at University of Nice (France), UWO, Waterloo (past Dean of Math)
- 2.5 compilers, several interpreters
- One of original authors of Maple
  - Interpreted language for symbolic computation
- Primary architect of Aldor
  - Compiled language with dependent types
- Research in
  - symbolic computation
  - programming languages and their implementation
  - machine learning for handwriting recognition.

Basic Course Facts

Prof: Stephen Watt

When: Thursdays 1pm to 4pm

Where: DC 2568

Text: None. Readings from the literature.

Exams: None

Assigned work:

- ***A study of a topic from the current literature.***
  Each student will make a 25 minute presentation of 1 (or more) articles.
- ***A software project.***
  Students will study or implement a language feature.

Evaluation:

- Presentation 40%
- Project 40%
- Participation 20%

Course Schedule

Thursdays 1pm to 4pm, DC 2528

- Sept 8: Prof intro. Basic course facts. Topics overview.
- Sept 15: Student intros *asynchronous – no class* *Intro video due Sept 16.*
- Sept 22: Lecture/Seminar *online*
- Sept 29: Lecture/Seminar *Article selection due.*
- Oct 6: Lecture/Seminar *Programming project proposal due.*
- Oct 13: *Break – No Class*
- Oct 20-Dec 1: Lectures/Seminars and Student Presentations
- Dec 6: *Programming projects due.*

Study of Topic from the Literature

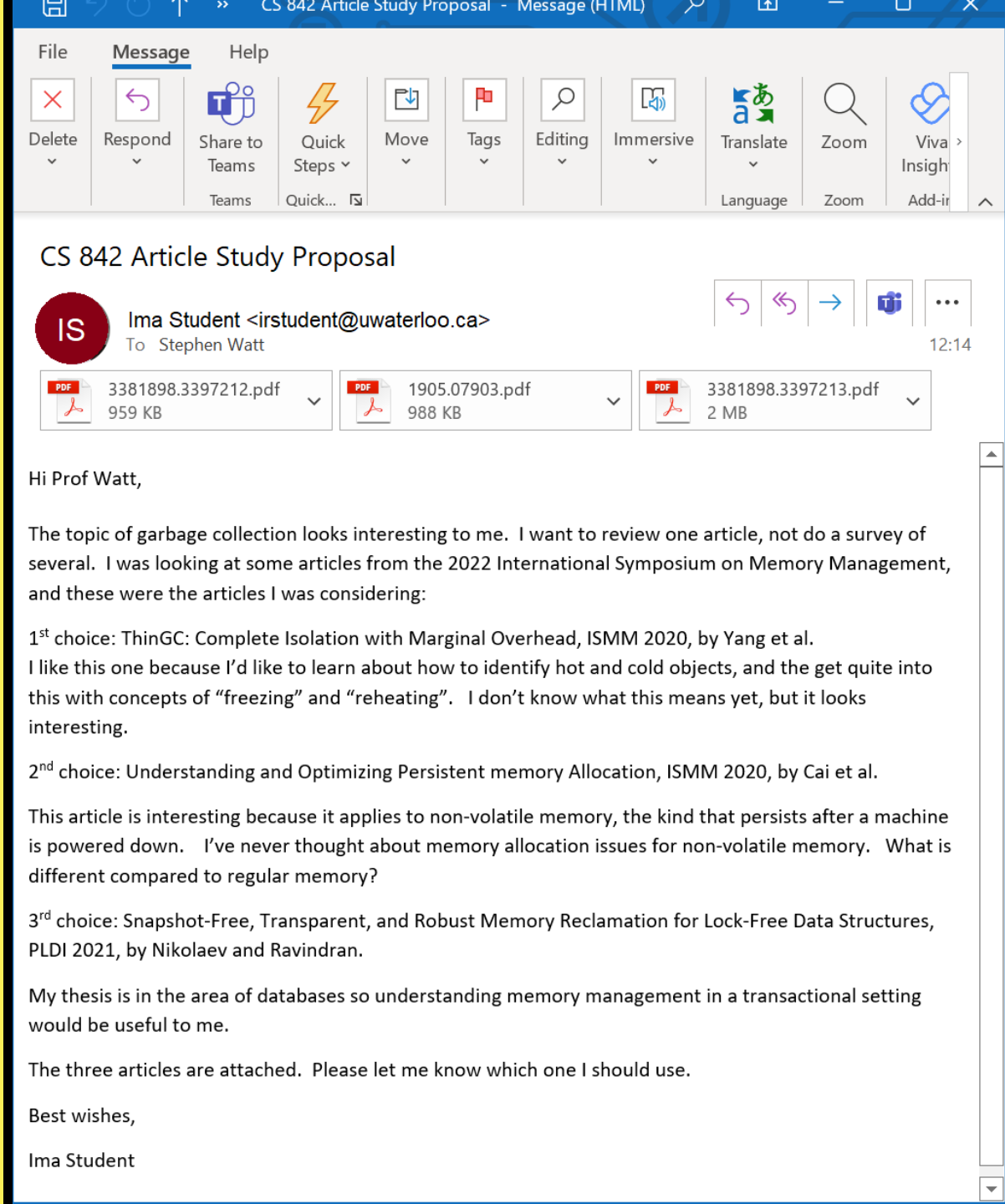
- Select article or topic from literature
  (see list of representative conferences and journals below).
- The third hour of the first lectures will be devoted to discussions
  to help with topics, relevant literature and selection.
- Get approval by E-mail:
  - Look at several articles from the sources given or similar, and pick **EITHER**
    - A ranked selection (1st choice, 2nd choice, 3rd choice) of articles that interest you **OR**
    - A topic on which there are several articles for which you would like to present an overview
  - Give bibliographic references (where appeared, date)
  - Attach a copy of the articles
  - Write a paragraph (100-200 words) about why you find the articles or topic interesting
  - Put as subject line: **CS 842 Article Study Proposal**
  - Send to Prof smwatt@uwaterloo.ca for approval by Sept 29.
  - In the case of the ranked selection, I will pick the best one for you.
- Prepare class presentation:
  25 minutes (15-25 slides). A 15 minute discussion will follow.

