INT201 Decision, Computation and Language

Lecture 10 – Church-Turing Thesis and Limits of Computation

Dr Yushi Li and Dr. Chunchuan Lyu

Xi’an Jiaotong-Liverpool University
西交利物浦大学

Recap

- Turing Machine
- Turing-recognizable (halts on accept) and Turing-decidable (always halts) languages
- Multi-tape TM and Nondeterministic TM

Today

- Cantor’s Diagonalization Method
- Church-Turing Thesis and Universal Turing Machine
- Examples of decidable languages
- Existence of undecidable language and non-Turing recognizable language

Diagonalization method

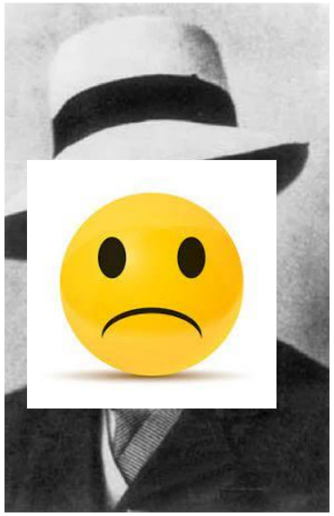
Questions:

If we use natural numbers to list natural numbers,
couldn't we construct a new number by flipping the diagonal and get a new number that is not in the natural number set?

Answer:

True, but the number you get is of infinite length (countable length).
All natural numbers are of finite length.
Therefore, you actually constructed a real number.
There is nothing wrong with a real number not in the natural number set.

A Brief History of the Limits of Computation



David Hilbert
1862-1943
German
Mathematician

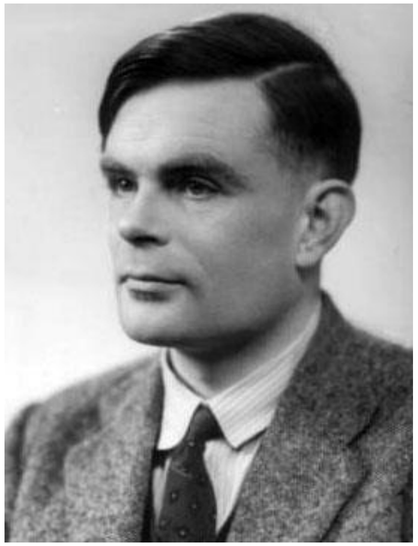
- The existence of uncountable sets - Georg Cantor 1874
- The diagonal method - Georg Cantor 1891
- Is there a set between real and natural numbers？ – David Hilbert 1900
- Prove axioms of arithmetic are consistent – David Hilbert 1900
- **We must know. We shall know – David Hilbert 1930**
- The existence of non-provable & non-disprovable statements
- Consistency of powerful system cannot be proved within itself - Kurt Gödel 1931
- Whether a statement is provable from axioms is not Turing-decidable – Church & Turing 1936
- Is there a set between real and natural numbers is independent of ZFC –Gödel 1940 & Cohen 1963
- **The Church–Turing Thesis**: every effectively calculable function (effectively decidable predicate) is general recursive (Turing computable)– Stephen Kleene 1952

The Church–Turing Thesis

Alonzo Church
1903 - 1995

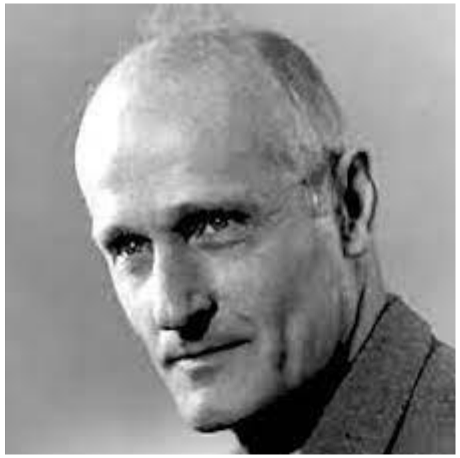
$$\boxed{\text{Algorithm}} = \boxed{\text{Turing machine}}$$

Intuitive   Formal



Alan Turing
1912–1954

How to prove this? Is it even possible?



Stephen Kleene
1909 -1994

How to disprove this? Is it even possible?

The Church–Turing Thesis
Existence of non Turing-recognizable languages.

Proposition: Each Turing machine can be encoded by a distinct, finite string of 1's and 0's and some finite special symbols.

Proof : encode TM as 7 tuple with special symbols, and encode alphabets in binary. Transitions can be encoded as a sequence of 5 tuples (state,tape,new state, new tape, left or right).

It is of critical importance that each single Turing machine is described in finite length.

Corollary: There are countable many Turing machines.

The Church–Turing Thesis
Existence of non Turing-recognizable languages.

Corollary: There are countable many Turing machines.

Proposition: There are uncountable many languages.

