Decidability

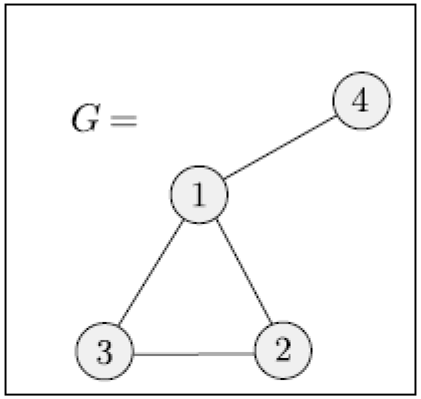
Theory behind existence of undecidable problems – Halting Problem --

Ref: https://www.andrew.cmu.edu/user/ko/pdfs/lecture-15.pdf

Describing Turing Machines and Their Inputs

- For the rest of the course we will have a rather standard way of describing TMs and their inputs.
- The input to TMs have to be strings.
- Every object O that enters a computation will be represented with an string $\langle O\rangle$, encoding the object.
- For example if G is a 4 node undirected graph with 4 edges $\langle O\rangle = \texttt{(1,2,3,4)((1,2),(2,3),(3,1),(1,4))}$
- Then we can define problems over graphs,e.g., as:

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



DESCRIBING TURING MACHINES AND THEIR INPUTS

- A TM for this problem can be given as:
- $M =$ "On input $\langle G \rangle$, the encoding of a graph G:
  1. Select the first node of G and mark it.
  2. Repeat 3) until no new nodes are marked
  3. For each node in G, mark it, if there is edge attaching it to an already marked node.
  4. Scan all the nodes in G. If all are marked, the *accept*, else *reject*"

- Important thing to understand is that a machine (computing machine) can be represented as a string in some language.
  - It will be a string over some alphabet.
  - This is, in some sense, equivalent to say that program is also data.

Representation Vs. Real Thing

- DNA represents a living thing (like a human-being).
- Some people believe (!) that this is a complete description of a human being.
- You can know his personality, you can even know what he will do in future?
  - Some Hollywood movies captured this idea.

- A representation in itself is lifeless.
- A program in itself cannot process the data.
- It has to be realized through a processing unit or DNA has to be realized through a laboratory test tube or so.

- But the processing unit, now can be entirely independent of the program.
- It can execute any program.

- We can give <program, data> to a general purpose computer which runs the program over the data and gives the output.

- A TM M also can be represented as a string.
- Call this string $< M >$.
- $< M >$ is like a program.
- Working of TM M on a string w can be realized by a *general purpose computer* like machine.
- This machine which can take input $< M, w >$ and outputs what the TM M does with w, is called the **U**niversal **T**uring **M**achine.

DECIDABILITY

- We investigate the power of algorithms to solve problems.
- We discuss certain problems that can be solved algorithmically and others that can not be.
- Why discuss unsolvability?
- Knowing a problem is unsolvable is useful because
  - you realize it must be simplified or altered before you find an algorithmic solution.
  - you gain a better perspective on computation and its limitations.

Overview

- Decidable Languages
- Diagonalization
- Halting Problem as a undecidable problem
- Turing-unrecognizable languages.

Decidable Languages

Some notational details

- $\langle B \rangle$ represents the encoding of the description of an automaton (DFA/NFA).
- We need to encode Q, Σ, δ and F.

ENCODING FINITE AUTOMATA AS STRINGS

- Here is one possible encoding scheme:
- Encode Q using unary encoding:
  - For $Q = \{q_0, q_1, \ldots q_{n-1}\}$, encode q_i using $i + 1$ 0's, i.e., using the string 0^{i+1}.
  - We assume that q_0 is always the start state.
- Encode Σ using unary encoding:
  - For $\Sigma = \{a_1, a_2, \ldots a_m\}$, encode a_i using i 0's, i.e., using the string 0^i.
- With these conventions, all we need to encode is δ and F!
- Each entry of δ, e.g., $\delta(q_i, a_j) = q_k$ is encoded as

$$\underbrace{0^{i+1}}_{q_i} 1 \underbrace{0^j}_{a_j} 1 \underbrace{0^{k+1}}_{q_k}$$

