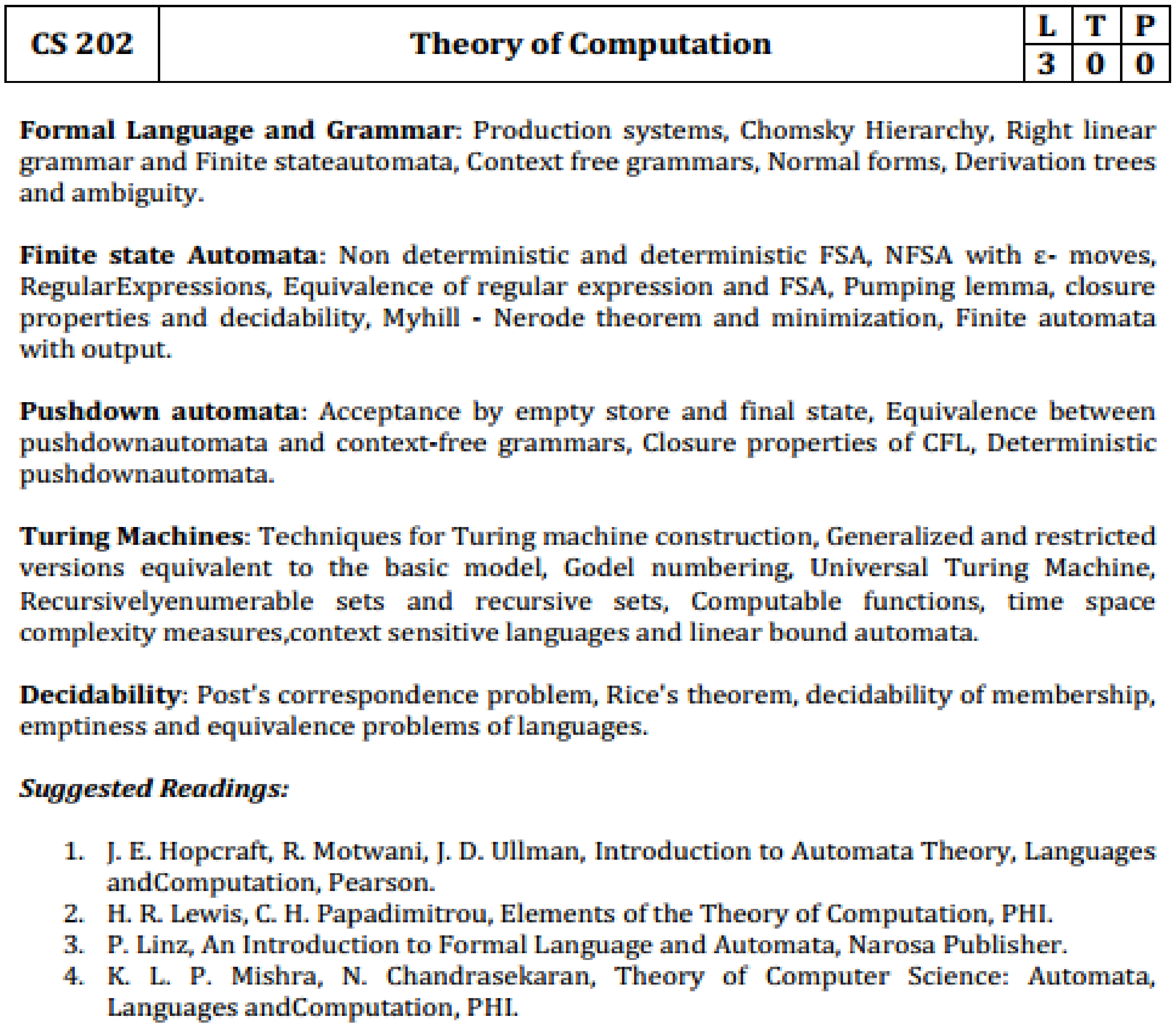
Introduction
to
Theory of Computation: CS-202

Syllabus



CS 202	Theory of Computation	L	T	P
		3	0	0

Formal Language and Grammar: Production systems, Chomsky Hierarchy, Right linear grammar and Finite stateautomata, Context free grammars, Normal forms, Derivation trees and ambiguity.

Finite state Automata: Non deterministic and deterministic FSA, NFSA with ε- moves, RegularExpressions, Equivalence of regular expression and FSA, Pumping lemma, closure properties and decidability, Myhill - Nerode theorem and minimization, Finite automata with output.

Pushdown automata: Acceptance by empty store and final state, Equivalence between pushdownautomata and context-free grammars, Closure properties of CFL, Deterministic pushdownautomata.

