

INT201 Decision, Computation and Language

Lecture 12 – Review

Dr Chunchuan Lyu



Xi'an Jiaotong-Liverpool University

西交利物浦大學

Summary of Summary

What can machines do and can not do?

Regular Languages

Deterministic Finite Automata (DFA)

Regular Expression

Nondeterministic Finite Automata (NFA)

Context Free Languages

Nondeterministic Push-down Automata (PDA)

Turing-Decidable Languages

Turing Machine that Always Halts (Decider)

Turing-Recognizable Languages

Turing Machine

All Languages

Nothing that we can enumerate
(by the countability argument)

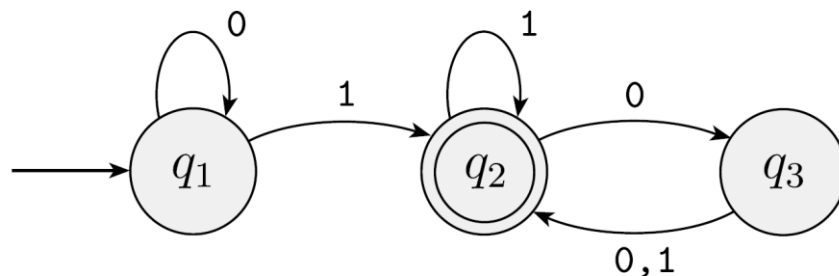


Deterministic Finite Automata and Regular Language

A *finite automaton* is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the *states*,
2. Σ is a finite set called the *alphabet*,
3. $\delta: Q \times \Sigma \longrightarrow Q$ is the *transition function*,¹
4. $q_0 \in Q$ is the *start state*, and
5. $F \subseteq Q$ is the *set of accept states*.²

If A is the set of all strings that machine M accepts, we say that A is the *language of machine* M and write $L(M) = A$.

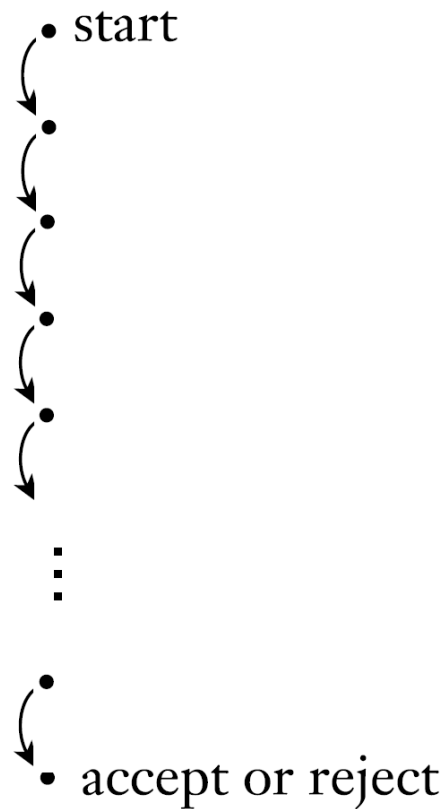


A language is called a *regular language* if some finite automaton recognizes it.