Encoding Finite Automata as Strings

- The whole δ can now be encoded as

$$\underbrace{00100001000}_{transition_1}1\underbrace{0000010010000000}_{transition_2}\cdots1\underbrace{0000000100000010}_{transition_t}$$

- F can be encoded just as a list of the encodings of all the final states. For example, if states 2 and 4 are the final states, F could be encoded as

$$\underbrace{000}_{q_2}1\underbrace{00000}_{q_4}$$

- The whole DFA would be encoded by

$$11\underbrace{00100010000100000\cdots0}_{encoding\ of\ the\ transitions}11\underbrace{0000000010000000}_{encoding\ of\ the\ final\ states}11$$

Encoding Finite Automata as Strings

- $\langle B \rangle$ representing the encoding of the description of an automaton (DFA/NFA) would be something like

$$\langle B \rangle = 11\,\underbrace{00100010000100000\cdots0}_{\textit{encoding of the transitions}}\,11\,\underbrace{0000000010000000}_{\textit{encoding of the final states}}\,11$$

- In fact, the description of all DFAs could be described by a regular expression like

$$11(0^+10^+10^+1)^*1(0^+1)^+1$$

- Similarly strings over Σ can be encoded with (the same convention)

$$\langle w \rangle = \underbrace{0000}_{a_4}\,1\,\underbrace{000000}_{a_6}\,1\cdots\underbrace{0}_{a_1}$$

Encoding Finite Automata as Strings

- $\langle B, w \rangle$ represents the encoding of a machine followed by an input string, in the manner above (with a suitable separator between $\langle B \rangle$ and $\langle w \rangle$.
- Now we can describe our problems over languages and automata as problems over strings (representing automata and languages).

Decidable Problems

Regular Languages

- Does B accept w?
- Is $L(B)$ empty?
- Is $L(A) = L(B)$?

The Acceptance Problem for DFAs

Theorem 4.1

$A_{DFA} = \{\langle B, w\rangle \mid B$ is a DFA that accepts input string $w\}$ is a decidable language.

Proof

- Simulate with a two-tape TM.
  - One tape has $\langle B, w\rangle$
  - The other tape is a work tape that keeps track of which state of B the simulation is in.
- $M =$ "On input $\langle B, w\rangle$
  1. Simulate B on input w
  2. If the simulation ends in an accept state of B, *accept*; if it ends in a nonaccepting state, *reject*."

The Acceptance Problem for NFAs

Theorem 4.2

$A_{NFA} = \{\langle B, w\rangle \mid B \text{ is a NFA that accepts input string } w\}$ is a decidable language.

Proof

- Convert NFA to DFA and use Theorem 4.1
- $N =$ "On input $\langle B, w\rangle$ where B is an NFA
  1. Convert NFA B to an equivalent DFA C, using the determinization procedure.
  2. Run TM M in Thm 4.1 on input $\langle C, w\rangle$
  3. If M accepts *accept*; otherwise *reject*."

The Generation Problem for Regular Expressions

Theorem 4.3

$A_{REX} = \{\langle R, w\rangle \mid R$ is a regular exp. that generates string $w\}$ is a decidable language.

Proof

- Note R is already a string!!
- Convert R to an NFA and use Theorem 4.2
- $P =$ "On input $\langle R, w\rangle$ where R is a regular expression
  1. Convert R to an equivalent NFA A, using the Regular Expression-to-NFA procedure
  2. Run TM N in Thm 4.2 on input $\langle A, w\rangle$
  3. If N accepts *accept*; otherwise *reject*."

The Emptiness Problem for DFAs

Theorem 4.4

$E_{DFA} = \{\langle A \rangle \mid A \text{ is a DFA and } L(A) = \Phi\}$ is a decidable language.

Proof

- Use the DFS algorithm to mark the states of DFA
- $T =$ "On input $\langle A \rangle$ where A is a DFA.
  1. Mark the start state of A
  2. Repeat until no new states get marked.
     - Mark any state that has a transition coming into it from any state already marked.
  3. If no final state is marked, *accept*; otherwise *reject*."

- Is the following a decidable language?