Proof
$$\begin{array}{rcccccccccccl}
\Sigma^* = \{ & \varepsilon, & 0, & 1, & 00, & 01, & 10, & 11, & 000, & 001, & \cdots & \} & ; \\
A = \{ & & 0, & & 00, & 01, & & & 000, & 001, & \cdots & \} & ; \\
\chi_A = & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & \cdots & & .
\end{array}$$

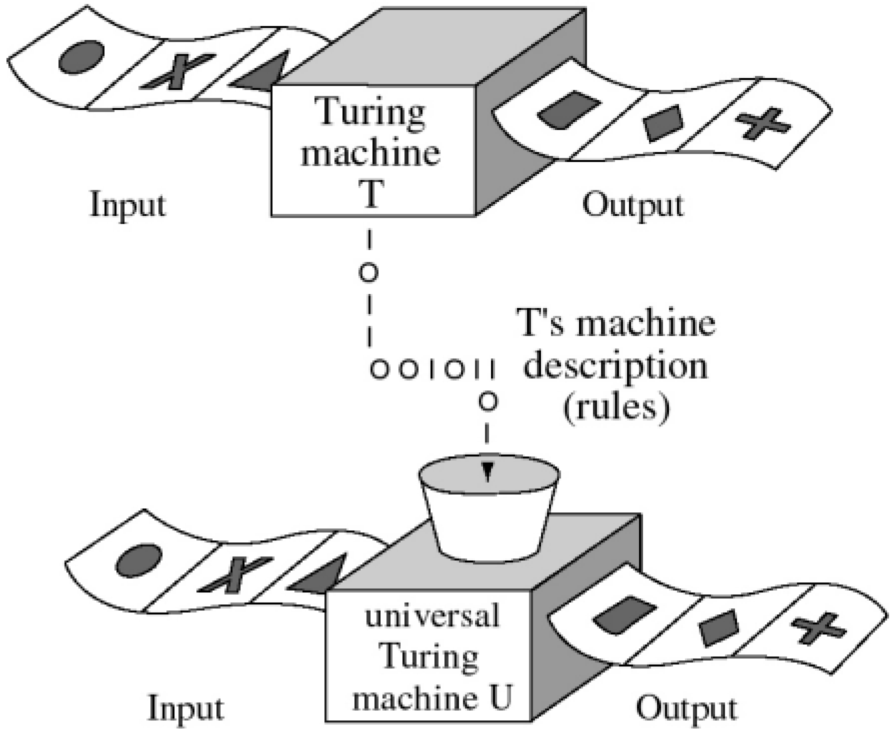
There is a bijection between languages and countable binary sequences that is uncountable by the diagonalization method.

Corollary There exists non Turing-recognizable languages.

Is there another languages between all languages and Turing-recognizable languages?
This is equivalent to the existence of a set number natural numbers and real numbers. So, it is independent of ZFC

The Church–Turing Thesis
Universal Turing Machine

Theorem Universal Turing Machine Exists
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.

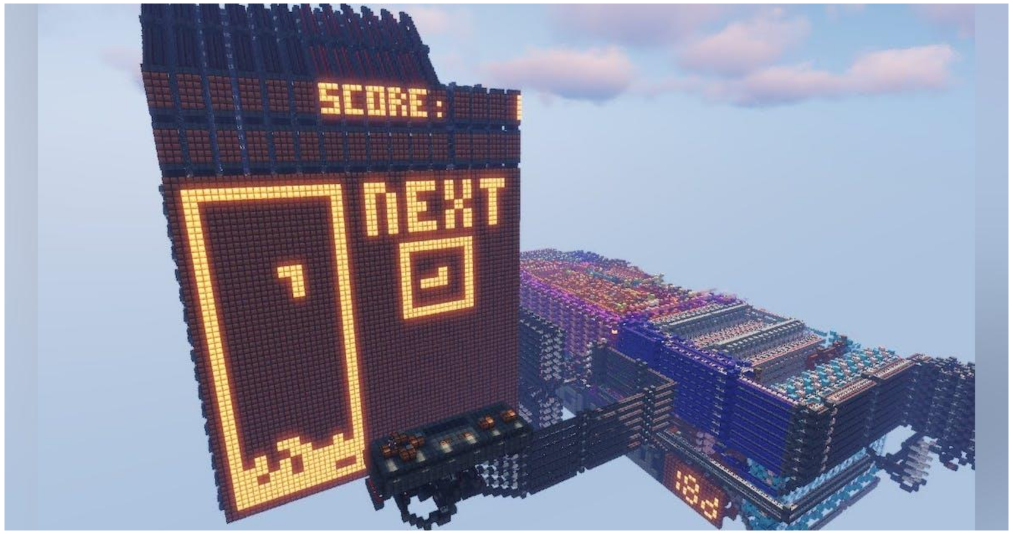


Input to TM U is in the format <M,w> where
M is the TM description, and
w the (converted) input.