Turing Machines: Techniques for Turing machine construction, Generalized and restricted versions equivalent to the basic model, Godel numbering, Universal Turing Machine, Recursivelyenumerable sets and recursive sets, Computable functions, time space complexity measures,context sensitive languages and linear bound automata.

Decidability: Post's correspondence problem, Rice's theorem, decidability of membership, emptiness and equivalence problems of languages.

Suggested Readings:

1. J. E. Hopcraft, R. Motwani, J. D. Ullman, Introduction to Automata Theory, Languages andComputation, Pearson.
2. H. R. Lewis, C. H. Papadimitrou, Elements of the Theory of Computation, PHI.
3. P. Linz, An Introduction to Formal Language and Automata, Narosa Publisher.
4. K. L. P. Mishra, N. Chandrasekaran, Theory of Computer Science: Automata, Languages andComputation, PHI.

What is Theory of Computation (TOC)?

- TOC is the study of mathematical machines called automata.

- What are the fundamental capabilities and limitations of computers?
- To answer this, we will study abstract mathematical models of computers
- These mathematical models abstract away many of the details of computers to allow us to focus on the essential aspects of computation
- It allows us to develop a mathematical theory of computation

Why study the theory of computing?

- Core mathematics of CS (has not changed in over 30 years)
- Many applications, especially in design of compilers and programming languages
- Important to be able to recognize uncomputable and intractable problems
- Need to know this in order to be a computer scientist, not simply a computer programmer

- Formal language
  - a subset of the set of all possible strings from a set of symbols
  - example: the set of all syntactically correct C programs
- Automata
  - abstract, mathematical model of computer
  - examples: finite automata, pushdown automata Turing machine

Formal language

Alphabet = finite set of symbols or characters
examples: $\Sigma = \{a,b\}$, binary, ASCII

String = finite sequence of symbols from an alphabet
examples: aab, bbaba, also computer programs

A **formal language** is a set of strings over an alphabet
Examples of formal languages over alphabet $\Sigma = \{a, b\}$:

$L_1 = \{aa, aba, aababa, aa\}$
$L_2 = \{$all strings containing just two a's and any number of b's$\}$

A formal language can be finite or infinite.

Formal languages (continued)

We often use **string variables** ; $u = aab$, $v = bbaba$

Operations on strings

length: $|u| = 3$

reversal: $u^R = baa$

concatenation: $uv = aabbbaba$

The **empty string** , denoted λ , has some special properties:

$$| \lambda | = 0$$

$$\lambda w = w \lambda = w$$

Formal languages (continued)

If w is a string, then w^n stands for the string obtained by repeating w n times.

$w^0 = \lambda$

$\Sigma^+ = \Sigma^* - \{\lambda\}$

$L^0 = \{\lambda\}$

$L^1 = L$

Important example of a formal language

- alphabet: ASCII symbols
- string: a particular C++ program
- formal language: set of all legal C++ programs

Grammars

A grammar G is defined as a quadruple:

$G = (V, T, S, P)$

Where V is a finite set of objects called variables

T is a finite set of objects called terminal symbols

$S \in V$ is a special symbol called the Start symbol

P is a finite set of productions or "production rules"

Sets V and T are nonempty and disjoint

Grammars

What is the relationship between a language and a grammar?

Let G = (*V*, *T*, *S*, *P*)

The set

$$L(G) = \{w \in T^* : S \overset{*}{\Rightarrow} w\}$$

is the language generated by G.

Grammars

Consider the grammar G = (*V*, *T*, *S*, *P*), where:

V =	{S}
T =	{a, b}
S =	S,
P =	$S \rightarrow aSb$
	$S \rightarrow \lambda$

Grammars

What are some of the strings in this language?

$S \Rightarrow aSb \Rightarrow ab$

$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$

$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb$

It is easy to see that the language generated by this grammar is:

$L(G) = \{a^n b^n : n \geq 0\}$

(See proof on pp. 21-22 in Linz)

Language-recognition problem

- There are many types of computational problem. We will focus on the simplest, called the "language-recognition problem."
- Given a string, determine whether it belongs to a language or not. (Practical application for compilers: Is this a valid C++ program?)
- We study simple models of computation called "automata," and measure their computational power in terms of the class of languages they can recognize.