Representative Conferences and Journals

- *Programming Language Design and Implementation* (PLDI) https://pldi22.sigplan.org
- *Principles of Programming Languages* (POPL) https://popl22.sigplan.org
- *International Conference on Functional Programming* (ICFP) https://icfp22.sigplan.org
- *Systems, Programming, Languages and Applications* (SPLASH) https://2021.splashcon.org
- *International Symposium on Memory Management* (ISMM) https://conf.researchr.org/home/ismm-2021
- *Compiler Construction* (CC) https://conf.researchr.org/home/CC-2022
- *Principles and Practice of Parallel Programming* (PPoPP) https://ppopp22.sigplan.org
- *Architectural Support for Programming Languages and Operating Systems* (ASPLOS) https://asplos-conference.org
- Partial Evaluation and Program Manipulation (PEPM) https://popl22.sigplan.org/home/pepm-2022
- *European Conference on Object-Oriented Programming* (ECOOP) https://2022.ecoop.org/
- *Types in Language Design and Implementation* (TLDI) https://dl.acm.org/conference/tldi (older)

- *ACM Transactions on Programming Languages and Systems*
- *Journal of Functional Programming*

Example Article Study Proposal



Programming Project

- A significant project of your own devising, *e.g.:*
  - Implement a mini interpreter to illustrate a language feature.
  - Extend an open source language processor with a feature.
  - Do a comparative analysis and benchmarking of a specific feature in different languages.
  - Analyze a code base to determine which language features are really used.
  - Write a program transformation tool for a specific purpose.
  - If you don't have a specific project, the professor can make suggestions.
- An average student should budget about 1 day of work per week for up to 2 months.
- Get approval by E-mail:
  - Write a 1 or 2 page project proposal with
    - General overview and objectives
    - Approach you will take (roughly what platform, what tools, what test cases, …)
    - 8-12 milestones with a description of what each will involve.
  - Send to Prof smwatt@uwaterloo.ca for approval by October 6.
    - **Use subject line: CS 842 Programming Project Proposal**
- Submit project.
  - Due December 6.
  - Submit tar file with documentation, sources, building scripts, tests and demo.
  - Provide a 5 minute video with a demonstration (post to YouTube or other streaming service).
  - Send tar file and video link to Prof **using subject line: CS 842 Programming Project Submission**

Sample Programming Project Proposal

CS 842 Programming Project Proposal

Ima Student #20209999

Performance Analysis of Lazy Evaluation

Overview and Objective

Lazy evaluation is available in many functional programming languages, and can be simulated in others. This project will compare lazy evaluation implementations for performance.