The smallest UTM has 2 states and 3 symbols - Alex Smith, a 20-year-old undergraduate from Birmingham 2007

Church-Turing Thesis

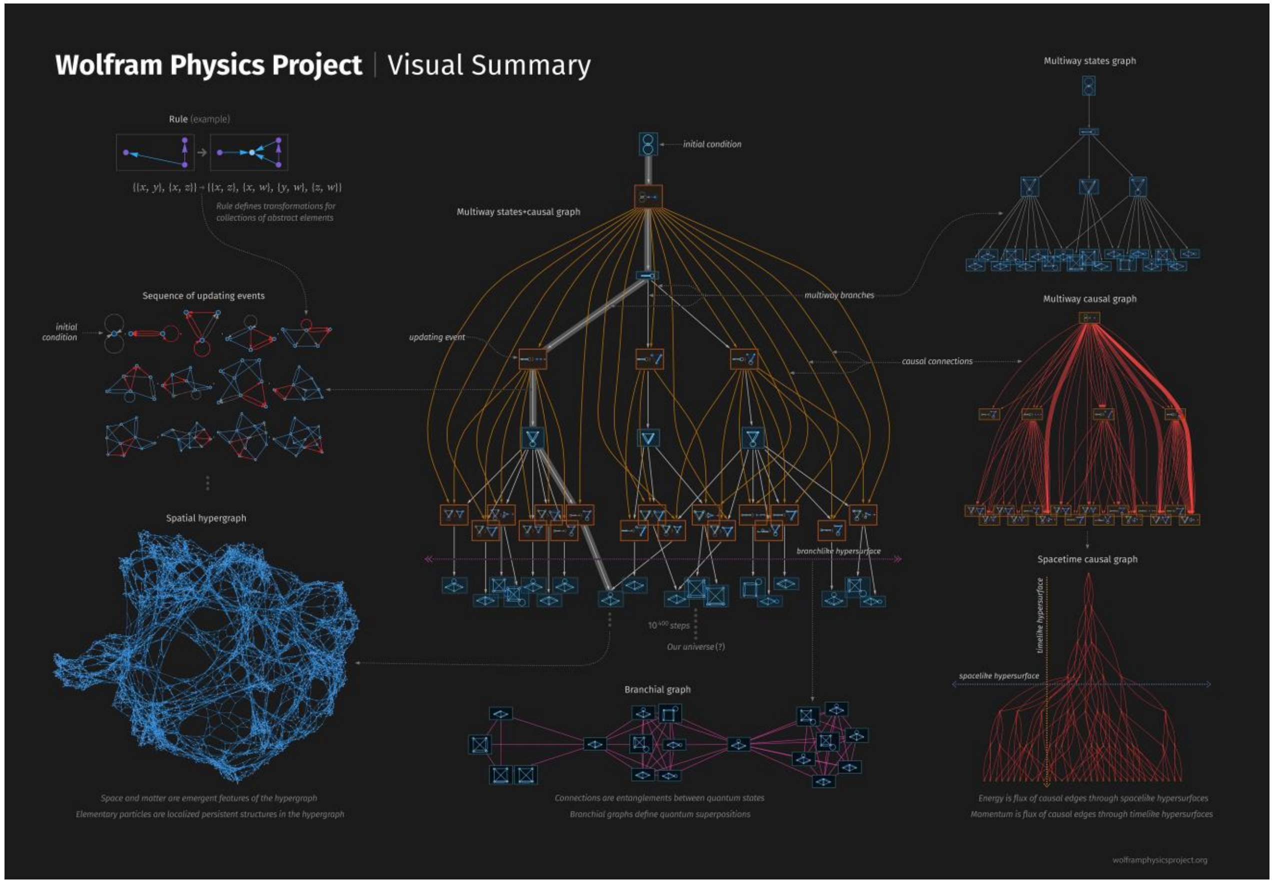
To **prove** the thesis, we need to show that the world is Turing computable.



The Minecraft world is simulated on digital computer, and we can build computer inside Minecraft. Hence, in the Minecraft world, computation is equivalent to Turing computable.

Church-Turing Thesis

To **prove** the thesis, we need to show that the world is Turing computable.



Wolfram Physics Project | Visual Summary
Rule (example)
{{x, y}, {x, z}} → {{x, z}, {x, w}, {y, w}, {z, w}}
Rule defines transformations for collections of abstract elements
Sequence of updating events
initial condition
Spatial hypergraph
Space and matter are emergent features of the hypergraph
Elementary particles are localized persistent structures in the hypergraph
initial condition
Multiway states+causal graph
updating event
multiway branches
causal connections
branchlike hypersurface
10^400 steps
Our universe (?)
Branchial graph
Connections are entanglements between quantum states
Branchial graphs define quantum superpositions
Multiway states graph
Multiway causal graph
Spacetime causal graph
timelike hypersurface
spacelike hypersurface
Energy is flux of causal edges through spacelike hypersurfaces
Momentum is flux of causal edges through timelike hypersurfaces
wolframphysicsproject.org

Stephen Wolfram has a hyper graph replacement based formalism for a theory of everything.