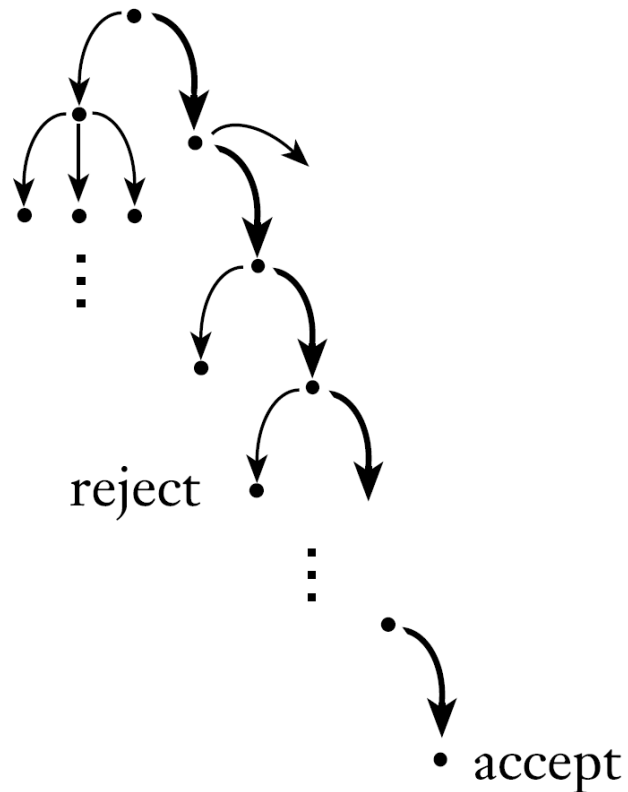


Nondeterministic Finite Automata

Deterministic
computation



Nondeterministic
computation



Accept as long as
one branch can
accept the string

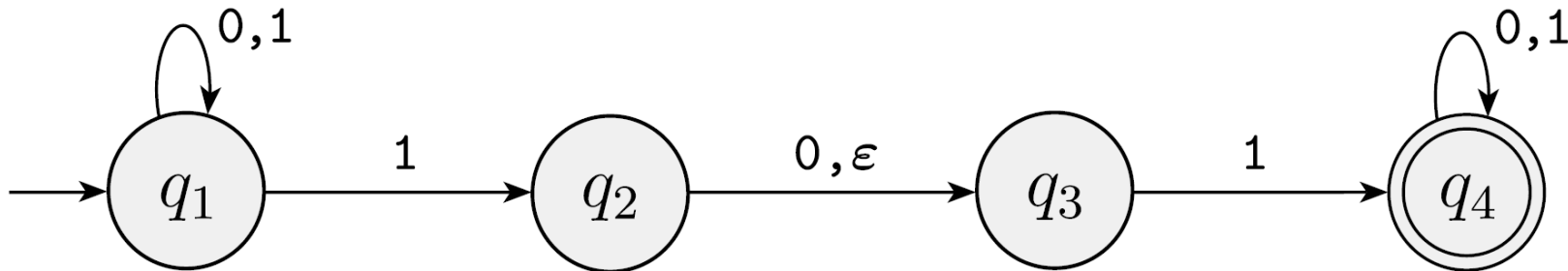


Nondeterministic Finite Automata

A *nondeterministic finite automaton* is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set of states,
2. Σ is a finite alphabet,
3. $\delta: Q \times \Sigma_{\epsilon} \longrightarrow \mathcal{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

Multiple transition in the same state for the same string, defined by transition function has range in the power set.



Nondeterministic Finite Automata

Theorem:

Every nondeterministic finite automaton has an equivalent deterministic finite automaton.

Proof idea:

Construct an equivalent DFA that simulates NFA by tracking the **set of all possible states** the NFA could be in at any given moment.



Regular Language Closure Properties

Theorem

The class of regular languages is closed under the **complement, union, intersection, concatenation, and Kleene star** operation.

Prove by constructing a new DFA/NFA to accept the language or relying on proven closure property.



Regular Expression

Say that R is a *regular expression* if R is

1. a for some a in the alphabet Σ ,
2. ϵ ,
3. \emptyset ,
4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions,
5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions, or
6. (R_1^*) , where R_1 is a regular expression.

Theorem

A language is regular if and only if some regular expression describes it.

Proof idea:

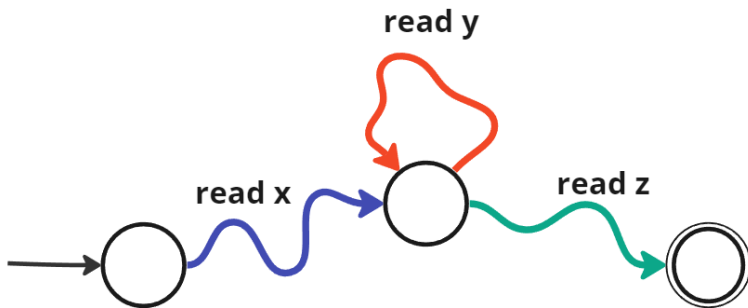
Build NFA through its' closure property and build RE from generalized NFA whose transitions are REs.



Regular Language Pumping Lemma

Pumping lemma If A is a regular language, then there is a number p (the pumping length) where if s is any string in A of length at least p , then s may be divided into three pieces, $s = xyz$, satisfying the following conditions:

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.



Proof idea:

DFA cannot recognize arbitrary long strings without repeating its state.

Application:

$\{0^n 1^n | n \geq 0\}$ is non regular.



Context-free Language

A *context-free grammar* is a 4-tuple (V, Σ, R, S) , where

1. V is a finite set called the *variables*,
2. Σ is a finite set, disjoint from V , called the *terminals*,
3. R is a finite set of *rules*, with each rule being a variable and a string of variables and terminals, and
4. $S \in V$ is the start variable.

Example:

$$G = (\{S\}, \{0, 1\}, \{S \rightarrow 0S1 \mid \epsilon\}, S)$$

$$L(G) = \{0^n 1^n \mid n \geq 0\}$$



Chomsky Normal Form

A context-free grammar is in *Chomsky normal form* if every rule is of the form

$$A \rightarrow BC$$

$$A \rightarrow a$$

where a is any terminal and A , B , and C are any variables—except that B and C may not be the start variable. In addition, we permit the rule $S \rightarrow \varepsilon$, where S is the start variable.

All variable generate two non-start variables or a terminal symbol.
Only start variable generates empty string.

CNF ruled out unnecessary production rules and constrained the derivation search space.



Chomsky Normal Form is Sufficient

A context-free grammar is in *Chomsky normal form* if every rule is of the form

$$A \rightarrow BC$$

$$A \rightarrow a$$

where a is any terminal and A , B , and C are any variables—except that B and C may not be the start variable. In addition, we permit the rule $S \rightarrow \varepsilon$, where S is the start variable.

Theorem

Any context-free language is generated by a context-free grammar in Chomsky normal form.

Proof idea:

Non start variables that generate empty string can be avoided in the first place.

