BU CS 332 – Theory of Computation

https://forms.gle/bAZkPdxJAgoinYfm9

Lecture 12:

- Church-Turing Thesis
- Decidable Languages
- Universal TM

Reading:

Sipser Ch 3.3, 4.1

Mark Bun

March 6, 2024

Last Time: Nondeterministic TMs

An NTM N accepts input w if when run on w it accepts on at least one computational branch

$$L(N) = \{w \mid N \text{ accepts input } w\}$$

$w \in L(N) \Rightarrow$ there exists a branch of N's computation leading it to accept input w

$w \notin L(N) \Rightarrow$ all branches of N's computation lead it to reject, run forever, or fail to reach any state on input w

An NTM N is a decider if on **every** input, it halts on **every** computational branch

$w \in L(N) \Rightarrow$ there exists a branch of N's computation leading it to accept input w

$w \notin L(N) \Rightarrow$ all branches of N's computation lead it to reject input w

Nondeterministic TMs

Theorem: Every nondeterministic TM can be simulated by an equivalent deterministic TM

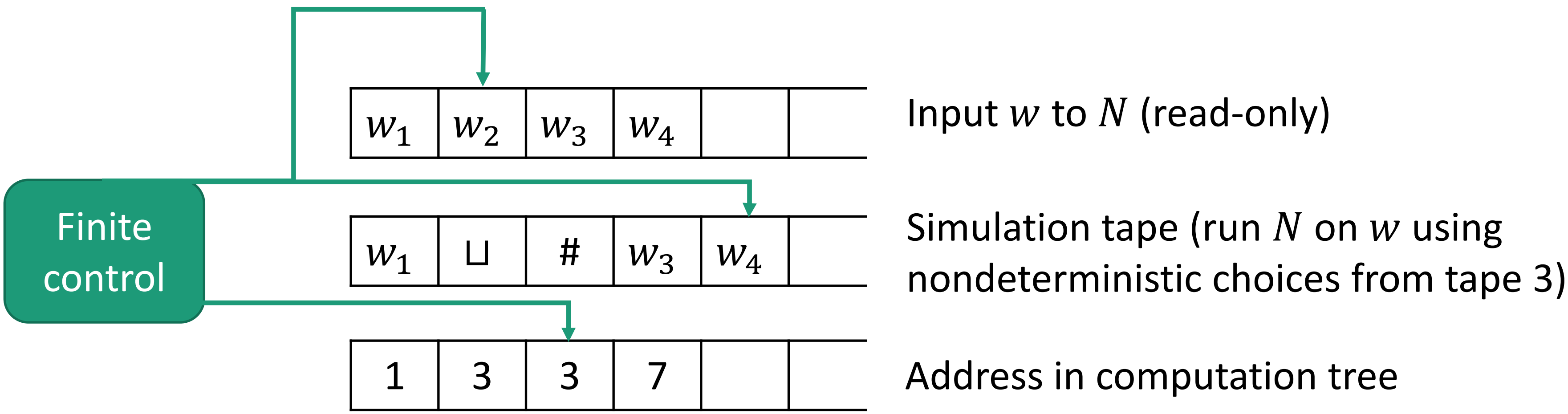
Proof idea: Explore “tree of possible computations”

Nondeterministic TMs

Theorem: Every nondeterministic TM has an equivalent deterministic TM

Proof idea: Simulate an NTM N using a 3-tape TM

(See Sipser for full description)



TMs are equivalent to…

- TMs with “stay put”
- TMs with 2-way infinite tapes
- Multi-tape TMs
- Nondeterministic TMs
- Random access TMs
- Enumerators
- Finite automata with access to an unbounded queue
- Primitive recursive functions
- Cellular automata

…

Church-Turing Thesis

The equivalence of these models is a mathematical theorem (you can prove that each can simulate another)

Church-Turing Thesis v1: The basic TM (hence all of these models) captures our intuitive notion of algorithms

Church-Turing Thesis v2: Any physically realizable model of computation can be simulated by the basic TM

The Church-Turing Thesis is not a mathematical statement! Can't be mathematically proved

Decidable Languages

1928 – The *Entscheidungsproblem*

The "Decision Problem"

Is there an algorithm which takes as input a formula (in first-order logic) and decides whether it's logically valid?