If a theory like this is true, then the world is Turing computable. (how can we know?)

https://writings.stephenwolfram.com/2020/04/finally-we-may-have-a-path-to-the-fundamental-theory-of-physics-and-its-beautiful/

Church-Turing Thesis

To **prove** the thesis, we need to show that the world is Turing equivalent *up to manipulating a sequence of finite symbols.*

Q: but we live in a quantum universe, clearly there are things that cannot be captured by discrete symbols.

A: Turing machine examines a finite sequence of symbols, it cannot represent all mathematical object.
The question is that given your extra power in the physical world, can you do more in terms of recognizing a finite sequence of symbols?

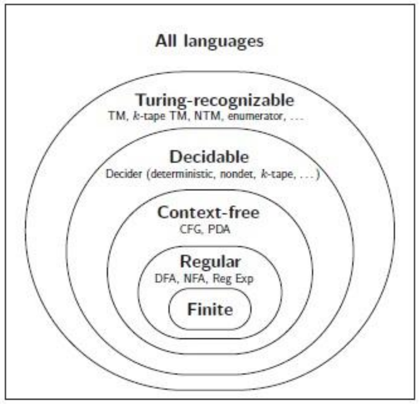
In fact, quantum Turing machine is equivalent to Turing machine.

Also, Church-Turing thesis is not about time complexity.

Church-Turing Thesis

To **disprove** the thesis, we need to show that there is a non Turing-recognizable/decidable language that can be recognized or decided by a physical device.

This process is how we find PDA on top of DFA, and TM on top of PDA.



Can we draw another circle? With possible overlap

If we can find a machine that manipulate the tape in a way that TM cannot simulate in finite time, we can construct a non Turing-recognizable language.

Church-Turing Thesis

Turing’s original argument where Turing shows that human computation can be reduced to a finite set of simple mechanical operations by establishing key limitations:

- Finite symbols: Humans can only write/recognize a limited set of distinct symbols
- Limited observation: We can only view a bounded number of squares at once
- Local movement: We can only move our attention within a fixed distance
- Direct manipulation: We must observe a square to modify it
- Deterministic behavior: Actions are determined by current observations and mental state
- Finite mental states: The number of possible mental states is bounded
- Elementary operations: All computation reduces to simple atomic actions (changing mental state, moving attention, or modifying one symbol)

Decidability

Given a language L whose elements are pairs of the form (B, w), where

- B is some computation model (e, g. DFA, NFA…).
- w is a string over the alphabet Σ.

The pair $(B, w) \in L$ if and only if $w \in L(B)$.

Since the input to computation model B is a string over Σ, we must encode the pair (B, w) as a string.

Acceptance problem for computation model

Decision problem: Dose a given model accept/generate a given string w?

Instance $\langle B, w\rangle$ is the encoding of the pair (B, w).

Universe Ω comprises every possible instance:

$$\Omega = \{\langle B, w\rangle \mid B \text{ is a model and } w \text{ is a string}\}$$

Language comprises all “yes” instances

$$L = \{\langle B, w\rangle \mid B \text{ is a model that accept } w\} \subseteq \Omega$$

Acceptance problem for Language L_{DFA}

Decision problem: Dose a given DFA B accept a given string w?

Instance $\langle B, w\rangle$ is the encoding of the pair (B, w).

Universe Ω comprises every possible instance:

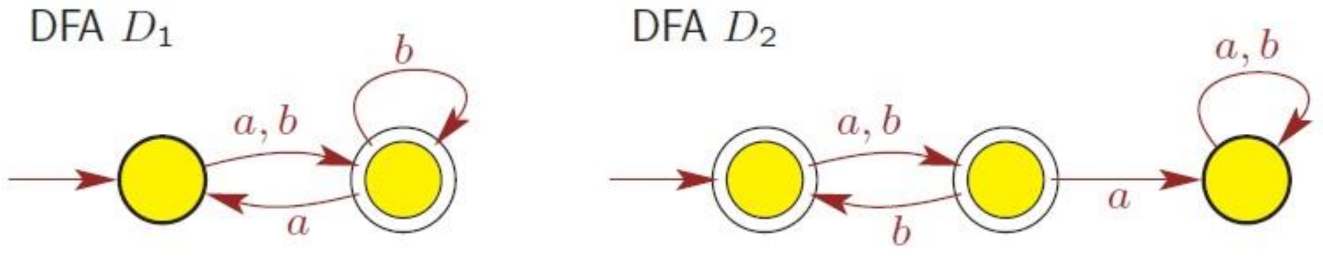
$$\Omega = \{\langle B, w\rangle \mid B \text{ is a DFA and } w \text{ is a string}\}$$

Language comprises all “yes” instances

$$L = \{\langle B, w\rangle \mid B \text{ is a DFA that accept } w\} \subseteq \Omega$$

Acceptance problem for Language L_{DFA}

Example



$\langle D_1, abb \rangle \in A_{\text{DFA}}$ and $\langle D_2, \varepsilon \rangle \in A_{\text{DFA}}$ are YES instances.

