Automata - Basic Review

James Oravec

November 16, 2006

Acknowledgment: This review is a short recap of lectures by Dr. Larmore and key topics from Introduction to the Theory of Computation by Michael Sipser.

The following are keywords from chapter 0, if you are unfamiliar with any of them, you should review that chapter.
Alphabet, Argument, binary relation, boolean operation, boolean value, Cartesian product, complement, concatenation, conjunction, connected graph, cycle, directed graph, disjunction, domain, edge, element, empty set, empty string, equivalence relation, function, graph, intersection, k-tuple, language, member, node, pair, path, predicate, property, range, relation, sequence, set, simple path, string, symbol, tree, union, and vertex.

Start Chapter 1
Finite Machine
Finite Automaton
State diagram
A finite automaton has states, an alphabet, transition function, start state, and set of accept states.
A language is a **regular language** if some finite automaton recognizes it.
Let A and B be languages.
Union: $A \cup B = \{x | x \in A \text{ or } x \in B\}$
Concatenation $A \circ B = \{xy | x \in A \text{ and } y \in B\}$
Star A* $= \{x_1 x_2 ... x_k | k \geq 0 \text{ and each } x_i \in A\}$

In a **nondeterministic machine** several choices may exist for the next state at any point.
A nondeterministic finite automaton is similar to a deterministic finite automaton. They differ essentially in their type of transition function.
Two machine are **equivalent** if they recognize the same language.
Every nondeterministic finite automaton has an equivalent deterministic finite automaton.
A language is regular if and only if some nondeterministic finite automaton recognizes it.
Union, concatenation, and star of regular languages are still regular.
Regular expression (Ex. 0*10*)
A language is regular if and only if some regular expression describes it.
You use the pumping lemma to prove a language is regular.
End Chapter 1

Start Chapter 2
Context-free grammars (CFG) are a more powerful method of describing languages.
CFGs have substitution rules called productions. The also contain a start variable, variables, and terminals. One substitution is referred to as **yields**, while the sequence of substitutions to obtain a string is called a **derivation**.

Language of the grammar
Context-free grammar - Any language that can be generated by some context-free grammar.
Ambiguity - If a grammar can generate the same string in several different ways, then it is said to be ambiguous.
A string w is derived ambiguously in context-free grammar G if it has two or more different leftmost derivations. Grammar G is ambiguous if it generates some string ambiguously.

A context-free grammar is in Chomsky normal form if every rule is of the form:
$A \to BC$
$A \to a$
where a is any terminal and A, B, C are any variables - except that B and C may not be the start variable. In addition we permit the rule $S \to \epsilon$, where S is the start variable.

Any context-free language is generated by a context-free grammar in Chomsky normal form.
Computational model: **pushdown automata**.
A pushdown automata has an extra component called a stack.
Pushdown automata may be nondeterministic. Deterministic and nondeterministic pushdown automata are *not* equivalent in power.

An example that a pushdown automata recognizes is:
$\{a^i b^j c^k | k \geq 0 \text{ and } i = j \text{ or } i = k\}$
nondeterminism is essential for recognizing the language with a PDA.

Context-free grammars and pushdown automata are equivalent in power.
A language is context free if and only if some pushdown automaton recognizes it.
If a language is context free, then some pushdown automaton recognizes it.
If a pushdown automaton recognizes some language, then it is context free.
Every regular language is context free.
There is a pumping lemma for context-free languages.
End Chapter 2

Start Chapter 3
A Turing Machine can do everything a real computer can do.
A Turing Machine cannot solve certain problems.
The Turing machine model uses an infinite tape.
A Turing machine has several configurations: Start configuration, accepting configuration, rejecting configuration and halting configuration.
Call a language Turing-recognizable (or a recursively enumerable language) if some Turing machine can recognize it.Call a language Turing-decidable (or a recursive language) or simply decidable if some Turing machine decides it.
Variants of the Turing machine model all have the same power.
Every nondeterministic Turing machine has an equivalent deterministic Turing machine. (Do each choice of the nondeterministic Turing machine in breadth first search mannor).
A language is decidable if and only if some nondeterministic Turing machine decides it.
A language is Turing-recognizable if and only if some enumerator enumerates it.
The Turing machine merely serves as a precise model for the definition of algorithm.
We need only to be comfortable enough with Turing machines to believe that they capture all algorithms, this allows us to continue the study of the theory of computation.