Approach

I will develop a test suite of problems to benchmark performance in a set of lazy evaluation implementations. I am willing to learn the basics of a couple of programming languages to be able to do this.

The implementations I initially propose to compare are:

1. Scheme with "force" and "delay"
2. Haskell, native lazy evaluation
3. Miranda, native lazy evaluation
4. C++, simulated lazy evaluation
5. Golang, simulated lazy evaluation

The benchmark problems I initially propose to benchmark are:

1. Computation of Fibonacci numbers
2. Lazy power series computation of sin^2(x) + cos^2(x) to various orders
3. Two other problems to be determined.

I will benchmark both time spent and memory use.

The plaforms I will benchmark on are

1. 64-bit Ubuntu 22.04 Linux running on a laptop with processor xxxx and yyyy memory.
2. 64-bit Windows 10 desktop with processor ssss and tttt memory.

Milestones

1. Download and install the programming language processors needed.
2. Learn the basics and write simple test programs for the languages I do not yet know (Miranda, Haskell)
3. Finalize the choice of benchmarks.
4. Write the benchmarks for Scheme, C++ and Golang
5. Write a test harness to gather the benchmark data for Scheme, C++ and Golang.
6. Run the benchmarks for Scheme, C++ and Golang.
7. Learn enough of the required languages to write the benchmarks for Miranda and Haskell.
8. Write the benchmark programs for Miranda and Haskell.
9. Run the benchmarks for Miranda and Haskell.
10. Analyze the results.

A Brief Overview of Topics

Topics will be selected from:

- Memory management / garbage collection.
- Functional programming and closures.
- Lazy evaluation and parallel futures.
- Programming language issues around arithmetic types.
- Polymorphic language techniques.
- Types as first-class values, type categories, dependent types.
- Method dispatch and optimization in object-oriented languages.
- Topics in code optimization, including dataflow analysis.
- Iterators, generators, co-routines and their optimization.
- Potentially others, by request.

Memory Management / Garbage Collection

- Heaps
- Reference counting
- Garbage collection basics: root sets, heap
- Mark and sweep
- Copying collectors
- Generational collectors
- Distributed collectors
- Implementation techniques: forwarding pointers, write barriers, …

Functional Programming and Closures

- Basic functional programming ideas
- Lexical bindings
- Environments and escaping functions
- Relationship between OO methods and closures
- Spaghetti stacks
- Closures vs continuations

Lazy Evaluation and Parallel Futures

- Strict vs normal order evaluation
- Implementation of delayed expressions
- Strictness analysis in lazy languages
- An extended example: infinite lists
- Fixed-point methods
- Futures

PL Issues around Arithmetic Types

- Numeric hierarchies
- Hardware representation of numbers
- Immediate versus boxed numbers
- Misaligned pointers as integers
- NAN-boxing

Polymorphic Language Techniques

- Tagged values
- Objects
- Parametric polymorphism
  - Compile-time analysis
  - Type erasure and homogeneous implementation
  - Compile-time specialization
  - Run-time parametric polymorphism
- Dependent types

Types as First-Class Values

- What is a type?
- Compile-time vs run-time types.
- Types of types, type categories, C++ “concepts”
- First class dependent types vs “templates”
- Type producing functions vs “template templates”

Method Dispatch and Optimization

- Representation of objects
- Static, virtual and final methods
- Separate compilation, indirection and inlining

Topics in Code Optimization

- Binding times
- Dataflow analysis
- Constant propagation
- Common subexpression elimination
- Value numbering
- Code specialization
- How inlining affects the above

Iterators, Generators and Co-Routines

- Traversing data structures
- Loops as syntactic sugar for maps
- Iterator state
- Generators and streams
- Parallel iterators
- Iterators as co-routines
- Optimization

Upcoming Classes

- September 15:
  - **Asynchronous – offline. Independent work**. **No class.**
  - Scan through some of the representative conferences and journals by looking them up by name on the UWaterloo Library web site.
  - Prepare 5 minute video describing *your* background and interests/what *you* hope to get from the course. ***Intro video due Sept 16.***
  - Post to YouTube or other streaming service and send link by E-mail to instructor.
  - **Use subject line: CS 842 Self Intro**

- September 22:
  - **Synchronous – online.**
  - Professor will be away at a research conference, but will send link for an on-line session by *Teams*.