Questions about regular languages

- Given a DFA D and a string w, does D accept input w?
- Given a DFA D, does D recognize the empty language?
- Given DFAs D_1, D_2, do they recognize the same language?

(Same questions apply to NFAs, regexes)

Goal: Formulate each of these questions as a language, and decide them using Turing machines

Questions about regular languages

Design a TM which takes as input a DFA D and a string w, and determines whether D accepts w

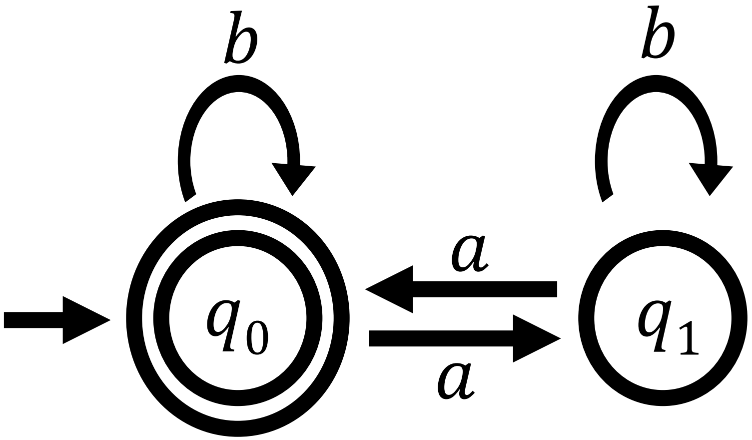
How should the input to this TM be represented?

Let $D = (Q, \Sigma, \delta, q_0, F)$. List each component of the tuple separated by #

- Represent Q by ,-separated binary strings
- Represent Σ by ,-separated binary strings
- Represent $\delta : Q \times \Sigma \to Q$ by a ,-separated list of triples (p, a, q), …

Denote the encoding of D, w by $\langle D, w \rangle$

Example



Representation independence

Computability (i.e., decidability and recognizability) is **not** affected by the precise choice of encoding

Why? A TM can always convert between different (reasonable) encodings

From now on, we'll take ⟨ ⟩ to mean "some reasonable encoding"

A "universal" algorithm for recognizing regular languages

$A_{\text{DFA}} = \{\langle D, w \rangle \mid \text{DFA } D \text{ accepts } w\}$

Theorem: A_{DFA} is decidable

Proof: Define a (high-level) 3-tape TM M on input $\langle D, w \rangle$:

1. Check if $\langle D, w \rangle$ is a valid encoding (reject if not)
2. Simulate D on w, i.e.,
   - Tape 2: Maintain w and head location of D
   - Tape 3: Maintain state of D, update according to δ
3. Accept if D ends in an accept state, reject otherwise

Other decidable languages

$A_{\text{DFA}} = \{\langle D, w \rangle \mid \text{DFA } D \text{ accepts } w\}$

$A_{\text{NFA}} = \{\langle N, w \rangle \mid \text{NFA } N \text{ accepts } w\}$

$A_{\text{REX}} = \{\langle R, w \rangle \mid \text{regular expression } R \text{ generates } w\}$

NFA Acceptance

Which of the following describes a **decider** for $A_{\text{NFA}} = \{\langle N, w\rangle \mid \text{NFA } N \text{ accepts } w\}$?

a) Using a deterministic TM, simulate N on w, always making the first nondeterministic choice at each step. Accept if it accepts, and reject otherwise.

b) Using a deterministic TM, simulate all possible choices of N on w for 1 step of computation, 2 steps of computation, etc. Accept whenever some simulation accepts.

c) Use the subset construction to convert N to an equivalent DFA M. Simulate M on w, accept if it accepts, and reject otherwise.