Long production rules can be decomposed.

Unary production rules can be eliminated.



Pushdown Automata

A *pushdown automaton* is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q , Σ , Γ , and F are all finite sets, and

1. Q is the set of states,
2. Σ is the input alphabet,
3. Γ is the stack alphabet,
4. $\delta: Q \times \Sigma_\epsilon \times \Gamma_\epsilon \longrightarrow \mathcal{P}(Q \times \Gamma_\epsilon)$ is the transition function,
5. $q_0 \in Q$ is the start state, and
6. $F \subseteq Q$ is the set of accept states.

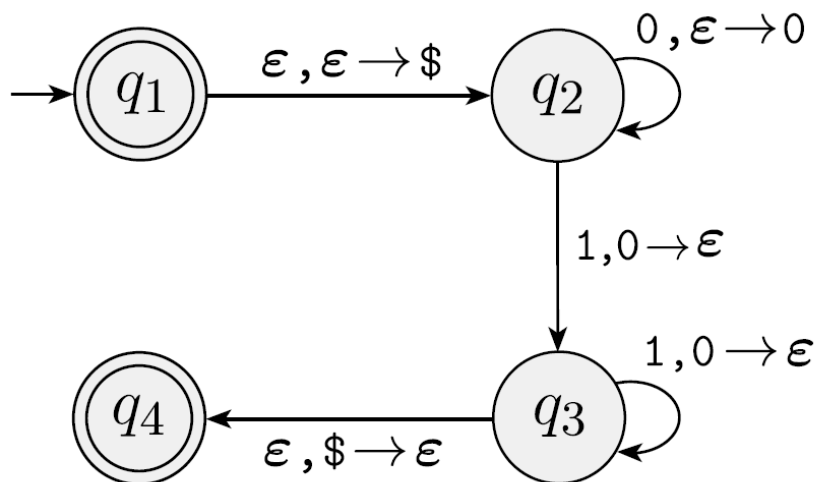
(Standard) PDA is non-deterministic. They are NFA with a stack.
Deterministic PDA is weaker than PDA.



Pushdown Automata

A **pushdown automaton** is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q , Σ , Γ , and F are all finite sets, and

1. Q is the set of states,
2. Σ is the input alphabet,
3. Γ is the stack alphabet,
4. $\delta: Q \times \Sigma_\epsilon \times \Gamma_\epsilon \longrightarrow \mathcal{P}(Q \times \Gamma_\epsilon)$ is the transition function,
5. $q_0 \in Q$ is the start state, and
6. $F \subseteq Q$ is the set of accept states.



$$L(M) = \{0^n 1^n \mid n \geq 0\}$$



Pushdown Automata and CFG are equivalent

Theorem

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

Proof idea:

PDA simulate CNF by replacing symbols on the stack.

All PDA can be simplified so that they start and ends with empty stack, have a unique accepting state and have simple transitions.

CFG production rules can cover all states to states transitions that did not look at the stack context.



Context Free Language Closure Properties

Theorem

The class of context free languages is closed under the **union**, **concatenation**, and **Kleene star** operation.

Proof idea:

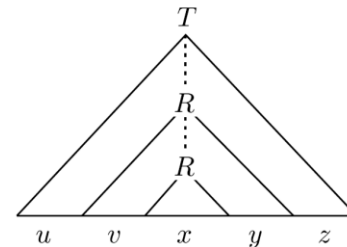
Easy to see from the PDA perspective.



Pumping Lemma for Context Free Language

Pumping lemma for context-free languages If A is a context-free language, then there is a number p (the pumping length) where, if s is any string in A of length at least p , then s may be divided into five pieces $s = uvxyz$ satisfying the conditions

1. for each $i \geq 0$, $uv^i xy^i z \in A$,
2. $|vy| > 0$, and
3. $|vxy| \leq p$.

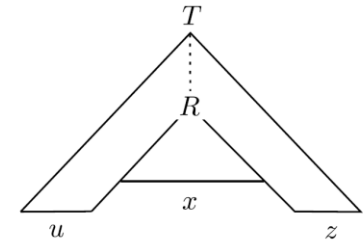
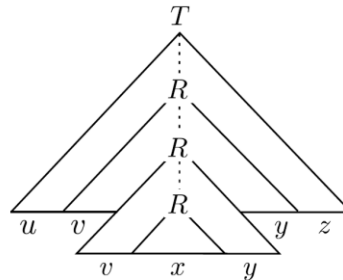


Proof idea:

CFG cannot generate arbitrary long strings without repeating its variable.

Application:

$\{ww \mid w \in \{0,1\}^*\}$ is non context free.



