1. **AMBIGUITY** :If a grammar generates the same string in several different ways, we say that the string is derived ambiguously in that grammar. If a grammar generates some string ambiguously, we say that the grammar is ambiguous.

QUESTION 2:

Difference between DFA and NFA :

DFA	NFA
DFA stands for Deterministic Finite Automata.	NFA stands for Nondeterministic Finite Automata.
For each symbolic representation of the alphabet, there is only one state transition in DFA.	No need to specify how does the NFA react according to some symbol.
DFA cannot use Empty String transition.	NFA can use Empty String transition.
DFA can be understood as one machine.	NFA can be understood as multiple little machines computing at the same time.
In DFA, the next possible state is distinctly set.	In NFA, each pair of state and input symbol can have many possible next states.

2. DFA is more difficult NFA is easier to

QUESTION 3

Formal definition of Context-Free Grammar

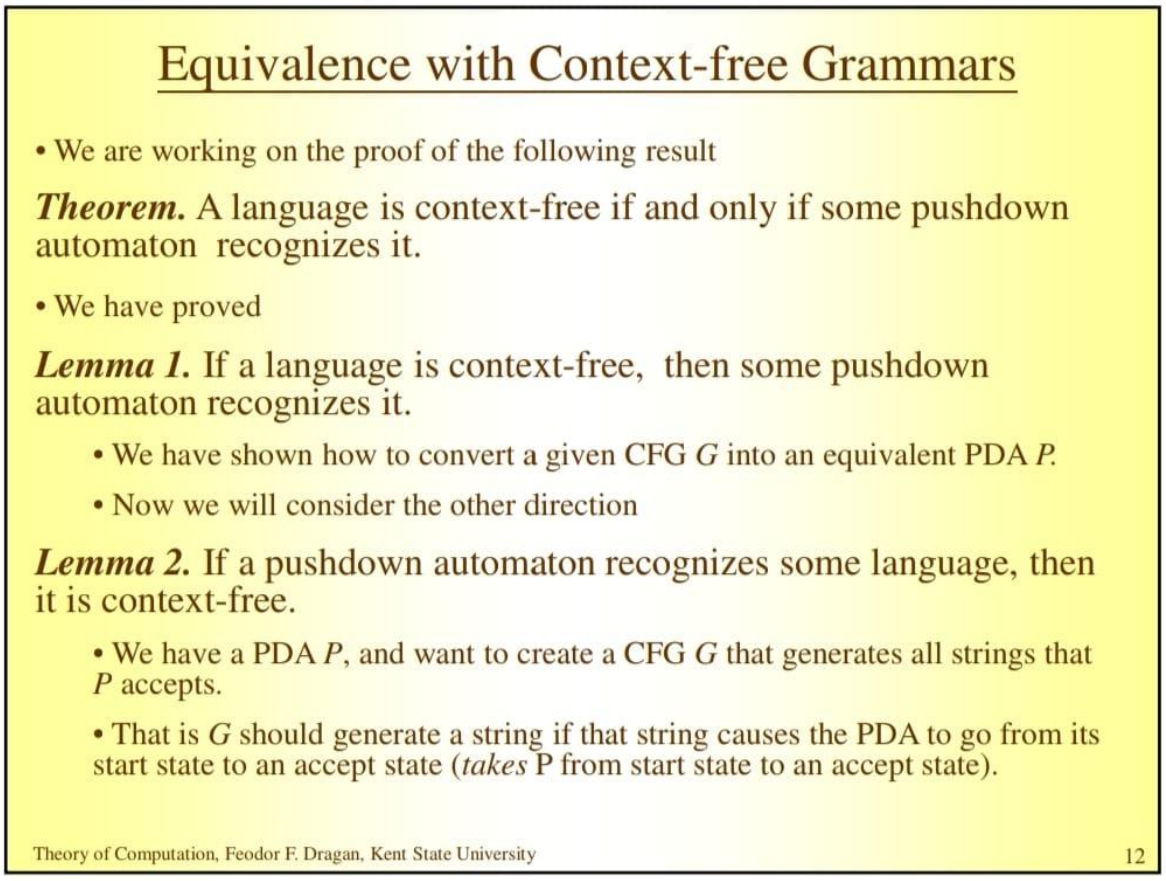
Context free grammar G can be defined by four tuples as: G= (V, T, P, S) Where, G describes the grammar

T describes a finite set of terminal symbols. V describes a finite set of non-terminal symbols

P describes a set of production rules

S is the start symbol.