$\langle D_1, \varepsilon \rangle \notin A_{\text{DFA}}$ and $\langle D_2, aab \rangle \notin A_{\text{DFA}}$ are NO instances.

The Language L_{DFA} is decidable

$$L_{DFA} = \{\langle B, w\rangle \mid B \text{ is a DFA that accept } w\} \subseteq \Omega$$

$$\Omega = \{\langle B, w\rangle \mid B \text{ is a DFA and } w \text{ is a string}\}$$

To prove L_{DFA} is decidable, we need to construct TM M that decides L_{DFA}.

For M that decides L_{DFA} :

- take $\langle B, w\rangle \in \Omega$ as input
- halt and **accept** if $\langle B, w\rangle \in L_{DFA}$
- halt and **reject** if $\langle B, w\rangle \notin L_{DFA}$

The Language L_{DFA} is decidable

Proof

Basic idea:

On input $\langle B, w\rangle \in \Omega$, where

- $B = (\Sigma, Q, \delta, q_0, F)$ is a DFA
- $w = w_1 w_2 \cdots w_n \in \Sigma^*$ is input string to process on B.

1. Check if $\langle B, w\rangle$ is "proper" encoding. If not, reject
2. Simulate B on w based on:

- $q \in Q$, the current state of B
- $i \in \{1, 2, \ldots, |w|\}$, the pointer that illustrates the current position in w.
- q changes in accordance with w_i and the transition function $\delta(q, w_i)$.

3. If B ends in $q \in F$, then M accepts; otherwise, reject.

The Language L_{NFA} is decidable

Decision problem: Dose a given NFA B accept a given string w?

$$L_{DFA} = \{\langle B, w\rangle \mid B \text{ is a NFA that accept } w\} \subseteq \Omega$$

$$\Omega = \{\langle B, w\rangle \mid B \text{ is a NFA and } w \text{ is a string}\}$$

Proof

On input $\langle B, w\rangle \in \Omega$, where

- $B = (\Sigma, Q, \delta, q_0, F)$ is a NFA
- $w \in \Sigma^*$ is input string to process on B.

1. Check if $\langle B, w\rangle$ is "proper" encoding. If not, reject.
2. Transform NFA B into an equivalent DFA C.
3. Run TM for L_{DFA} on input $\langle C, w\rangle$
4. If M accepts $\langle C, w\rangle$, accept; otherwise, reject.

L_{CFG} are decidable

Decision problem: Dose a CFG G generate a string w ?

$$L_{CFG} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\} \subseteq \Omega$$

$$\Omega = \{\langle G, w\rangle \mid G \text{ is a CFG and } w \text{ is a string}\}$$

$\langle G, w\rangle \in L_{CFG}$ if G generates w, $w \in L(G)$

$\langle G, w\rangle \notin L_{CFG}$ if G dosen't generate w, $w \notin L(G)$

CFGs are decidable

Recall

A context-free grammar $G = (V, \Sigma, R, S)$ is in **Chomsky normal form** if each rule is of the form

$$A \rightarrow BC \text{ or } A \rightarrow x \text{ or } S \rightarrow \varepsilon$$

- variable $A \in V$
- variables $B,C \in V - \{S\}$
- terminal $x \in \Sigma$.