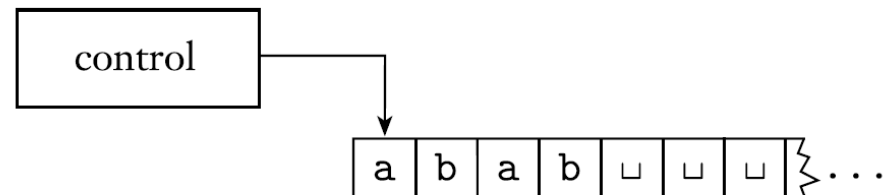
Take time to fill the MQ



Turing Machine

A **Turing machine** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where Q, Σ, Γ are all finite sets and

1. Q is the set of states,
2. Σ is the input alphabet not containing the **blank symbol** \sqcup ,
3. Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the transition function,
5. $q_0 \in Q$ is the start state,
6. $q_{\text{accept}} \in Q$ is the accept state, and
7. $q_{\text{reject}} \in Q$ is the reject state, where $q_{\text{reject}} \neq q_{\text{accept}}$.

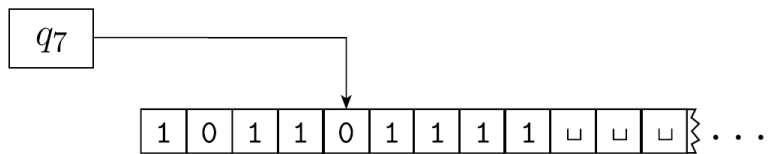


They are DFA with a writable infinite tape.

Accepts or rejects once it is in accept state or reject state.



Turing Machine Configuration



A Turing machine with
configuration $1011q_701111$

$\delta(q_i, b) = (q_j, c, L)$ gives
 $C_t = uaq_i bv$ yields $C_{t+1} = uq_j acv$

A Turing machine M **accepts** input w if a sequence of configurations C_1, C_2, \dots, C_k exists, where

1. C_1 is the start configuration of M on input w ,
2. each C_t yields C_{t+1} , and
3. C_k is an accepting configuration.



Recognizable Language and Decidable Language

Call a language A **Turing-recognizable** if some Turing machine M recognizes it in the sense that $\forall w \in \Sigma^*, w \in A$ if and only if M accepts w .

Call a language A **Turing-decidable** if some Turing machine (*decider*) M decides it in the sense that $\forall w \in \Sigma^*$, if $w \in A$, M accepts w , and if $w \notin A$, M rejects it.

Clearly, all decidable languages are recognizable.



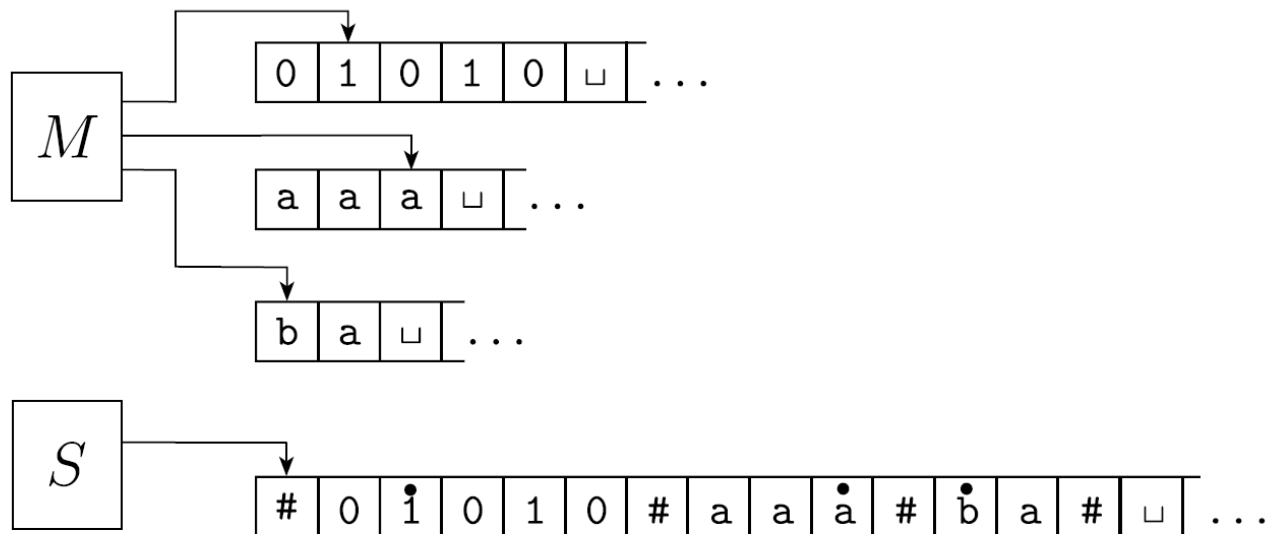
Multi-Tape Turing Machine

$$\delta: Q \times \Gamma^k \longrightarrow Q \times \Gamma^k \times \{L, R, S\}^k,$$

Theorem

Every multi-tape Turing machine has an equivalent single-tape Turing machine.