The input to a Turing machine is always a string.
End Chapter 3

Start Chapter 4
In this chapter we begin to investigate the power of algorithms to solve problems. See page 170 of Sipser.
$A_{DFA} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}$
A_{DFA} is a decidable language.
$A_{NFA} = \{\langle B, wB\rangle \mid B \text{ is an NFA that accepts input string } w\}$
A_{NFA} is a decidable language.
$A_{REX} = \{\langle R, w\rangle \mid R \text{ is a regular expression that generates string } w\}$
A_{REX} is a decidable language.
$E_{DFA} = \{\langle A, w\rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$
E_{DFA} is a decidable language.
$EQ_{DFA} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$
EQ_{DFA} is a decidable language.
$A_{CFG} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}$
A_{CFG} is a decidable language.
$E_{CFG} = \{\langle G\rangle \mid G \text{ is a CFG and } L(G) = O\}$
E_{CFG} is a decidable language.
$EQ_{CFG} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}$
EQ_{CFG} is **not** decidable.

Every context-free language is decidable.

The halting problem
$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$
A_{TM} is undecidable. (A_{TM} is sometimes called the **halting problem**)

A set A is **countable** if either it is finite or it has the same size as N.
Q is countable.
R is uncountable.

Some languages are not Turing-recognizable.
The Halting Problem is undecidable.

We say that a language is **co-Turing-recognizable** if it is the complement of a Turing-recognizable language.
A language is decidable if and only if it is Turing-recognizable and co-Turing-recognizable.
The complement of A_{TM} is not Turing-recognizable.
End Chapter 4

Other Stuff:
Natural Languages - English, Chinese, *etc.*We do not discuss natural languages in this course.
Formal Languages - Symbols, alphabets, strings, and operations such as concatenation.
Configuration of M: a complete description of the current situation. Each configuration is finitely describable.
Classes of Machines: DFA, NFA, PDA, TM, NTM (non deterministic Turing Machine)
If NFA has n states its equivalent minimal DFA has no more than 2^n states.

The *star* operation can also be referred to as Kleene Closure.

3 definitions of a Regular Language:
1. Any language accepted by any DFA
2. Any language accepted by any NFA
3. Any language generated by any regular expression.

Example of a non-regular language: $L = \{a^n b^n\}$ (Formal proof done by the Pumping Lemma)
Every regular language has a unique minimal DFA.

Pumping Lemma for Regular Languages
For any regular language L, there exists a number n, such that, for any string $w \in L$, if $|w| \geq n$ then there exists strings x, y, z such that

1. $w = xyz$
2. $|xy| \leq n$
3. $|y| \geq 1$
4. For any $i \geq 0$, $xy^i z \in L$

Use the pumping lemma to prove a language is not regular.
If L_1 and L_2 are regular then
$L_1 L_2$ (Concatenation), $L_1 + L_2$ (Union), $L_1 \cap L_2$ (intersection), $L_1 *$ (Kleene Closure or Star), $L_1' = \Sigma * - L_1$ (complement) are all regular.

Homomorphism - is when each symbol in one alphabet is replaced by a string over another alphabet. (They could be the same alphabet.) A Caesar Cypher is an example.
T or F - If h is a homomorphism and $h(L_1) = L_2$ is regular then L_1 is regular (Ans: False)
Def: A language L is **context-free** if L is generated by some **context-free grammar (CFG)**
Examples of Algebraic Languages (which are all generaged by CFGs): High School Algebra, Regular Expressions, Boolean Expressions, etc.
Programming Languages - Are not context free, even though they have a CFG. The CFG does not generate the language, so it is not CF.
An application of automata theory is compilers. Two examples of common parsers in compilers are: top down parser (uses left most derivation) and bottom up parser (Reverse Right Most Derivation).
G_1 is equivalent to G_2 if both generate the same language.
Def: a CFG G is ambiguous if there is some string $w \in L(G)$ which has more than one parse tree in G.
Question: Given an ambiguous CFG G does there exist an unambiguous CFG equivalent to G? (Ans: Maybe not; there are some CF languages that are inherently ambiguous)
Question: Given two CFGs are they equivalent? (Ans: Undecidable)
A PDA has one stack. A PDA has limited computational power (Turing machines can do more). A PDA with 2 stacks can do more than a PDA with one stack. A PDA with 2 stacks has *Turing power*, *i.e.*, it can do anything any machine can do.
A PDA can accept $L = a^n b^n | n \geq 0$
A PDA is nondeterministic (a deterministic PDA is called a DPDA).
Not every CFL is accepted by a DPDA.
T or F. Every CFL with an ambiguous grammar is accepted by some DPDA. (Ans: False)