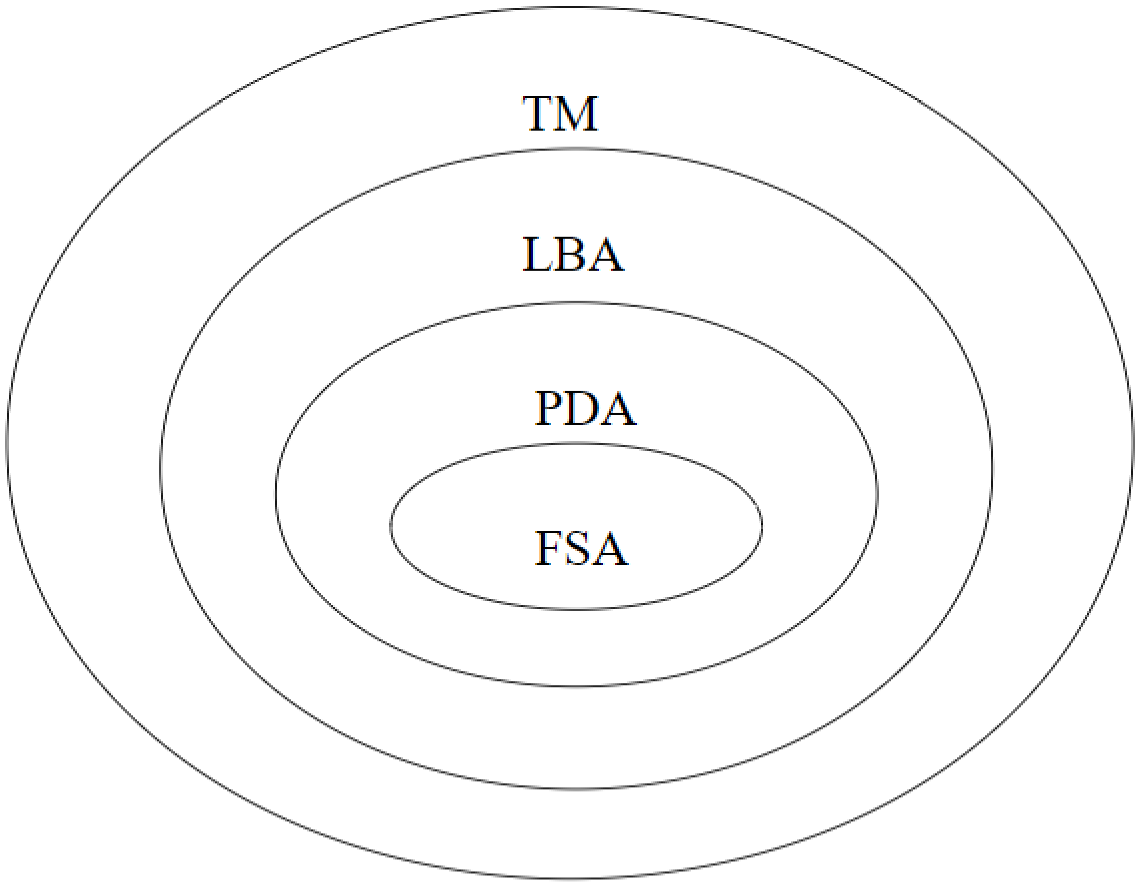
Finite automata

- Developed in 1940's and 1950's for neural net models of brain and computer hardware design
- *Finite memory!*
- Many applications:
  - text-editing software: search and replace
  - many forms of pattern-recognition (including use in WWW search engines)
  - compilers: recognizing keywords (lexical analysis)
  - sequential circuit design
  - software specification and design
  - communications protocols

Pushdown automata

- Noam Chomsky’s work in 1950’s and 1960’s on grammars for natural languages
- infinite memory, organized as a stack
- Applications:
  - compilers: parsing computer programs
  - programming language design

Computational power



Automata, languages, and grammars

- In this course, we will study the relationship between automata, languages, and grammars
- Recall that a formal language is a set of strings over a finite alphabet
- Automata are used to *recognize* languages
- Grammars are used to *generate* languages
- (All these concepts fit together)

Classification of automata, languages, and grammars

Automata	Language	Grammar
Turing machine	Recursively enumerable	Recursively enumerable
Linear-bounded automaton	Context sensitive	Context sensitive
Nondeterministic push-down automaton	Context free	Context free
Finite-state automaton	regular	regular

Besides developing a theory of classes of languages and automata, we will study the limits of computation. We will consider the following two important questions:

- What problems are impossible for a computer to solve?
- What problems are too difficult for a computer to solve in practice (although possible to solve in principle)?

Uncomputable (undecidable) problems

- Many well-defined (and apparently simple) problems cannot be solved by any computer
- Examples:
  - For any program x, does x have an infinite loop?
  - For any two programs x and y, do these two programs have the same input/output behavior?
  - For any program x, does x meet its specification? (i.e., does it have any bugs?)

Thank you