COMPILER DESIGN

Review

Examination

- Objective: to verify that the students grasp the theoretical aspects of compiler design, as taught in class.
- Duration: 180 minutes.
- Open-book examination: all course notes, any textbook, or paper documents allowed, no electronic device permitted.

Course review

- Compiler architecture
  - Phases:
    - lexical analysis
    - syntactic analysis
    - semantic analysis
    - code optimization
    - code generation
  - Front-end, back-end
  - Intermediate representations
  - Mechanisms:
    - parsing tables
    - symbol table
    - semantic actions/semantic records
    - attribute migration
  - Functioning/role of each phase/component/mechanisms
  - Optionality of some phases

Course review

- Lexical analysis
  - Roles
    - White space removal
    - Processing comments
    - Check and recover from lexical errors
    - Creation of a stream of tokens
  - Design
    - Translation of regular expression into a DFA
    - Thompson construction
    - Rabin–Scott powerset construction
  - Implementation
    - Case statement or state transition table/algorithm
  - Notable examination questions
    - Generate a DFA from regular expressions
    - Generate a DFA from NDFA

Course review

- Syntactic analysis
  - Roles
    - Analyze the program's structure
    - Check, report and recover from syntax errors leading to useful and comprehensive compiler output
  - Design
    - Generative context-free grammars, generating a derivation proving the validity of the input program according to the grammar
    - First and follow sets
    - Grammar transformation (removal of left recursions, ambiguities)
    - All designs are based on a stack mechanism
    - Top-down: predictive parsing, recursive descent, table-driven (require removal of left recursions, ambiguities)
    - Bottom-up: SLR, CLR, LALR (item generation)
    - Error recovery using "synchronizing tokens"
    - AST generation as intermediate representation
    - Attribute migration, semantic stack

Course review

- Syntactic analysis (cont.)
  - Implementation
    - Recursive descent top-down: each production is implemented as a function matching terminals and calling other such functions to parse non-terminals according to other rules
    - Table-driven top-down: table is constructed using the first and follow sets, based on the notion of "generative grammar"
    - Bottom-up: SLR, CLR, LALR: creation of a DFA using items and first and follow sets, creation of the state transition table with "action" and "goto" parts, called a "shift/reduce" parser.
  - Notable examination questions
    - Given a grammar, generate a table for a table-driven top-down predictive parser
    - Given a grammar, write some functions for a recursive-descent predictive parser
    - Given a grammar, eliminate left recursions and ambiguities
    - Given a grammar and a valid sentence, provide a derivation proving that this sentence is derivable from the grammar
    - Given a grammar, generate the sets of (CLR, SLR, LALR) items, then generate the corresponding state transition table and/or state transition diagram
    - Given a state transition table and a token stream, execute a parse trace for any of the above bottom-up parsing techniques

Course review

- Semantic Analysis/translation
  - Roles
    - Verify the semantic validity of the program
    - Translate the program into executable code
  - Design
    - Symbol table
    - Intermediate representations (optional)
    - Optimization, high level and/or low level (optional)
    - Semantic actions and semantic records
    - AST traversal and the Visitor design pattern

Course review

- Semantic Analysis/translation (cont.)
    - Implementation
        - Symbol table: nested tables to manage scoping
        - Intermediate representation: trees, directed acyclic graphs, tree traversal/analysis algorithms for further processing
        - Intermediate code: postfix notation, three-address code, quadruples, pcode, byte code
    - Notable examination questions
        - Given a program, sketch its corresponding symbol table structure.
        - Given a simple statement/expression write the corresponding Moon code translation.