QUESTION 4



Equivalence with Context-free Grammars

- We are working on the proof of the following result

Theorem. A language is context-free if and only if some pushdown automaton recognizes it.

- We have proved

Lemma 1. If a language is context-free, then some pushdown automaton recognizes it.

  - We have shown how to convert a given CFG *G* into an equivalent PDA *P.*
  - Now we will consider the other direction

Lemma 2. If a pushdown automaton recognizes some language, then it is context-free.

  - We have a PDA *P*, and want to create a CFG *G* that generates all strings that *P* accepts.
  - That is *G* should generate a string if that string causes the PDA to go from its start state to an accept state (*takes* P from start state to an accept state).

Theory of Computation, Feodor F. Dragan, Kent State University 12

QUESTION 5 :In computer science and mathematics, a language is a set of strings made up of symbols from an alphabet. A language is called decidable if an algorithm can determine whether a given string is a member of that language.

Question 6:

Finite Automata	Turing Machine
It recognizes the language called regular language.	It will recognize not only regular language but also context-free language, context-sensitive language, and recursively enumerable languages.
In this, the input tape is of finite length from both the left and right sides.	In this, the input tape is of finite length from the left but is of infinite length from the right.
It consists of a finite number of states, a finite set of input symbols, an initial state of automata, and a finite set of transition rules for moving from one state to another.	It also contains a finite set of tape symbols and a blank symbol on the tape in addition to a finite number of states, a finite set of input symbols, an initial state of automata, and a finite set of transition rules for moving from one state to another.
In this head is able to move in the right direction only. In two-way automata, the head is able to move in both directions.	In this, the head can move in both directions.
The Head is only able to read the symbols from the tape but can not write symbols on the tape.	The Head is able to read as well as write symbols on the tape.
It is weak as compared to Turing Machine.	It is more powerful than Finite Automata.

QUESTION 7:

P Class

The P in the P class stands for **Polynomial Time.** It is the collection of decision problems(problems with a “yes” or “no” answer) that can be solved by a deterministic machine in polynomial time.

NP Class

The NP in NP class stands for **Non-deterministic Polynomial Time**. It is the collection of decision problems that can be solved by a non-deterministic machine in polynomial time.

NP-complete class

A problem is NP-complete if it is both NP and NP-hard. NP-complete problems are the hard problems in NP

NP-complete languages are significant because all NP-complete languages are thought of having similar hardness, in that process solving one implies that others are solved as well.

If some NP-complete languages are proven to be in P, then all of NPs are proven to be in P. Because, note that any NP language can be reduced to an NP-complete language. Use this reduction and the algorithm for the given NP-complete problem to solve any problem in NP. Hence all of them will have polynomial time solutions.

If some NP-complete language is not in P, then this means this language is in NP but not in P, hence this certifies that P and NP are not equal.

Hence P vs. NP can be resolved if some NP-complete problem is proven to either be in P or not in P.

QUESTION EXTRA: Is the problem is decidable or not . IS a number ‘m’ prime?

Example 1

Find out whether the following problem is decidable or not –

Is a number ‘m’ prime?

