Lecture 3

Frank

January 25, 2022

1 Nondeterministic TMs (NTM)

1.1 Transition Function

- The transition function of a TM is

$$\delta(q, a) \to (q', a', D)$$

- The transition function of a NTM is

$$\delta(q, a) \to \text{ set of triples of } (q', a', D)$$

NTM has a finite set of choice as to what to do next, in terms of which state to enter, which symbol to change to, and which direction to go to.

1.2 Computation

- A computation in a TM is a sequence of configurations

$$C_0 \vdash C_1 \vdash C_2 \vdash \dots$$

- However, in a NTM, we have a tree as our computation

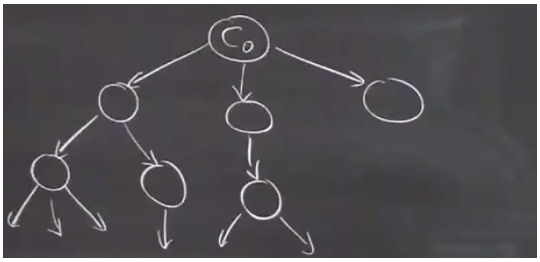


Figure 1: l31

A NTM accepts a string if and only if there is some computational path from the root of our computation tree that ends in an accepting state

1.3 Example

- Consider the Clique problem
  - CLIQUE

    Input: Undirected graph $G = (V, E)$ and $k \in \mathbb{Z}^+$

    Output: yes, if G has a clique of size k. No if otherwise

    Where a clique is a set of nodes where there is a edge between any two nodes in that set

  This problem can be converted into a language by looking at which encoding of the input leads to a yes output. In other words, if the problem is computed by a TM, then the language is the set of all strings that leads to an accepting state for that TM.

- The way that a NTM recognizes this language is as follows:

  Look at all combination of nodes of size k using the NTM's non-determinism, and see if any combination leads to a clique (accept)

- We can look more into this:

  If G has n nodes, represent each node with a string of $\lceil \log_2 n \rceil$ bits

  Set of k nodes represented by a string of $k \cdot \lceil \log_2 n \rceil$ bits

  Then we create a tree with paths to the left representing 0's and paths to the right representing 1's, until we reach $k \cdot \lceil \log_2 n \rceil$ bits, at which point we will have encoded every set of k nodes (total is n choose k $\binom{n}{k}$), then we can deterministically check whether any one of them leads to an accepting state (is a clique)

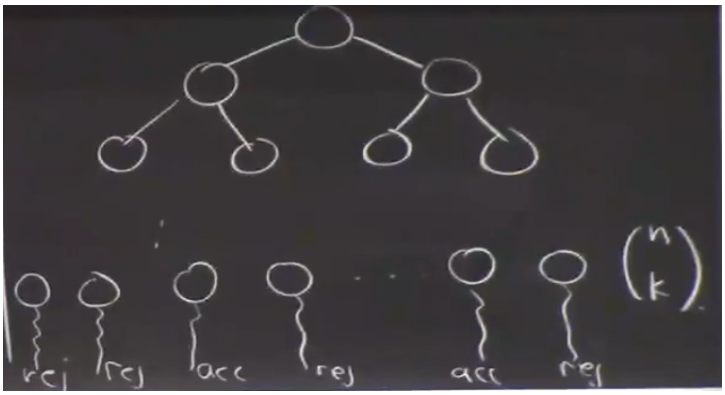


Figure 2: l32

This an example of the way a NTM can solve a problem (the way we will see in this course)

1.4 Equivalence with TM

- **Thm 3.1**: L is recognized by a TM $\iff$ L is recognized by a NTM

  $\implies$ is trivial, because a TM is a special case of NTM

  $\impliedby$ At the high level, use a TM to explore the NTM computation tree in a BFS, until we see a configuration that has an accepting state, in which case, accept.

  Use a MTM (multi-tape TM) to simulate the NTM

  Using 2 tapes, the first tape is the queue of the configurations (of the configurations on each level of BFS). Where the $\star$ indicates the configuration xqy that the TM is scanning and the q inside that configuration is the location of the head of the NTM of that branch, and # is a delimeter

  The TM then takes that configuration and copies all configurations going out from that configuration in the NTM into the work tape, where it is processed into the appropriate configuration according to the δ of the NTM. Then the contents of the work tape is transfer over to the queue (to the next available slot in the queue). We stop until our work tape has generated a configuration with an accepting state.