Regular Languages are Decidable

Theorem: Every regular language L is decidable

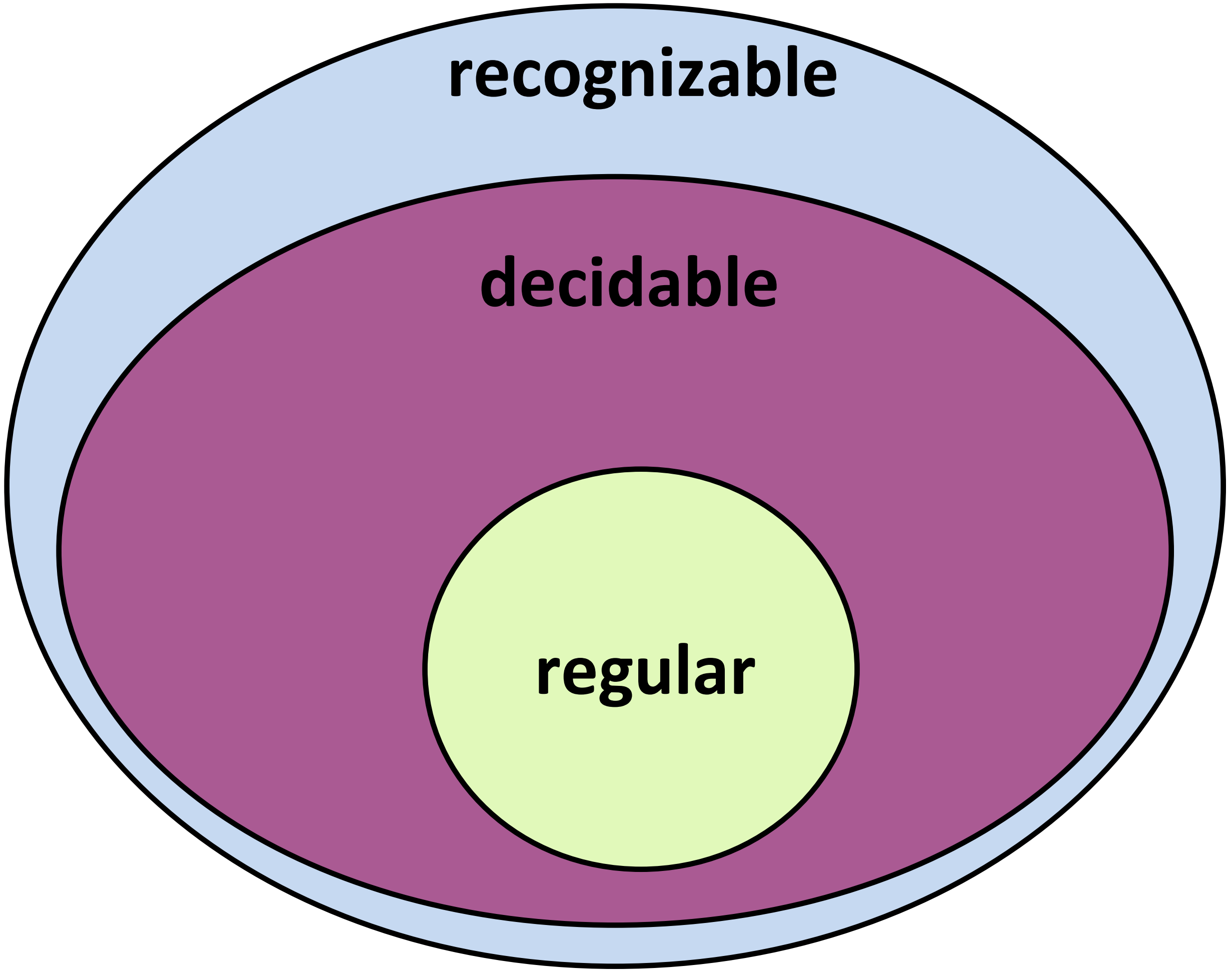
Proof 1: If L is regular, it is recognized by a DFA D. Convert this DFA to a TM M. Then M decides L.