Proof idea:



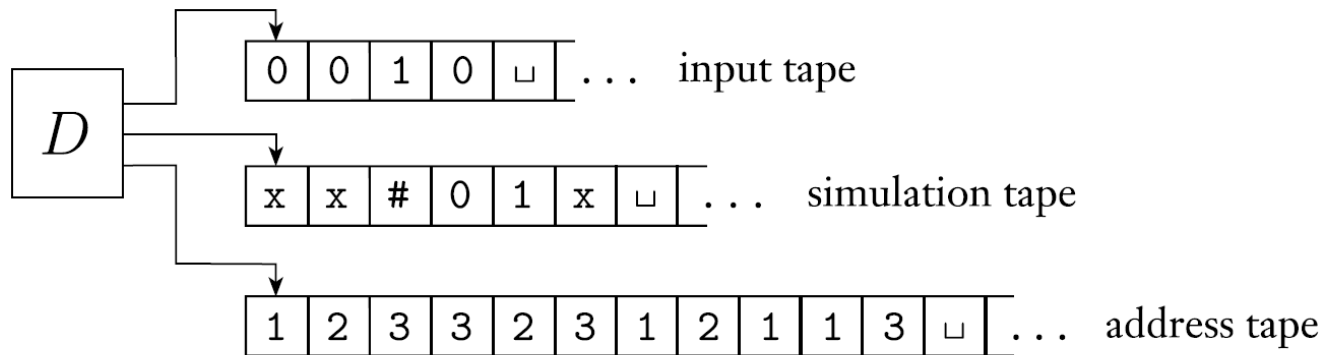
Nondeterministic Turing Machine

$$\delta: Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\}).$$

Theorem

Every nondeterministic Turing machine has an equivalent deterministic Turing machine.

Proof idea:



Check all branches of NTM through BFS over computation history recorded in the address tape.



Existence of Non-Turing Recognizable Languages

Theorem

There exists non Turing-recognizable languages.

Proof:

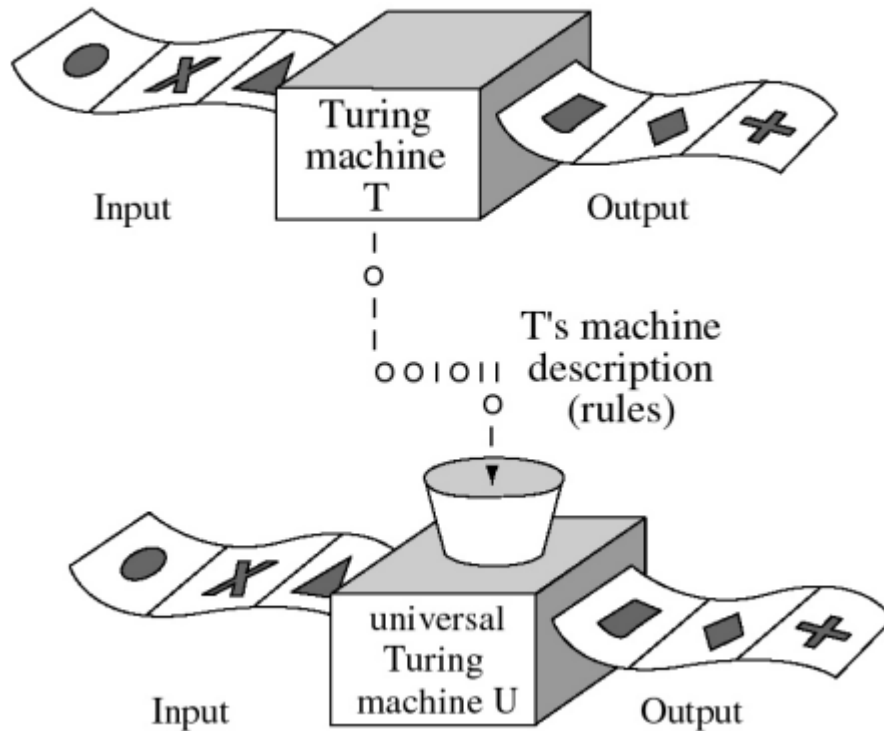
There are uncountable many languages, but countable many Turing machines.



Universal Turing Machines

Theorem

There exists a TM U that takes a Turing machine description and input tape and simulate one step of that given Turing machine on the input tape.



Proof idea:

Direct construction with multi-tape Turing machine.

<https://www.geeksforgeeks.org/universal-turing-machine/>



Turing Machines High Level Description

$A = \{\langle G \rangle \mid G \text{ is a connected undirected graph}\}.$

The following is a high-level description of a TM M that decides A .

$M =$ “On input $\langle G \rangle$, the encoding of a graph G :

1. Select the first node of G and mark it.
2. Repeat the following stage until no new nodes are marked:
3. For each node in G , mark it if it is attached by an edge to a node that is already marked.
4. Scan all the nodes of G to determine whether they all are marked. If they are, *accept*; otherwise, *reject*.”

TM can perform computation on high level data structure that modern computers can perform.

In reality everything is encoded in binary after all.



Turing Language Closure Properties

Theorem

The class of Turing recognizable languages is closed under ?

Theorem

The class of Turing decidable languages is closed under ?

Left to you, as Ass2 has a question on this.

Think about whether they have the same closure properties.



Decidable Languages

- $L_{\text{DFA}} = \{\langle B, w \rangle \mid B \text{ is a DFA that accept } w\}$
- $L_{\text{CFG}} = \{\langle G, w \rangle \mid G \text{ is a CFG that generates string } w\}$
- $E_{\text{CFG}} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G)=\emptyset\}$

Intuitively, we consider decidable languages to be solvable by Turing machines.

