IT8003 FORMAL LANGUAGES AND AUTOMATA THEORY

DETAILED SYLLABUS

OBJECTIVES:

- To understand a finite automata for a given language.
- To understand the relation between grammar and language
- To understand the basic principles of working of a compiler
- To study about the type checking procedure during the compilation
- To understand the storage structure of the running program

UNIT I AUTOMATA

Introduction to formal proof – Additional forms of proof – Inductive proofs –Finite Automata (FA) – Deterministic Finite Automata (DFA) – Non-deterministic Finite Automata (NFA) – Finite Automata with Epsilon transitions- Equivalence and minimization of Automata.

UNIT II CONTEXT FREE GRAMMARS AND LANGUAGES

Context-Free Grammar (CFG) – Parse Trees – Ambiguity in grammars and languages – Definition of the Pushdown automata – Languages of a Pushdown Automata – Equivalence of Pushdown automata and CFG– Deterministic Pushdown Automata- Normal forms for CFG – Pumping Lemma for CFL – Closure Properties of CFL – Turing Machines – Programming Techniques for TM.

UNIT III BASICS OF COMPILATION

Compilers – Analysis of source program – Phases of a compiler – Grouping of phases – Compiler construction tools – Lexical Analyzer: Token Specification – Token Recognition – A language for Specifying lexical analyzer – Top down parser: Table implementation of Predictive Parser – Bottom up Parser: SLR (1) Parser – Parser generators.

UNIT IV TYPE CHECKING AND RUNTIME ENVIRONMENTS

Syntax directed definitions – Construction of syntax trees – Type systems – Specification of a simple type checker- Equivalence of type expressions – Type conversions – Attribute grammar for a simple type checking system – Runtime Environments: Source language issues – Storage organization – Storage allocation strategies – Parameter passing.

UNIT V CODE GENERATION AND OPTIMIZATION

Issues in the design of a code generator - The target machine - Run-time storage management - Basic blocks and flow graphs - Next-use information - A simple code generator - Register allocation and assignment - The dag representation of basic blocks - Generating code from DAG – Dynamic programming code generation algorithm – Code generator generators - Code optimization.

TEXT BOOKS:

1. J.E. Hopcroft, R. Motwani and J.D. Ullman, —Introduction to Automata Theory, Languages and Computations‖, Second Edition, Pearson Education, 2007.

2. Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman, —Compilers: Principles, Techniques and Tools‖, Second Edition, Pearson Education,2008.

REFERENCES:

1. J. Martin, —Introduction to Languages and the Theory of computation‖ Third Edition, Tata Mc Graw Hill, 2007

2. Randy Allen, Ken Kennedy, —Optimizing Compilers for Modern Architectures: A Dependence-based Approach‖, Morgan Kaufmann Publishers, 2002.

3. Steven S. Muchnick, —Advanced Compiler Design and Implementation‖, Morgan Kaufmann Publishers - Elsevier Science, India, Indian Reprint 2003.

4. Muneeswaran. K, —Compiler Design‖, Oxford University Press, 2012