Suppose f is a function. If M_f exists, we say f is **computable**. Def: A Language L is **decidable** if the membership problem for L is decidable. More precisely stated, if Σ is an alphabet and L is a language over Σ, define the *characteristic function* of L to be the function from $\Sigma^\star$ to $\{0,1\}$ which is 1 for any string in L and 0 for any other string. We way that L is **decidable**, or *recursive*, if its characteristic function is a recursive function.
Every regular language L is decidable.
If L is a CFL, then the membership problem for L is decidable.

The CYK algorithm
The CYK algorithm is used to determine if a string can be generated by from a CFL.
The CYK algorithm is O(n^3).

LALR parser is a shift-reduce parser.

If any machine can do anything, some Turing Machine can do it.
Not everything can be done by machines.

A TM could:
a) Accept a language
b) Decide a language
c) Compute a function
d) Enumerate a language

Some functions can be computed by a TM, some cannot. Recursive Functions can be computed. An example of one that cannot is those in the **diagonal language**.
Is $w \in L$? This is called the **membership problem**.
Decision Problem also referred to as the $0-1$ **Problem**.

Examples of Undecidable Problems:
Halting Problem
The Knot Problem

If f is a recursive reduction of L_1 to L_2, and if L_2 is decidable, then L_1 is decidable.
Contrapositively: If f is a recursive reduction of L_1 to L_2, and if L_1 is undecidable, then L_2 is undecidable.

No TM accepts the L_{diag}.
L_{halt} is undecidable, but it is Turing acceptable.

A language L is *recursively enumerable* if and only if L is Turing acceptable if and only if L is generated by some unrestricted grammar.

Every NP language has P-TIME certificates.
There are many known NP-complete problems. If any one NP complete problem is P-TIME, then P-TIME = NP-TIME
$L_{prime} \in P-TIME$
$L_{composite} \in P-TIME$

We abbreviate the class P-TIME by P and the class NP-TIME by NP, provided there is no confusion.

Common Problems:
The Clique Problem
The Independent Set Problem

Every NP-Complete problem is a subset of NP.
The opposite of contradictory is satisfiable.
The **Boolean Satisfiability Problem**: Given a boolean expression e, does e have a satisfying assignment?

$L_{\text{IndependentSet}}$ is NP-Complete
Every NP-problem reduces to a NP-complete problem in Polynomial time.
L_{Sat} is NP-Complete

Other Well Known NP-Complete Problems:
Traveling Salesman Problem
Integer Programming (Note: Linear Programming is in P)
Lots of Design Problems
Clique Problem
Firehouse Problem
"Putting circuits on a board" is NP-Complete
"Optimizing a circuit board" is NP-Complete

Finding winning move for a chess-like board game is not NP.

Other notations:
RP - Randomized Polynomial Time
QP - Quantum Polynomial Time
NC - Nick's Class (Parallel Computing)

If L_1 and L_2 are CFLs which of the following are CFL?
L_1L_2 (Is CFL)
L_1* (Is CFL)
L_1^R (L reverse. Is CFL)
$L_1 \cap L_2$ (Is not CFL)
$L = a^n b^n c^n | n \geq 0$ (Is not CF, by Pumping Lemma for CFL)