Solution

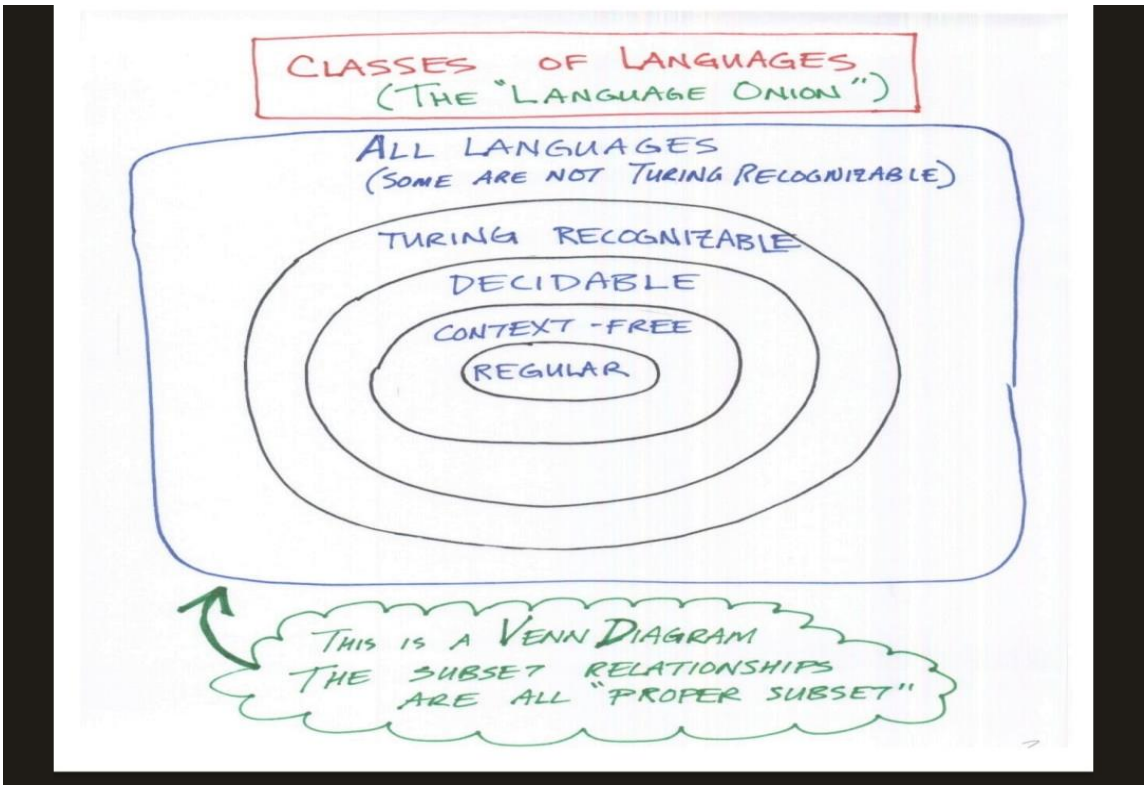
Prime numbers = {2, 3, 5, 7, 11, 13, ...}

Divide the number **‘m’** by all the numbers between ‘2’ and ‘√m’ starting from ‘2’.

If any of these numbers produce a remainder zero, then it goes to the "Rejected state", otherwise it goes to the "Accepted state". So, here the answer could be made by ‘Yes’ or ‘No’.

Hence, it is a decidable problem.

QUESTION 8:



QUESTION 9:

In theory, a problem solvable by a nondeterministic machine can also be solved by a deterministic machine. However, the deterministic machine might require more time (computation steps) to solve the problem, but it would not require extra space.

This is known as the "nondeterministic time versus deterministic time" relationship, and it's a central concept in theoretical computer science. If a problem can be solved by a nondeterministic Turing machine in a certain amount of time, then it can also be solved by a deterministic Turing machine, but the deterministic machine may require more time. In other words, nondeterministic machines can explore multiple possibilities in parallel, potentially leading to faster solutions, while deterministic machines have to follow a single path and may take longer.

It's important to note that this relationship is a theoretical concept and doesn't necessarily have practical implications for all real-world computation. Nondeterministic machines are used in theoretical discussions to help analyze the complexity of problems and algorithms. In practice, many problems can be efficiently solved by deterministic algorithms, and the use of nondeterministic machines is more of a theoretical tool for understanding computational complexity classes like P and NP.

