50.054 Compiler Design and Program Analysis

Course Objectives

This course aims to introduce a new programming paradigm to the learners, Functional programming and the suite of advanced language features and design patterns for software design and development. By building on top of these techniques, the course curates the process of designing a modern compiler, through syntax analysis, intermediate presentation construction, semantic analysis and code generation. More specifically the course focuses on the application of program analysis in areas of program optimization and software security analysis.

Prerequisites

50.005 Algorithms

Module Learning Outcomes

By the end of this module, students are able to

- Implement software solutions using functional programming language and applying design patterns
- Define the essential components in a program compilation pipeline
- Design a compiler for an imperative programming language
- Optimise the generated machine codes by applying program analysis techniques
- Detect bugs and security flaws in software by applying program analysis techniques

Measurable Outcomes

By the end of this module, students are able to

- Develop a parser for an imperative programming language with assignment, if-else and loop (AKA the source language) using Functional Programming
- Implement a type checker for the source language
- Develop a static analyser to eliminate dead codes
- Implement the register allocation algorithm in the target code generation module
- Develop a static analyser to identify potential security flaws in the source language

Topics

1. Functional Programming : Expression, Function, Conditional
2. Functional Programming : List, Algebraic data type and Pattern Matching
3. Functional Programming : Type class
4. Functional Programming : Generic and Functor
5. Functional Programming : Applicative and Monad
6. Syntax analysis: Lexing
7. Syntax analysis: Parsing (LL, LR, SLR)
8. Syntax analysis: Parser Combinator
9. Intermediate Representation: Pseudo-Assembly
10. Intermediate Representation: SSA
11. Semantic analysis: Dynamic Semantics
12. Semantic analysis: Type checking
13. Semantic analysis: Type Inference
14. Semantic analysis: Sign analysis
15. Semantic analysis: Liveness analysis
16. Code Gen: Instruction selection
17. Code Gen: Register allocation
18. Memory Management

Assessment

Mid-term 20%
Project 30%
Homework 15%
Final 30%
Class Participation 5%

Schedule

Week	Session 1	Session 2	Session 3	Assessment
1	Intro	FP: Expression, Function, Conditional, Recursion	Cohort Problem 1 Homework 1	Homework 1 no submission required
2	FP: List, Pattern Matching	FP: Algebraic Data type	Cohort Problem 2 Homework 2 Project Briefing	
3	FP: Generic, GADT	FP: Type Class	Cohort Problem 3	Homework 2 2%

		Functor	Homework 3	
4	FP: Applicative	FP: Monad	Cohort Problem 4 HW 3 (Cont'd)	Homework 3 4%
5	Syntax Analysis: Lexing, Parsing,	Top-down Parsing	Cohort Problem 5 Homework 4	
6	Bottom-up Parsing	IR: Pseudo-Assembly	Cohort Problem 6 Homework 4 (Cont'd)	Homework 4 4%
7				
8	Mid-Term	IR: SSA Dynamic Semantics	Project Check-off	Mid-Term 20% Project check-off 5%
9	Static Semantics Type Checking	Type Inference	Cohort Problem 7 Homework 5	
10	Semantic Analysis Lattice	Semantic Analysis Sign Analysis	Cohort Problem 8	Homework 5 5%
11	Code Gen Instruction Selection Register Allocation	Semantic Analysis Liveness analysis	Information Flow analysis	
12	Memory Management	Activation Record and function call	Project Presentation	Project 20%
13	Guest Lecture	Revision		Project Quiz 5%
14				Final 30%

Text Books

- Compilers: Principles, Techniques, and Tools is a computer science textbook by Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman
- Modern Compiler Implementation in ML by Andrew W. Appel
- Static Program Analysis by Anders Møller and Michael I. Schwartzbach