2017 Functional Programming

Academic unit or major	Graduate major in Mathematical and Computing Science
Instructor(s)	Wakita Ken Masuhara Hidehiko
Course component(s)	Lecture
Day/Period(Room No.)	Mon5-6(W831) Thr5-6(W831)
Group	-
Course number	MCS.T502
Credits	2
Academic year	2017
Offered quarter	3Q
Syllabus updated	2017/4/3
Language used	English

Syllabus

Course description and aims

The gift from decades of research and development activities of functional programming languages includes efficient garbage collection, type-directed optimization, and closure conversions. These techniques are starting to be incorporated in programming languages from non-functional paradigms such as C++ and Java, promoting more "functional style" in general programming.

When we see programs written in functional programming languages and shocked at its beauty and simplicity, we may wonder its execution efficiency. In this course, we jump into an implementation of a working compiler for a functional programming language and learn techniques to gradually converting highly abstract description of the functional-style program to lower level executable code, through series of conversions. These conversions are defined over well-defined interfaces: from upper-lever abstract interface down to lowest-level machine description are abstract syntax trees, K-normal forms, closure language, virtual machine.

Students will be exposed to one of the best example of systematically organized software project which deals with software complexity with formalism, layers of abstraction, and machine independence.

Student learning outcomes

Students will learn
1) a functional programming language,
2) methodologies of functional programming,
3) organization of a compiler for a functional programming language

During the course, we read a compiler of a tiny functional programming language, called MinCaml, which is written in a functional programming language called OCaml. From this experience, we can learn (1) an organization of middle-scale software project, (2) that highly abstract description of is gradually transformed down to lower-level representation passing through abstraction layers, (3) techniques to balance between description power and execution efficiency.

Keywords

Functional programming, compiler organization, OCaml

Competencies that will be developed

Intercultural skills	Communication skills	Specialist skills	Critical thinking skills	Practical and/or problem-solving skills
-	-	✔	-	-

Class flow

The course gives lectures for the first four weeks.

For the rest, students choose parts of the compiler components and explain the implementation. Each class starts with students' explanation, followed by the instructor's brief overview for next components that are covered in the coming class.

Course schedule/Required learning

	Course schedule	Required learning
Class 1	Overview	Guidance
Class 2	Introduction to functional programming in OCaml (1)	Primitive data types, compound data types, algebraic data types.
Class 3	Introduction to functional programming in OCaml (2)	Recursive data structures, recursive functions, higher-order functions, mutable states.
Class 4	Introduction to functional programming in OCaml (3)	Records, exception handling, modules, standard library, tools
Class 5	Software architecture of the MinCaml compiler	MinCaml is a tiny functional programming language and is implemented in a functional programming language OCaml.
Class 6	From program to abstract syntax tree	Lexical analysis and parsing.
Class 7	Type analysis	Type analysis, type inference, unification
Class 8	From abstract syntax tree to K-normal form	K-normal form, alpha-conversion
Class 9	Optimization(1)	Beta-reduction, reduction of nested let's, inline code expansion

	Course schedule	Required learning
Class 10	Optimization (2)	Constant folding, elimination of redundant definitions
Class 11	Elimination of functional closures	Closure conversion
Class 12	Generation of abstract machine code	Abstract machine code generation
Class 13	Register assignment	Register assignment
Class 14	Generation of executable code	Generation of assembly code, runtime system
Class 15	Wrap up	Wrap up

Textbook(s)

Unfixed

Reference books, course materials, etc.

Courseware will be provided on GitHub. GitHub repository information is found on OCW-i.
http://esumii.github.io/min-caml/index7.html
http://esumii.github.io/min-caml/paper.pdf

Assessment criteria and methods

Students will be assessed on their understanding of functional programming and organization of a compiler for a tiny functional programming language. There is no term-end examination.

Related courses

MCS.T213 ： Introduction to Algorithms and Data Structures
MCS.T224 ： Programming I
MCS.T303 ： Programming II
MCS.T334 ： Compiler Construction
CSC.T372 ： Compiler Construction

Prerequisites (i.e., required knowledge, skills, courses, etc.)

Basic understanding of algorithms and data structures, and fluency with at least one programming language are required.