In practice, it means there is an algorithm that runs in finite time (given any input) and decide whether the input is in the language or not.



Acceptance Problem is Recognizable but Undecidable

$L_{TM} = \{ \langle M, w \rangle : M \text{ is a Turing machine that accepts the string } w \}$

Theorem

L_{TM} is Turing-recognizable but not decidable.

Proof idea:

UTM can simulate M on w , so it is Turing-recognizable.

If a decider H exists:

$D =$ “On input $\langle M \rangle$, where M is a TM:

1. Run H on input $\langle M, \langle M \rangle \rangle$.
2. Output the opposite of what H outputs. That is, if H accepts, *reject*; and if H rejects, *accept*.”

D is paradoxical.



Unrecognizable Languages

Theorem

A language A is decidable if and only if it is Turing-recognizable and co-Turing-recognizable.

Proof idea:

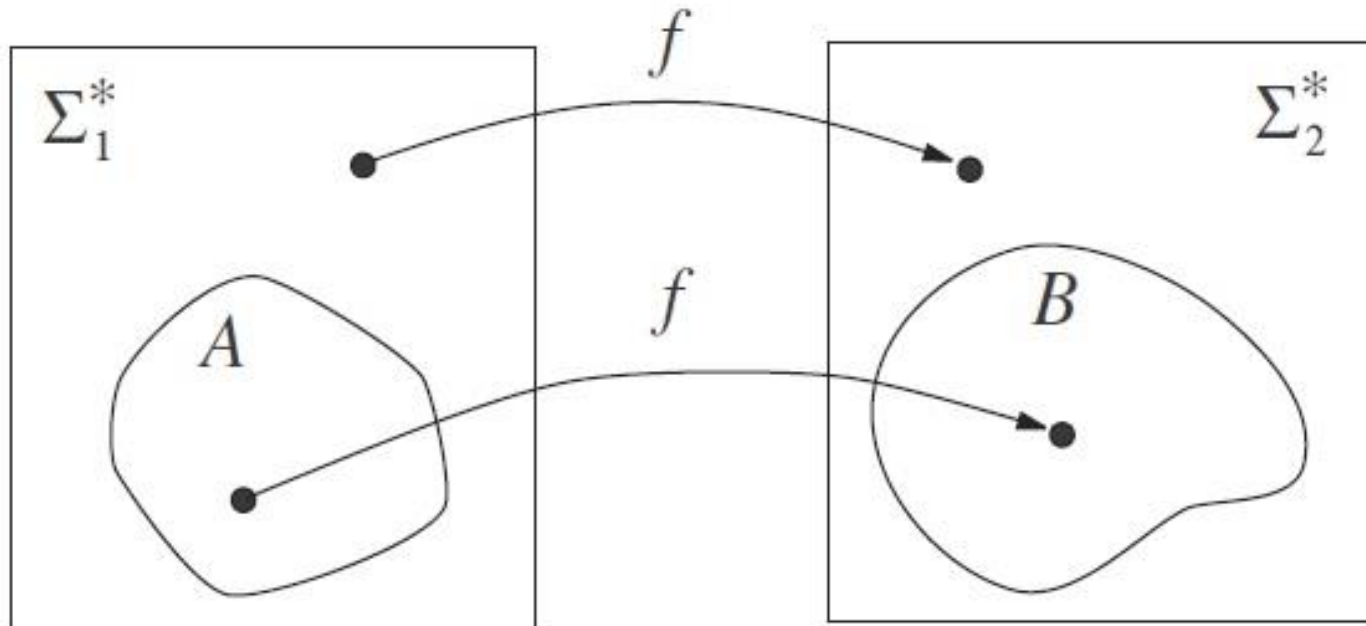
Only if part is easy.

For the if part, assuming recognizable and co-recognizable, we simulate two recognizers in parallel to form a decider.

Corollary $\overline{L_{TM}}$ is not Turing-recognizable.



Mapping Reduction



$$w \in A \iff f(w) \in B$$

We write A is mapping reducible to B as $A \leq_m B$



Reducibility and Halting Problem

Theorems

- If $A \leq_m B$ and B is decidable, then A is decidable.
- If $A \leq_m B$ and A is undecidable, then B is undecidable also.
- If $A \leq_m B$ and B is Turing-recognizable, then A is Turing-recognizable.
- If $A \leq_m B$ and A is not Turing-recognizable, then B is not Turing-recognizable also.

$H_{TM} = \{ \langle M, w \rangle : M \text{ is a Turing machine that halts on the string } w \}$

Theorem

H_{TM} is Turing-recognizable but not decidable.

Proof idea:

(General) Reduce undecidable L_{TM} to H_{TM} shows H_{TM} is undecidable.