Proof 2: If L is regular, it is recognized by a DFA D. The following TM M_D decides L.

On input w:

1. Run the decider for A_{DFA} on input $\langle D, w \rangle$
2. Accept if the decider accepts; reject otherwise

Classes of Languages



More Decidable Languages: Emptiness Testing

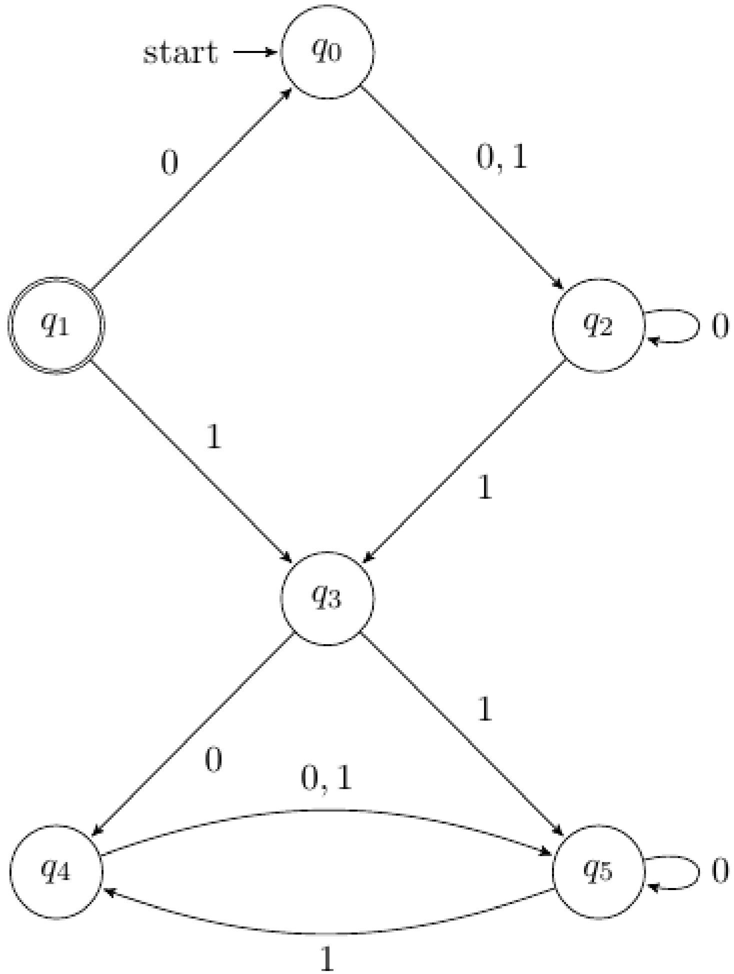
Theorem: $E_{\text{DFA}} = \{\langle D \rangle \mid D \text{ is a DFA such that } L(D) = \emptyset\}$ is decidable

Proof: The following TM decides E_{DFA}

On input $\langle D \rangle$, where D is a DFA with k states:

1. Perform k steps of breadth-first search on state diagram of D to determine if an accept state is reachable from the start state
2. Reject if a DFA accept state is reachable; accept otherwise

E_{DFA} Example



New Deciders from Old: Equality Testing

$EQ_{\text{DFA}} = \{\langle D_1, D_2 \rangle \mid D_1, D_2 \text{ are DFAs and } L(D_1) = L(D_2)\}$

Theorem: EQ_{DFA} is decidable

Proof: The following TM decides EQ_{DFA}

On input $\langle D_1, D_2 \rangle$, where $\langle D_1, D_2 \rangle$ are DFAs:

1. Construct DFA D recognizing the **symmetric difference** $L(D_1) \triangle L(D_2)$
2. Run the decider for E_{DFA} on $\langle D \rangle$ and return its output