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

THE EQUIVALENCE PROBLEM FOR DFAS

THEOREM 4.5

$EQ_{DFA} = \{\langle A, B\rangle \mid A$ and B are DFAs and $L(A) = L(B)\}$ is a decidable language.

PROOF

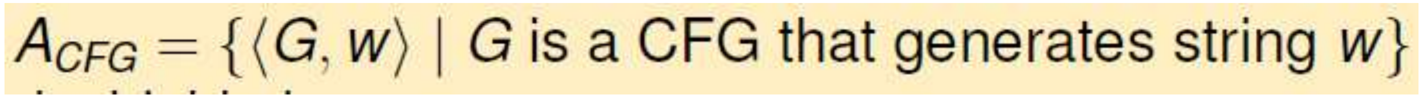
- Construct the machine for $L(C) = (L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$ and check if $L(C) = \Phi$.
- $T =$ "On input $\langle A, B\rangle$ where A and B are DFAs.
  1. Construct the DFA for $L(C)$ as described above.
  2. Run TM T of Theorem 4.4 on input $\langle C\rangle$.
  3. If T accepts, *accept*; otherwise *reject*."

Decidable Problems

Context-free Languages

- Does grammar G generate w?
- Is $L(G)$ empty?

- Is the following language decidable?



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

THE GENERATION PROBLEM FOR CFGS

THEOREM 4.7

$A_{CFG} = \{\langle G, w\rangle \mid G$ is a CFG that generates string $w\}$ is a decidable language.

PROOF

- Convert G to Chomsky Normal Form and use the CYK algorithm.
- $C =$ "On input $\langle G, w\rangle$ where G is a CFG
  1. Convert G to an equivalent grammar in CNF
  2. Run CYK algorithm on w of length n
  3. If $S \in V_{i,n}$ *accept*; otherwise *reject*."

THE GENERATION PROBLEM FOR CFGS

ALTERNATIVE PROOF

- Convert G to Chomsky Normal Form and check all derivations up to a certain length (Why!)
- $S =$ "On input $\langle G, w\rangle$ where G is a CFG
  1. Convert G to an equivalent grammar in CNF
  2. List all derivations with $2n - 1$ steps where n is the length of w. If $n = 0$ list all derivations of length 1.
  3. If any of these strings generated is equal to w, *accept*; otherwise *reject*."

- This works because every derivation using a CFG in CNF either increase the length of the sentential form by 1 (using a rule like $A \rightarrow BC$ or leaves it the same (using a rule like $A \rightarrow a$)
- Obviously this is not very efficient as there may be exponentially many strings of length up to $2n - 1$.

Decidability of CFLs

Theorem 4.9

Every context free language is decidable.

Proof

- Design a TM M_G that has G built into it and use the result of A_{CFG}.
- $M_G =$ “On input w
  1. Run TM S (from Theorem 4.7) on input $\langle G, w \rangle$
  2. If S accepts, *accept*, otherwise *reject*.

The Emptiness Problem for CFGs

Theorem 4.8

$E_{CFG} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \Phi\}$ is a decidable language.

Proof

- Mark variables of G systematically if they can generate terminal strings, and check if S is unmarked.
- $R =$ "On input $\langle G \rangle$ where G is a CFG.
  1. Mark all terminal symbols G
  2. Repeat until no new variable get marked.
     - Mark any variable A such that G has a rule $A \to U_1 U_2 \cdots U_k$ and $U_1, U_2, \ldots U_k$ are already marked.
  3. If start symbol is NOT marked, *accept*; otherwise *reject*."

THE EQUIVALENCE PROBLEM FOR CFGs

$EQ_{CFG} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}$

- Is this decidable?

The Equivalence Problem for CFGs

$EQ_{CFG} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}$

- It turns out that EQ_{DFA} is not a decidable language.
- The construction for DFAs does not work because CFLs are NOT closed under intersection and complementation.
- Proof comes later.

Acceptance Problem for TMs

Theorem 4.11

$A_{TM} = \{\langle M, w\rangle \mid M$ is a TM and M accepts $w\}$ is undecidable.