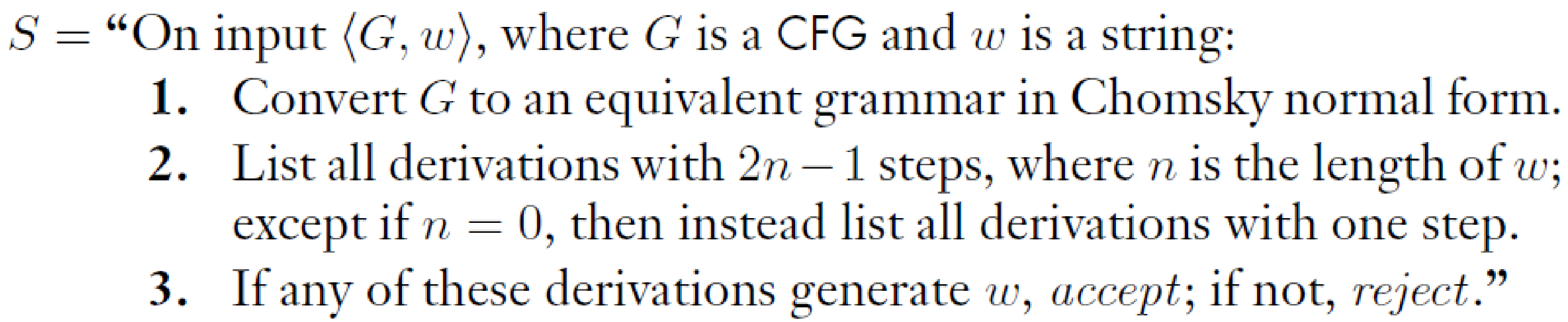
Every CFG can be converted into Chomsky normal form

CFG G in Chomsky normal form is easier to analyze.

- for any string $w \in L(G)$ with $w \neq \varepsilon$ by derivation $S \overset{*}{\Rightarrow} w$ takes exactly $2|w| - 1$ steps. Base case $S \rightarrow w$ where w is singular letter takes 1 step. Additional one step to increase number of variables and another to realize it.
- $\varepsilon \in L(G)$ if G includes rule $S \rightarrow \varepsilon$.

CFGs are decidable

Proof



$S =$ "On input $\langle G, w \rangle$, where G is a CFG and w is a string:

1. Convert G to an equivalent grammar in Chomsky normal form.
2. List all derivations with $2n-1$ steps, where n is the length of w; except if $n = 0$, then instead list all derivations with one step.
3. If any of these derivations generate w, *accept*; if not, *reject*."

Or just run CYK algorithm to find all derivations

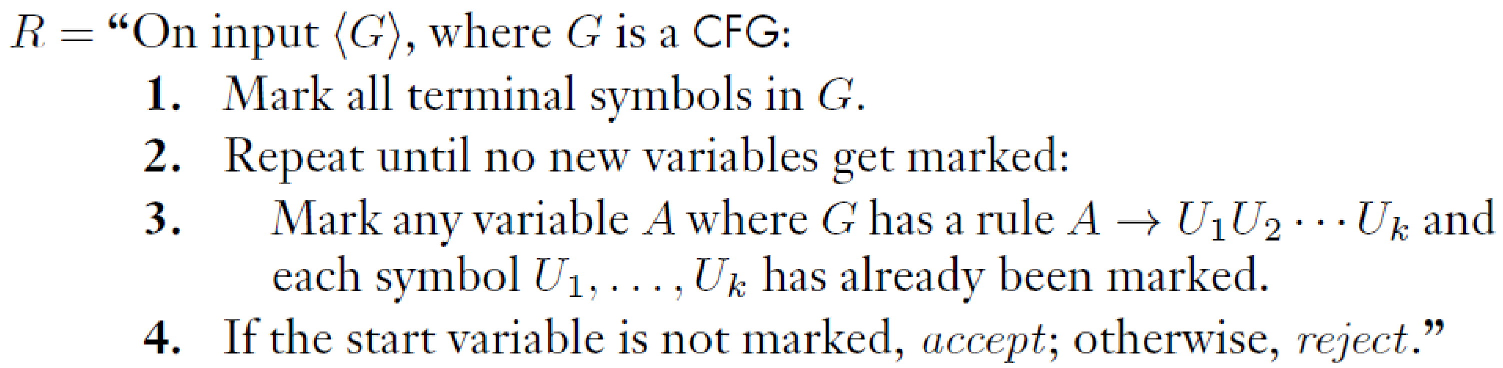
Emptiness of CFLs are decidable

E_{CFG} = {⟨G⟩ | G is a CFG and L(G)=Ø}

Proof idea:

Rule set is finite, and try exhaustively build parse tree from all potential list of terminals, and check whether we can reach the start symbol.

Proof



```
R = "On input ⟨G⟩, where G is a CFG:
    1. Mark all terminal symbols in G.
    2. Repeat until no new variables get marked:
    3.     Mark any variable A where G has a rule $A \to U_1U_2 \cdots U_k$ and
           each symbol $U_1, \ldots, U_k$ has already been marked.
    4. If the start variable is not marked, accept; otherwise, reject."
```

The Language L_{TM} is Turing-recognizable

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

- If M accepts w, then $\langle M, w\rangle \in L_{TM}$
- If M doesn't accept w (reject or loop), then $\langle M, w\rangle \notin L_{TM}$

The language L_{TM} is Turing-recognizable.

Proof:

- A universal Turing machine U simulates M on w
    - If M accepts w, simulation will halt and accept
    - If M doesn't accept w (reject or loop), TM U either reject or loops.

The Language L_{TM} is undecidable

The language L_{TM} is undecidable.