Symmetric Difference

$$A \triangle B = \{w \mid w \in A \text{ or } w \in B \text{ but not both}\}$$

Universal Turing Machine

Meta-Computational Languages

$A_{\text{DFA}} = \{\langle D, w\rangle \mid \text{DFA } D \text{ accepts } w\}$

$A_{\text{TM}} = \{\langle M, w\rangle \mid \text{TM } M \text{ accepts } w\}$

$E_{\text{DFA}} = \{\langle D\rangle \mid \text{DFA } D \text{ recognizes the empty language } \emptyset\}$

$E_{\text{TM}} = \{\langle M\rangle \mid \text{TM } M \text{ recognizes the empty language } \emptyset\}$

$EQ_{\text{DFA}} = \{\langle D_1, D_2\rangle \mid D_1 \text{ and } D_2 \text{ are DFAs}, L(D_1) = L(D_2)\}$

$EQ_{\text{TM}} = \{\langle M_1, M_2\rangle \mid M_1 \text{ and } M_2 \text{ are TMs}, L(M_1) = L(M_2)\}$

The Universal Turing Machine

$A_{\text{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM that accepts input } w\}$

Theorem: A_{TM} is Turing-recognizable

The following "Universal TM" U recognizes A_{TM}

On input $\langle M, w\rangle$:

1. Simulate running M on input w
2. If M accepts, accept. If M rejects, reject.

Universal TM and A_{TM}

Why is the Universal TM not a decider for A_{TM}?

The following "Universal TM" U recognizes A_{TM}

On input $\langle M, w\rangle$:

1. Simulate running M on input w
2. If M accepts, accept. If M rejects, reject.

a) It may reject inputs $\langle M, w\rangle$ where M accepts w

b) It may accept inputs $\langle M, w\rangle$ where M rejects w

c) It may loop on inputs $\langle M, w\rangle$ where M loops on w

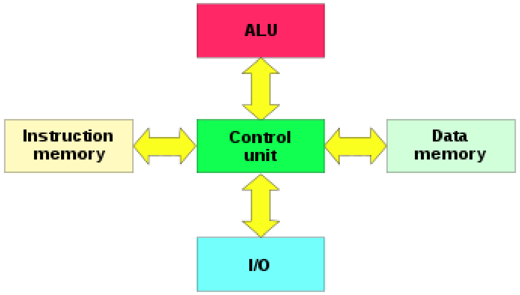
d) It may loop on inputs $\langle M, w\rangle$ where M accepts w

More on the Universal TM

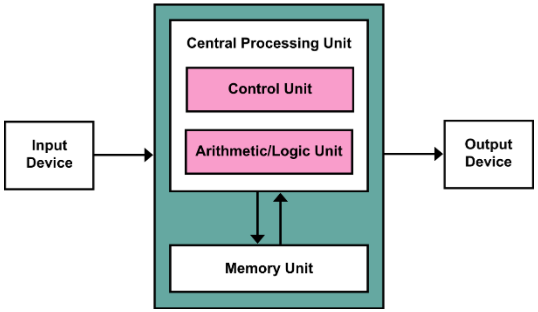
"It is possible to invent a single machine which can be used to compute any computable sequence. If this machine **U** is supplied with a tape on the beginning of which is written the S.D ["standard description"] of some computing machine **M**, then **U** will compute the same sequence as **M**."

- Turing, "On Computable Numbers…" 1936

- Foreshadowed general-purpose programmable computers
- No need for specialized hardware: Virtual machines as software



Harvard architecture:
Separate instruction and data pathways



von Neumann architecture:
Programs can be treated as data

Undecidability

A_{TM} is Turing-recognizable via the Universal TM

…but it turns out A_{TM} (and $E_{\mathrm{TM}}, EQ_{\mathrm{TM}}$) is **undecidable**

i.e., computers cannot solve these problems no matter how much time they are given

How can we prove this?

… after the break