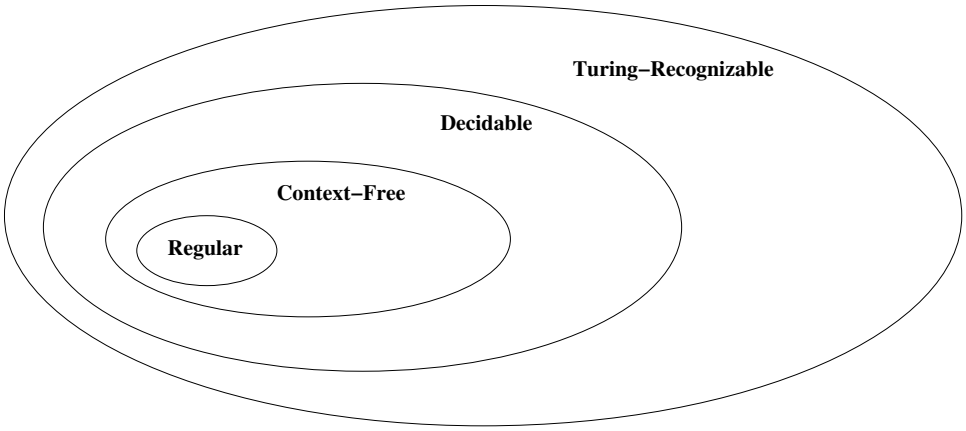


Figure 1: Relationship among Classes of Languages

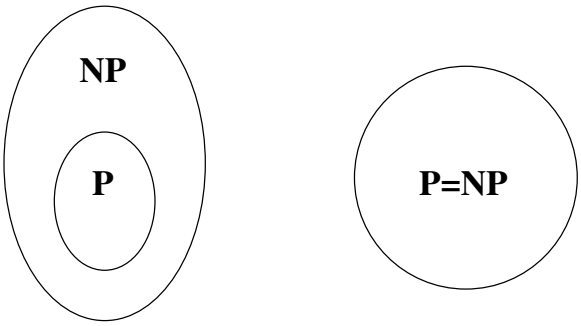


Figure 2: Two Possible Relations of P and NP

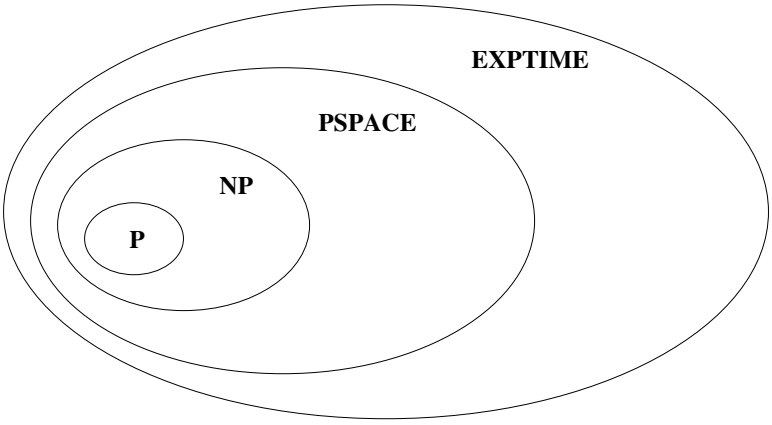


Figure 3: Conjectured Relationship

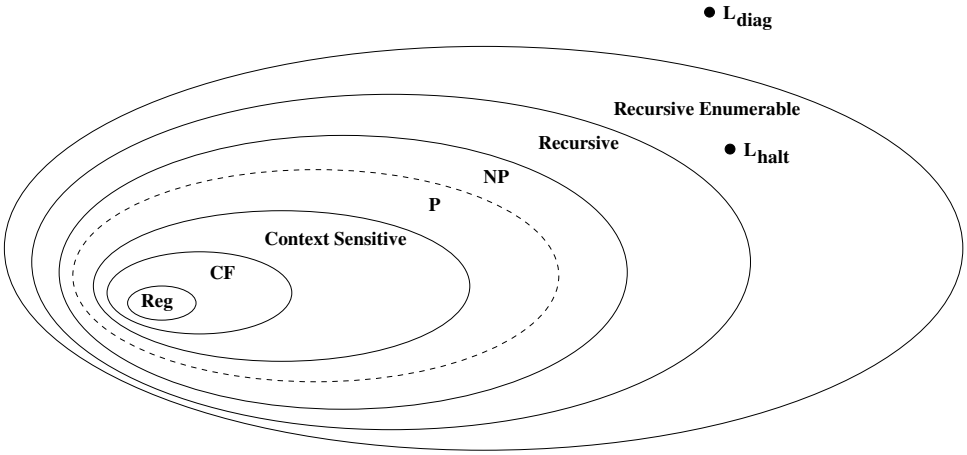


Figure 4: Larmore's Diagram of Classes