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

- The problem is that we don't really know whether the universal Turing machine will halt or not. Unlike all the machines we saw earlier, TM might run forever.
- Intuitively, it looks hard to find a decider for this problem.
- However, to show this is indeed undecidable is not trivial.

The Language L_{TM} is undecidable

The language L_{TM} is undecidable.

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

Proof:
We will prove by contradiction and use the diagonalization method. We assume such a decider H exists, then show that the set of Turing machine is uncountable.

$$H(\langle M, w\rangle) = \begin{cases} accept & \text{if } M \text{ accepts } w \\ reject & \text{if } M \text{ does not accept } w. \end{cases}$$

The Language L_{TM} is undecidable

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

Proof:
We will prove by contradiction and use the diagonalization method. We assume such a decider H exists, then derive a contradiction in terms of the countability of Turing machines.

$$H(\langle M, w\rangle) = \begin{cases} \mathit{accept} & \text{if } M \text{ accepts } w \\ \mathit{reject} & \text{if } M \text{ does not accept } w. \end{cases}$$

As the set of Turing machine is known to be countable, let's index them by M_i.
Now, it makes sense to ask the answer to $H(< M_i, < M_j >>)$ that is whether M_i accepts the description of M_j as input.

The Language L_{TM} is undecidable

L_{TM} = {⟨M, w⟩ : M is a Turing machine that accepts the string w}

Proof continue:
As the set of Turing machine is known to be countable, let's index them by M_i.
Now, it makes sense to ask the answer to $H(< M_i, < M_j >>)$ that is whether M_i accepts the description of M_j as input.
We have this table

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$
M_1	*accept*	*reject*	*accept*	*reject*	
M_2	*accept*	*accept*	*accept*	*accept*	$\cdots$
M_3	*reject*	*reject*	*reject*	*reject*	
M_4	*accept*	*accept*	*reject*	*reject*	
$\vdots$			$\vdots$		

Can we construct a Turing machine that is not in this list by the diagonalization method?
So that we will show the set of Turing machines is not countable, and we have a contradiction.

The Language L_{TM} is undecidable

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

	$\langle M_1\rangle$	$\langle M_2\rangle$	$\langle M_3\rangle$	$\langle M_4\rangle$	$\cdots$
M_1	*accept*	*reject*	*accept*	*reject*	
M_2	*accept*	*accept*	*accept*	*accept*	$\cdots$
M_3	*reject*	*reject*	*reject*	*reject*	
M_4	*accept*	*accept*	*reject*	*reject*	
$\vdots$			$\vdots$		

Proof continue:

We know that this process will halt, as H is a decider.

Now, we construct a Turing machine D that flips the diagonal (**saying flipping is not enough, as the flipping needs to be done by a Turing machine**):

```
$D =$ "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*."
```

Clearly, D is a decider and hence a Turing machine. It should be in the list.

(Importantly, if H is not a decider, step 1 could loop forever, and D loop forever. Therefore, D does not really flip the diagonal.)

The Language L_{TM} is undecidable

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

$$
\begin{array}{c|ccccc}
 & \langle M_1\rangle & \langle M_2\rangle & \langle M_3\rangle & \langle M_4\rangle & \cdots \\
\hline
M_1 & accept & reject & accept & reject & \\
M_2 & accept & accept & accept & accept & \cdots \\
M_3 & reject & reject & reject & reject & \\
M_4 & accept & accept & reject & reject & \\
\vdots & & & \vdots & &
\end{array}
$$

Proof continue:

$D =$ "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*."

Clearly, D is a decider and hence a Turing machine. It should be in the list. However, $\forall i\ D(< M_i >) = !\,H(M_i, < M_i >) = !\,M_i(< M_i >)$. The first equality comes from the definition of D, and the second from the definition of H.

Now, we have $\forall i, D! = M_i$ as they differ in the input $< \mathrm{M_i} >$. Therefore, D is not in the list. This is a contradiction to the fact TMs are countable.

The Language L_{TM} is undecidable

L_{TM} = {⟨M, w⟩ : M is a Turing machine that accepts the string w}

$$\begin{array}{c|ccccc}
 & \langle M_1\rangle & \langle M_2\rangle & \langle M_3\rangle & \langle M_4\rangle & \cdots \\
\hline
M_1 & accept & reject & accept & reject & \\
M_2 & accept & accept & accept & accept & \cdots \\
M_3 & reject & reject & reject & reject & \\
M_4 & accept & accept & reject & reject & \\
\vdots & & & \vdots & &
\end{array}$$

Alternative Proof:

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*."

Another common presentation of this proof is to ask directly:
Should D accepts <D>?
If D accepts <D>, in step 1 H accepts <D,<D>>, then in step 2 D rejects the <D>.
If D rejects <D>, in step 1 H rejects <D,<D>>, then in step 2 D accepts the <D>.
We have a contradiction.