Non-trivial Properties and Rice's Theorem

- $E_{TM} = \{ \langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset \}$.
- $INFINITE_{TM} = \{ \langle M \rangle \mid M \text{ is a TM and } |L(M)| = \infty \}$.
- $LT_{TM} = \{ \langle M \rangle \mid M \text{ is a TM and } L(M) = L(T) \}$.
- $FINITE_{TM} = \{ \langle M \rangle \mid M \text{ is a TM and } \exists n \in \mathbb{N}, |L(M)| = n \}$.
- $ALL_{TM} = \{ \langle M \rangle \mid M \text{ is a TM and } L(M) = \Sigma^* \}$

Turing machine descriptions of non-trivial properties P :

1. none empty set. i.e., $P \neq \emptyset$
2. none of them includes all Turing machines . i.e., $P \neq \{ \langle M \rangle \mid M \text{ is a TM} \}$
3. for any two Turing machines M_1 and M_2 with $L(M_1) = L(M_2)$,
 - (a) either both $\langle M_1 \rangle$ and $\langle M_2 \rangle$ are in P or
 - (b) none of $\langle M_1 \rangle$ and $\langle M_2 \rangle$ is in P .

Rice's Theorem

Any non-trivial property of Turing machines is undecidable



Rice's Theorem and Applications

Let P be a subset of Turing machine descriptions such that

1. $P \neq \emptyset$,
2. P is a proper subset of Turing machine descriptions ,
3. for any two Turing machines M_1 and M_2 with $L(M_1) = L(M_2)$,
 - (a) either both $\langle M_1 \rangle$ and $\langle M_2 \rangle$ are in P or
 - (b) none of $\langle M_1 \rangle$ and $\langle M_2 \rangle$ is in P .

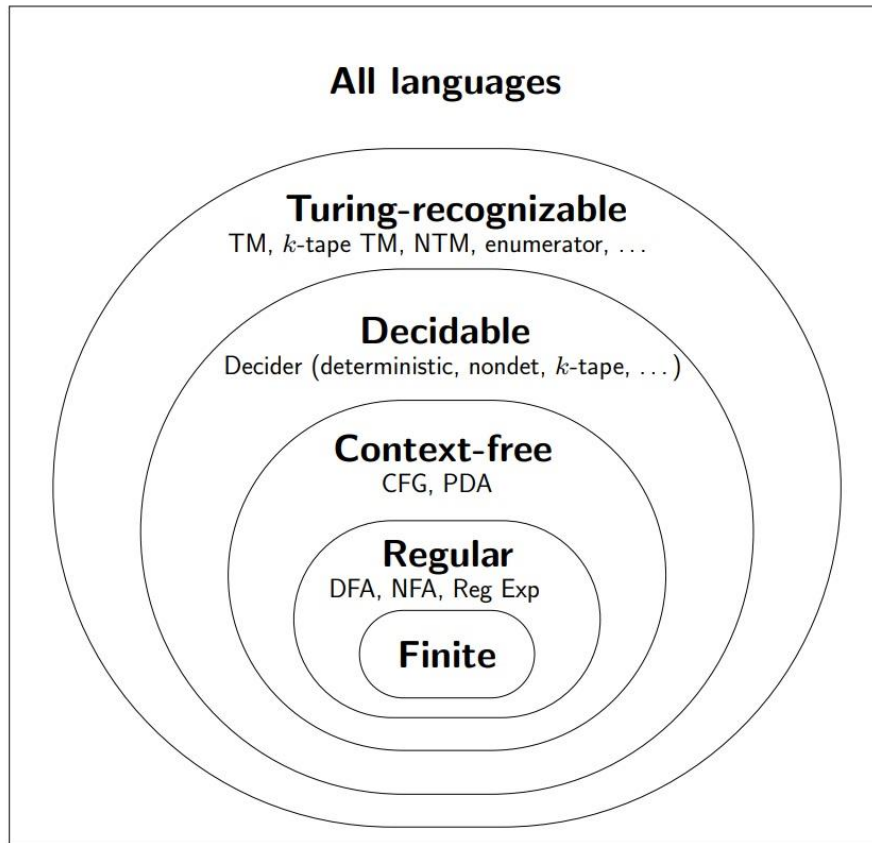
$$INFINITE_{TM} = \{ \langle M \rangle \mid M \text{ is a TM and } |L(M)| = \infty \}.$$

1. A TM accepts all inputs is in the set.
2. A TM rejects all inputs is not in the set.
3. If $L(M_1) = L(M_2)$, $|L(M_1)| = |L(M_2)|$ so they are both either infinite or finite, either both in the set or not in the set respectively.

Therefore $INFINITE_{TM}$ is undecidable by Rice's theorem



Language Hierarchy



Except enumerator (check Sipser),
you need to know them all.

You need to know their closure
properties.

You need to know examples that
distinguish the circles.

You need to know theorems that
allow you to distinguish the circles.



Are they useful?