- Note that A_{TM} is Turing-recognizable. Thus this theorem when proved, shows that recognizers are more powerful than deciders.
- We can encode TMs with strings just like we did for DFA's (How?)

- We shall assume the states are $q_1, q_2, \ldots, q_r$ for some r. The start state will always be q_1, and q_2 will be the only accepting state. Note that, since we may assume the TM halts whenever it enters an accepting state, there is never any need for more than one accepting state.

- We shall assume the tape symbols are $X_1, X_2, \ldots, X_s$ for some s. X_1 always will be the symbol 0, X_2 will be 1, and X_3 will be B, the blank. However, other tape symbols can be assigned to the remaining integers arbitrarily.

- We shall refer to direction L as D_1 and direction R as D_2.

- We shall assume the states are $q_1, q_2, \ldots, q_r$ for some r. The start state will always be q_1, and q_2 will be the only accepting state. Note that, since we may assume the TM halts whenever it enters an accepting state, there is never any need for more than one accepting state.

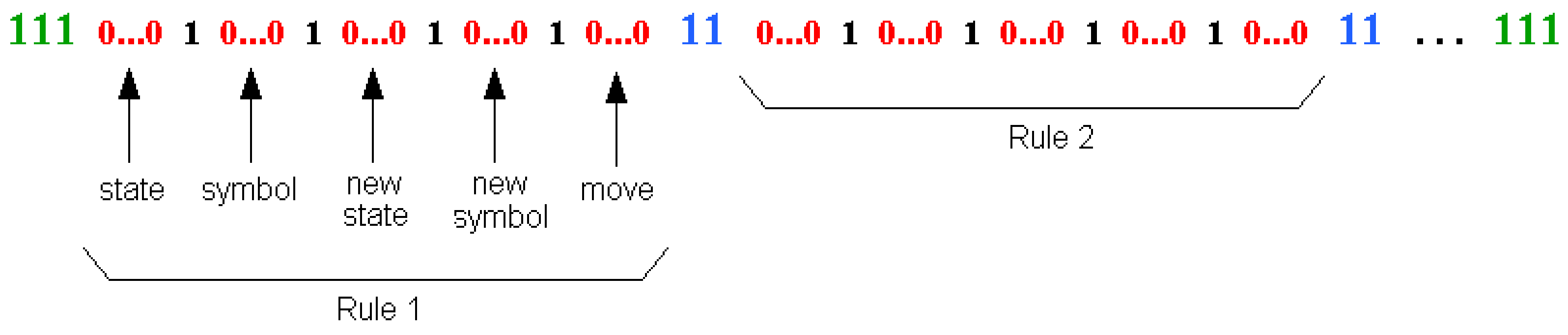
- We shall assume the tape symbols are $X_1, X_2, \ldots, X_s$ for some s. X_1 always will be the symbol 0, X_2 will be 1, and X_3 will be B, the blank. However, other tape symbols can be assigned to the remaining integers arbitrarily.

- We shall refer to direction L as D_1 and direction R as D_2.