This proof is similar to Barber paradox

The Language L_{TM} is undecidable

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

$$
\begin{array}{c|ccccc}
 & \langle M_1\rangle & \langle M_2\rangle & \langle M_3\rangle & \langle M_4\rangle & \cdots \\
\hline
M_1 & accept & reject & accept & reject & \\
M_2 & accept & accept & accept & accept & \cdots \\
M_3 & reject & reject & reject & reject & \\
M_4 & accept & accept & reject & reject & \\
\vdots & & & \vdots & &
\end{array}
$$

```
$D$ = "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*."
```

Comments:

The Barber paradox style proof pulls D out of air.

The diagonalization method requires that we find a TM opposite to H on the diagonal, and D is the realization.

Instance of non Turing-recognizable languages.

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

Proof: the only if part is simple, a decider always halts, and the decider accepts the language. For the complement of the language, a Turing machine accepts when the decider rejects and vice versa.

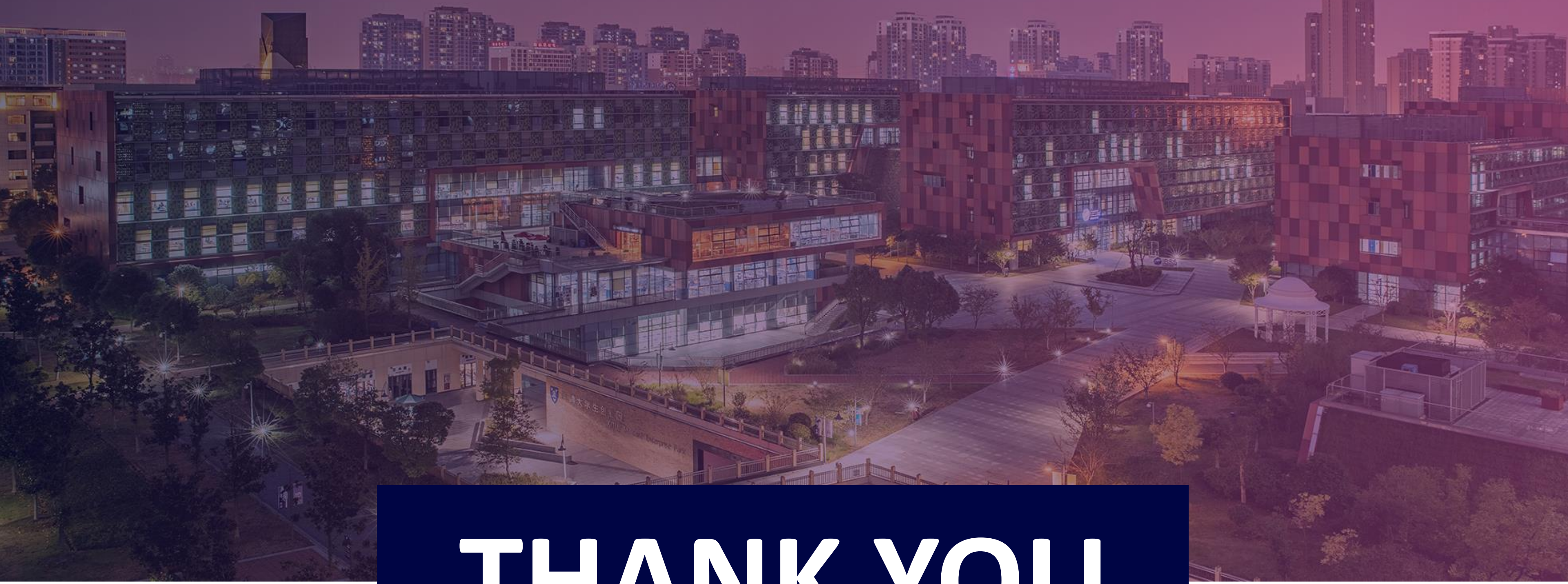
Now, for the if part, if both A and $\bar{A}$ are Turing-recognizable, we let M_1 be the recognizer for A and M_2 be the recognizer for $\bar{A}$. The following Turing machine M is a decider for A.

```
$M$ = "On input $w$:
    1. Run both $M_1$ and $M_2$ on input $w$ in parallel.
    2. If $M_1$ accepts, *accept*; if $M_2$ accepts, *reject*."
```

Corollary $\overline{\mathrm{L}_{\mathrm{TM}}}$ is not Turing-recognizable.

Quick review

- Cantor’s Diagonalization Method
- Church-Turing Thesis and Universal Turing Machine
- Examples of decidable languages
- Existence and instance of undecidable language and non-Turing recognizable language



THANK YOU

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

XJTLU | SCHOOL OF FILM AND TV ARTS