DFA/RE is useful, because you do need them to find some strings when search over documents

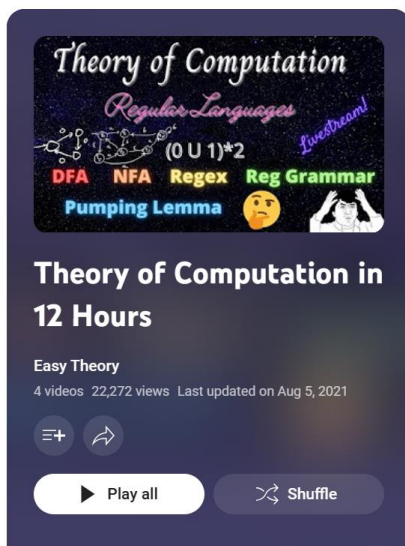
CFG/PDA are used for almost all programming language interpretation. All the compilers are PDA that turns one high level language into lower level ones. Even for large language models, see [XGrammar](#)

Knowing there are problems that cannot be solved exactly is useful

Builds foundation to formally talk about the computation complexity (INT202)



Highly Recommended Revision Resources



- 1 **Regular Languages in 4 Hours (DFA, NFA, Regex, Pumping Lemma, all conversions)**
Easy Theory • 24K views • Streamed 3 years ago
- 2 **Context-Free Languages in 3.5 Hours (CFG, PDA, Conversions, Closure, Pumping Lemma)**
Easy Theory • 21K views • Streamed 3 years ago
- 3 **Turing Machines + Decidability in 3 Hours (TM, Variants, Church-Turing, Decidability)**
Easy Theory • 19K views • Streamed 3 years ago
- 4 **Undecidability in 2 Hours (A_TM, Rice's Theorem, LBAs, EQ_CFG)**
Easy Theory • 7.9K views • 3 years ago

Theory of Computation in 12 Hours really nice explanation

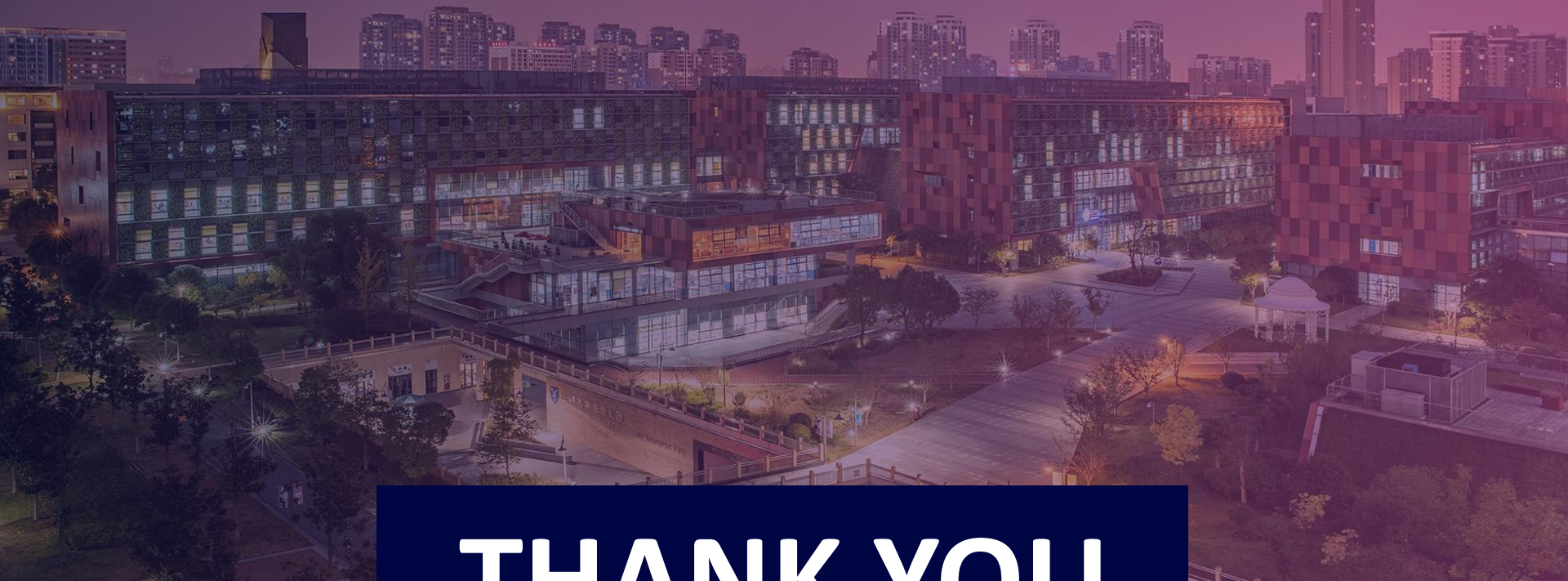
<https://github.com/ND-CSE-30151/spring-2025/tree/main> code to illustrate concepts

<https://ocw.mit.edu/courses/18-404j-theory-of-computation-fall-2020/> we covered a third

Email me at chunchuan.lyu@xjtlu.edu.cn at any time

Extra mini-Lecture about LLMs for formal reasoning + QA on Thursday





THANK YOU



Xi'an Jiaotong-Liverpool University
西交利物浦大學

XJTLU | SCHOOL OF
FILM AND
TV ARTS