QUESTION 10:

Context-Free Languages

If L is a CFL, then $\exists p$ (pumping length) such that $\forall z \in L$, if $|z| \geq p$ then $\exists u, v, w, x, y$ such that $z = uvwxy$

1. $|vwx| \leq p$
2. $|vx| > 0$

1

[redacted]

3. $\forall i \geq 0.\ uv^i wx^i y \in L$

QUESTION 11:

RE:

$$(a \cup b)^* \cup ab(a \cup b)^*$$

CFG:

Terminals:

$$S_1 \rightarrow a$$
$$S_2 \rightarrow b$$

This is for the first $(a + b)^*$*:*

$$S_3 \rightarrow S_1 \mid S_2 \qquad (a \cup b)$$
$$S_4 \rightarrow S_3 S_4 \mid \epsilon \qquad (a \cup b)^*$$

This is for the ab *in the middle:*

$$S_5 \rightarrow a$$
$$S_6 \rightarrow b$$

This is for the second $(a \cup b)$ *and* (ab) *in the middle:*

$$S_7 \rightarrow S_1 \mid S_2 \qquad (a \cup b)$$
$$S_8 \rightarrow S_7 S_8 \mid \epsilon \qquad (a \cup b)^*$$
$$S_9 \rightarrow S_5 S_6 \qquad (ab)$$

Concatenated ab *with the second* $(a \cup b)^*$*:*

$$S_{10} \rightarrow S_8 S_9 \qquad (ab(a \cup b)^*)$$

This the final CFG:

$$S_{11} \rightarrow S_4 \mid S_{10} \qquad (a \cup b)^* \cup ab($$

QUESTION 12:

Removal of Null Productions

In a CFG, a non-terminal symbol **'A'** is a nullable variable if there is a production **A → ε** or there is a derivation that starts at **A** and finally ends up with

ε: A → ..... → ε

Removal Procedure

Step 1 – Find out nullable non-terminal variables which derive ε.

Step 2 – For each production **A → a**, construct all productions **A → x** where **x** is obtained from **'a'** by removing one or multiple non-terminals from Step 1.

Step 3 – Combine the original productions with the result of step 2 and remove **ε - productions**.

Removal of Unit Productions

Any production rule in the form A → B where A, B ∈ Non-terminal is called **unit production.**.

Removal Procedure –

Step 1 – To remove **A → B**, add production **A → x** to the grammar rule whenever **B → x** occurs in the grammar. [x ∈ Terminal, x can be Null]

Step 2 – Delete **A → B** from the grammar.

Step 3 – Repeat from step 1 until all unit productions are removed.

QUESTION 16:

A PDA is more powerful than FA. Any language which can be acceptable by FA can also be acceptable by PDA. PDA also accepts a class of language which even cannot be accepted by FA. Thus PDA is much more superior to FA.

QUESTION 17:

turing machine is defined as an abstract representation of a computing device such as hardware in computers. Alan Turing proposed Logical Computing Machines (LCMs), i.e. Turing's expressions for Turing Machines. This was done to define algorithms properly. So, Church made a mechanical method named as 'M' for manipulation of strings by using logic and mathematics. This method M must pass the following statements:

Number of instructions in M must be finite.

Output should be produced after performing finite number of steps.

It should not be imaginary, i.e. can be made in real life.

It should not require any complex understanding.

Using these statements Church proposed a hypothesis called

QUESTION 18:

 EDFA is decidable

Theorem

The language EDFA = {⟨ B⟩ | B is a DFA and L(B) = ∅ } is decidable

Proof. Let's build a TM M to decide EDFA:

M = "On input ⟨ B⟩ , 1. Mark the initial state of B

2. Repeat until no new states are marked

3. Mark any state that has an incoming transition from a marked state

4. If any accept state is marked, reject; otherwise accept"

If L(B) ≠ ∅ , then there must be a path from the initial state to an accept state so M will reject. If L(B) = ∅ , then no such path exists and M will accept.