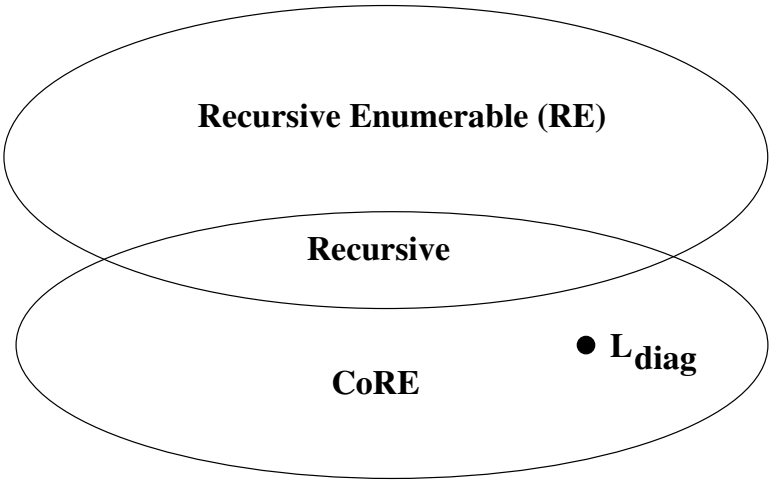


Figure 5: Larmore's Diagram of Recursive Relations

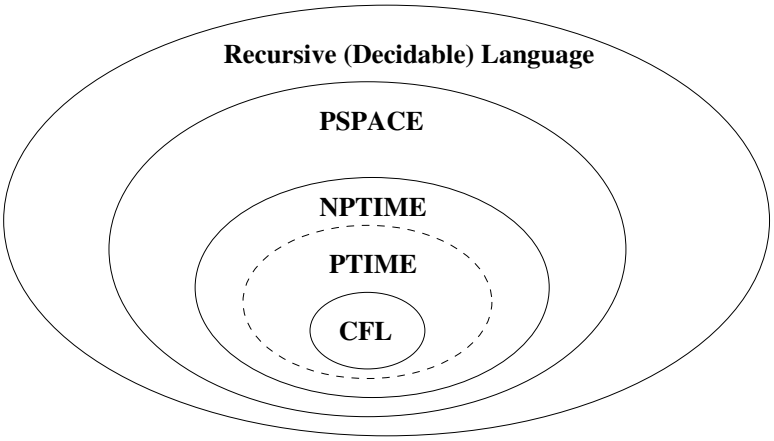


Figure 6: Another Larmore Diagram of Classes

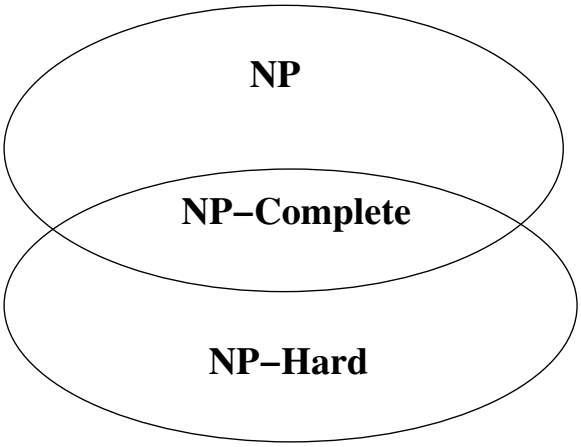


Figure 7: Relations of NP Problems

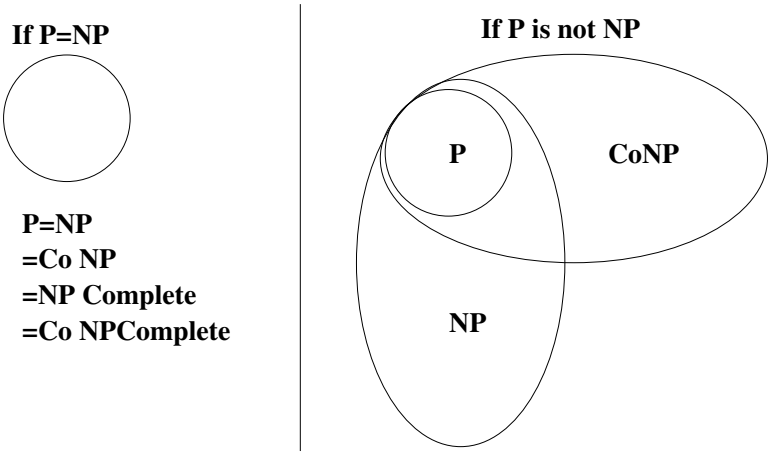


Figure 8: More P vs NP Stuff