CSE 6341 (Approved): Foundations of Programming Languages

Course Description

Conceptual foundations of programming languages: attribute grammars; types; functional languages; language semantics; abstract interpretation.

Prior Course Number: CSE 755
Transcript Abbreviation: Fndns Prog Langs
Grading Plan: Letter Grade
Course Deliveries: Classroom
Course Levels: Graduate
Student Ranks: Senior, Masters, Doctoral
Course Offerings: Autumn, Spring
Flex Scheduled Course: Never
Course Frequency: Every Year
Course Length: 14 Week
Credits: 3.0
Repeatable: No
Time Distribution: 3.0 hr Lec
Expected out-of-class hours per week: 6.0
Graded Component: Lecture
Credit by Examination: No
Admission Condition: No
Off Campus: Never
Campus Locations: Columbus
Prerequisites and Co-requisites: CSE 3341 or CSE 5341 or CSE 655
Exclusions: Not open to students with credit for CSE 755
Cross-Listings:

The course is required for this unit's degrees, majors, and/or minors: No
The course is a GEC: No
The course is an elective (for this or other units) or is a service course for other units: Yes

Subject/CIP Code: 14.0901
Subsidy Level: Doctoral Course

Programs

Abbreviation	Description
BS CSE	BS Computer Science and Engineering
MS CSE	MS Computer Science and Engineering
PhD CSE	PhD Computer Science and Engineering

Course Goals

Master using attribute grammars to specify context-sensitive conditions related to standard imperative language constructs.
Master using attribute grammars to specify code generation for standard imperative language constructs.
Be competent with principles of functional languages and techniques for their implementations.
Be competent with analyzing types in functional, imperative, and object-oriented languages.
Be competent with analyzing the formal semantics of imperative languages.
Be familiar with abstract interpretation and its application to derive approximate semantics of programs.

Course Topics

Topic	Lec	Rec	Lab	Cli	IS	Sem	FE	Wor
Attribute grammars; using AGs to define context-sensitive syntax of languages;	6.0							
Using AGs to define translational semantics of imperative languages;	3.0							
Functional languages: typed and untyped (Lambda calculus, Lisp, ...); implementations of functional languages including memory management; meta-circular interpreters;	12.0							
Types in programming languages: Type systems, type checking, type inference/ reconstruction, recursive types, subtypes, higher-order types; dynamically typed languages; implementation issues;	9.0							
Formal semantics of programming languages: structured operational semantics; denotational semantics; abstract interpretation and static analysis;	9.0							
Exams, review;	3.0							

Representative Assignments

Homework assignments involving designing suitable AGs to express typical context-sensitive conditions and generation of code for standard imperative constructs;
Implement a complete interpreter for pure Lisp (plus function definitions);
Homework assignments involving analyzing type systems and relations in standard languages;
Implement a basic type-inferencing algorithm;
Homework assignments involving structural operational semantics and static analysis of variations of standard imperative constructs;

Grades

Aspect	Percent
Homeworks	20%
Programming projects	35%
Midterms, Finals	45%

Representative Textbooks and Other Course Materials

Title	Author
Types and programming languages	B. Pierce

ABET-EAC Criterion 3 Outcomes

Course Contribution		College Outcome
***	a	An ability to apply knowledge of mathematics, science, and engineering.
**	b	An ability to design and conduct experiments, as well as to analyze and interpret data.
***	c	An ability to design a system, component, or process to meet desired needs.
*	d	An ability to function on multi-disciplinary teams.

Course Contribution		College Outcome
***	e	An ability to identify, formulate, and solve engineering problems.
*	f	An understanding of professional and ethical responsibility.
*	g	An ability to communicate effectively.
*	h	The broad education necessary to understand the impact of engineering solutions in a global and societal context.
**	i	A recognition of the need for, and an ability to engage in life-long learning.
*	j	A knowledge of contemporary issues.
***	k	An ability to use the techniques, skills, and modern engineering tools necessary for engineering practice.

BS CSE Program Outcomes

Course Contribution		Program Outcome
***	a	an ability to apply knowledge of computing, mathematics including discrete mathematics as well as probability and statistics, science, and engineering;
**	b	an ability to design and conduct experiments, as well as to analyze and interpret data;
***	c	an ability to design, implement, and evaluate a software or a software/hardware system, component, or process to meet desired needs within realistic constraints such as memory, runtime efficiency, as well as appropriate constraints related to economic, environmental, social, political, ethical, health and safety, manufacturability, and sustainability considerations;
*	d	an ability to function on multi-disciplinary teams;
***	e	an ability to identify, formulate, and solve engineering problems;
*	f	an understanding of professional, ethical, legal, security and social issues and responsibilities;
*	g	an ability to communicate effectively with a range of audiences;
*	h	an ability to analyze the local and global impact of computing on individuals, organizations, and society;
**	i	a recognition of the need for, and an ability to engage in life-long learning and continuing professional development;
*	j	a knowledge of contemporary issues;
***	k	an ability to use the techniques, skills, and modern engineering tools necessary for practice as a CSE professional;
**	l	an ability to analyze a problem, and identify and define the computing requirements appropriate to its solution;
***	m	an ability to apply mathematical foundations, algorithmic principles, and computer science theory in the modeling and design of computer-based systems in a way that demonstrates comprehension of the tradeoffs involved in design choices;
**	n	an ability to apply design and development principles in the construction of software systems of varying complexity.

Prepared by: Neelam Soundarajan