EQDFA is decidable

Theorem

The language EQDFA = {⟨ A, B⟩ | A and B are DFAs and L(A) = L(B)} is decidable

Proof. Let R be a decider for EDFA and build

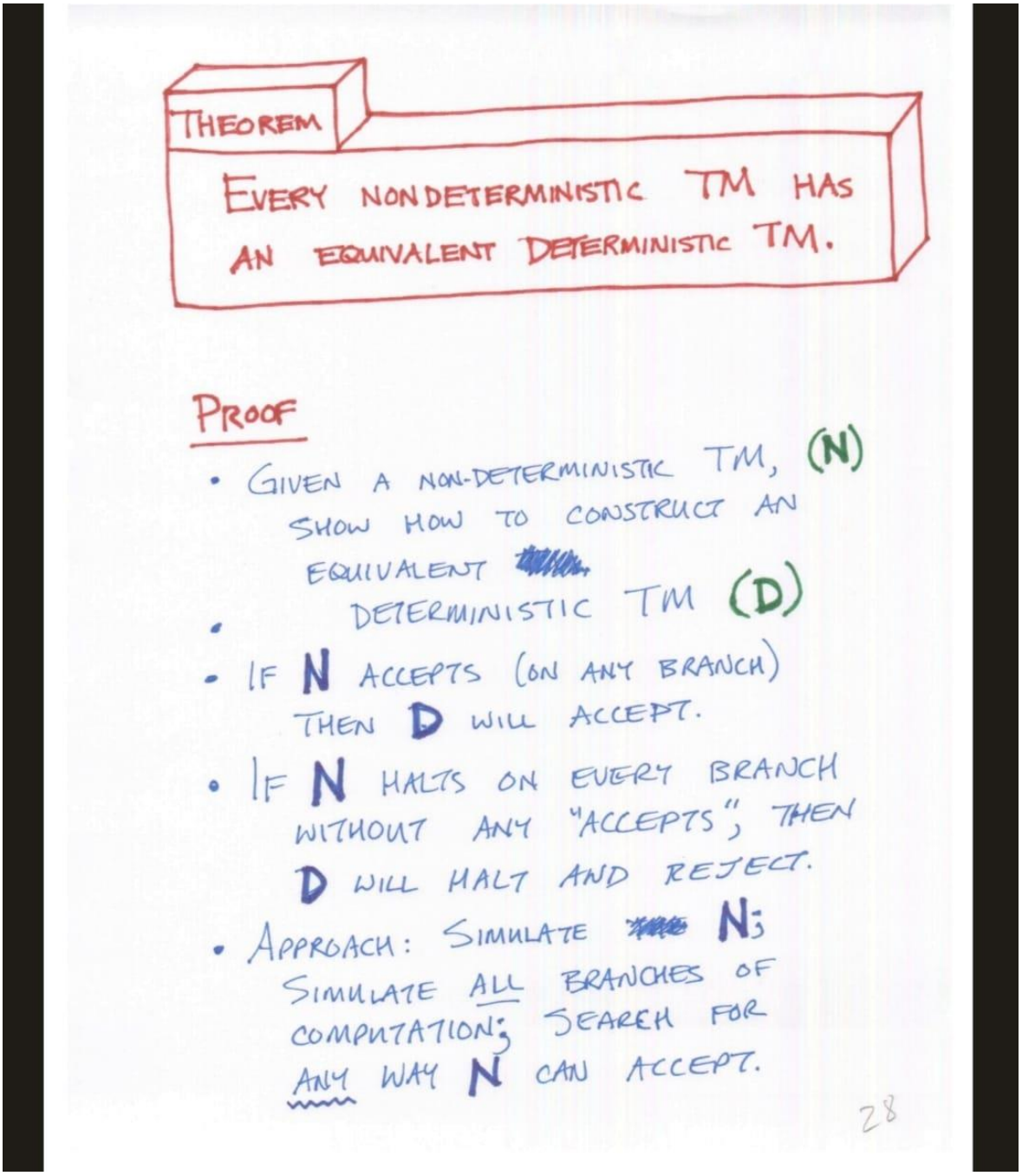
M = “On input ⟨ A, B ⟩ , 1. Construct DFA C where L(C) = (L(A) \L(B)) ∪ (L(B) \L(A))

2. Run R on <C>

3. If R accepts, accept; otherwise reject” M is a decider because R is a decider

L(C) = Ø iff L(A) = L(B) so M accepts iff L(A) = L(B)

QUESTION 20



QUESTION 21:

Halting problem is undecidable Proof

- We modify *H* to construct a new TM *H'*, which behaves very much as *H*:
  - The computations of *H* are the same as *H*, except that *H* loops indefinitely whenever *H* terminates in an accepting state, i.e., whenever *H* halts on input *w*, and halts otherwise