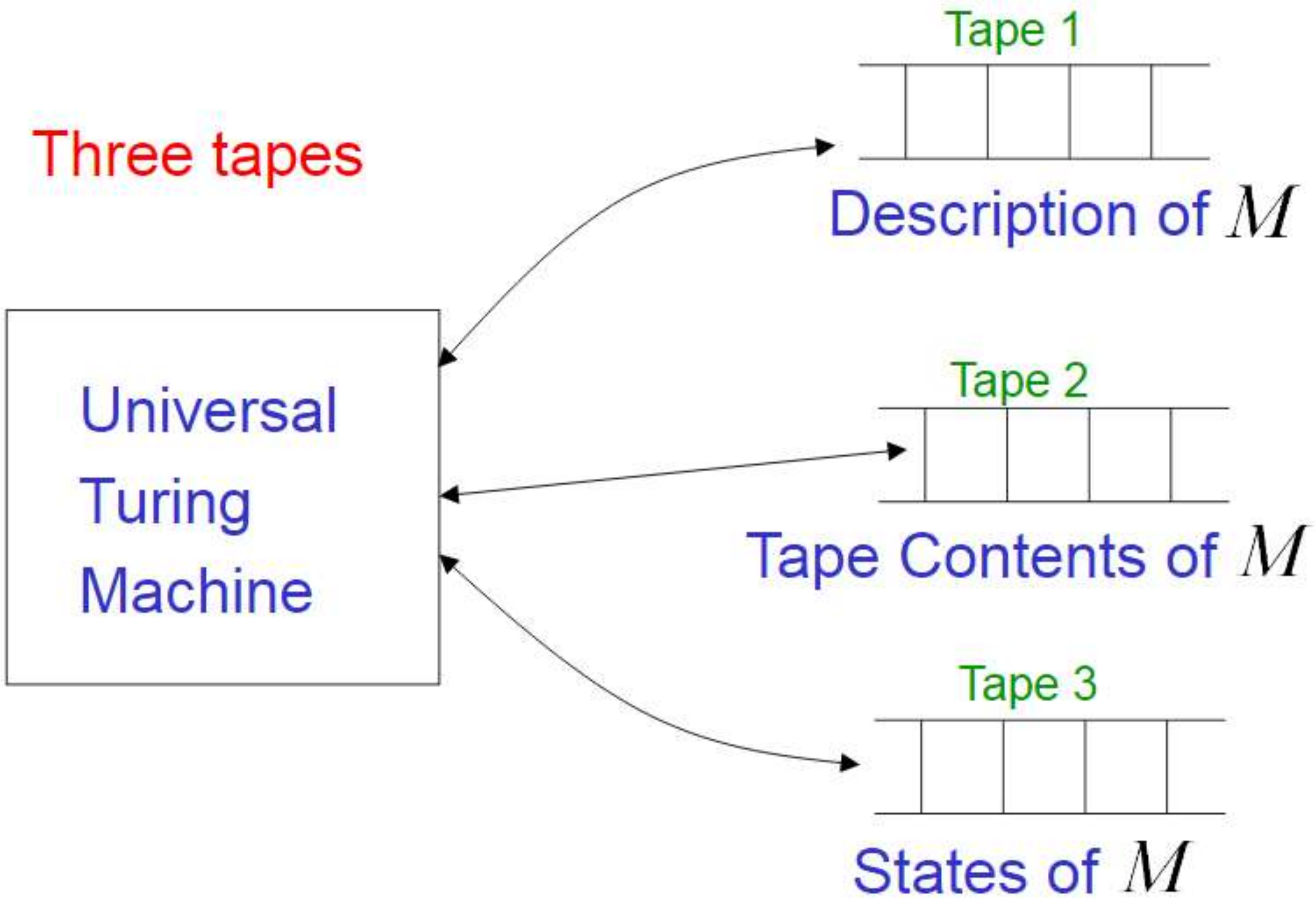
Suppose one transition rule
is $\delta(q_i, X_j) = (q_k, X_l, D_m)$, for some integers i, j, k, l, and m.

We shall code this rule by the string $0^i 1 0^j 1 0^k 1 0^l 1 0^m$.

Encoding of a TM



Three tapes


Universal Turing Machine
Tape 1
Description of M
Tape 2
Tape Contents of M
Tape 3
States of M

- Turing Machine M can be encoded as a string $< M >$
- These encodings, of various Turing Machines, can be seen as a language.
- Language of TMs = $\{ < M > | < M > \text{ is an encoding of a TM } M\}$.
- Strings of the above language can be lexicographically ordered.
- As per the above ordering, one can say the 1st TM, the 2nd TM, so on.
  - Same machine may be encoded in multiple ways
  - 2nd TM may be same as 10032nd TM

Acceptance Problem for TMs

- The TM U recognizes A_{TM}
- $U =$ "On input $\langle M, w\rangle$ where M is a TM and w is a string:
  1. Simulate M on w
  2. If M ever enters its accepts state, *accept*; if M ever enters its reject state, *reject*.
- Note that if M loops on w, then U loops on $\langle M, w\rangle$, which is why it is NOT a decider!
- U can not detect that M halts on w.
- A_{TM} is also known as the Halting Problem
- U is known as the Universal Turing Machine because it can simulate every TM (including itself!)