  Note that we basically recursively generate the next level of the BFS from each configuration at the current level of the BFS.

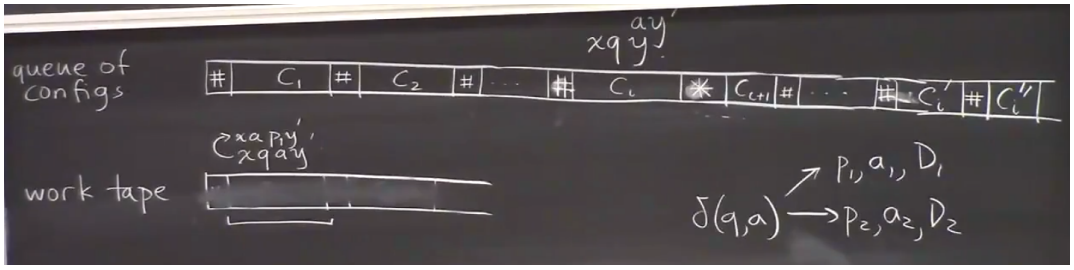


Figure 3: l33

  Since MTM is equivalent to a TM, thus NTM is also equivalent to a TM

- Questions to think about

  Doing a DFS instead of a BFS: No, as seen in textbook, we can miss an accepting path by following a looping path indefinitely.

  BFS the computation tree until we see an accepting state, in which case accept, or until we see a rejecting state, in which case, reject: I don't think so? Because a NTM can have both accepting paths and rejecting paths, so we will be wrong to reject when we follow a rejecting path.

1.5 How many more steps a TM takes to simulate a NTM

- Worst case exponential
- NTM M has accepting computation path in m moves (depth of the path that leads to an accepting configuration), how many moves before simulating TM M' accepts?

  Suppose maximum number of choices of $M = k \leq |Q| \cdot |\Gamma| \cdot 2$ for the number states to go to, the number of symbols to change to, and going left or right

  number of configurations in the tree explored by M' before finding an accepting configuration of M is $1 + k + k^2 + \ldots + k^m = \frac{k^{m+1}-1}{k-1} = O(k^m)$, for first level, second level, and so on...

  Each configuration has length $m + 1 = O(m)$, and the amount of work we do for each configuration is proportional to its length, so total number of moves by $M' = O(m \cdot k^m) = 2^{O(m)}$

2 Church's Thesis

2.1 Perspective on the TM

- We see TMs as language recognizers and deciders
- However, TM can also be seen as computers of (partial) functions $\mathbb{N} \to \mathbb{N}$

  For example, if TM is a computer of f

  $$f : \mathbb{N} \to \mathbb{N}$$

  Then we load input string onto the TM, then after computation, we will see the output string on the tape of the TM.

  On inputs that never halt, the function for that input value is undefined.

2.2 Introduction to CT (Not a theorem)

- Connects informal notions (algorithm, effective procedure) to formal notions (Turing Machine that does the job)

  algorithm, effective procedure $\sim$ TM

  Since TM's steps could be described algorithmically, the reverse direction is not too difficult to see.

  However, we can see that for every algorithm or procedure in say the C language, we can convert that algorithm into a TM (not easy). This is supported by empirical and mathematical evidences (no counter examples yet).

  Mathematical evidences for CT:

  1. extensions to TMs are equivalent to TMs (more tapes, non-determinism)
  2. Competing formalisms (computability) are equivalent (different ways of looking at the same problem), which suggests that this is the solution
     - Partial recursive functions
       (certain functions is what effective procedure means) - Godel 1933
     - λ-calculus (Church, Kleene 1936)
     - Turing Machines (Turing 1936)

- Markov algorithms (1950s)

Seems to suggest we got the concept right (all equivalent)

3. Universality
   - The idea is there exists a TM that takes as input, a TM with its input string, and simulate the input TM. (Self-referencing aspect)

     Can describe a TM that describes TMs (introspective). Other simpler type automatas do not have that ability (FSA, PDA)

This means anything done by a modern machine can be done by a TM.

3 Universal TM

3.1 Encoding TMs

- Universal TM M_u takes as input, description of any TM M and input x, and simulates M on x

  Thus, we need a way to encode an entire TM into a string over a finite alphabet. Note a TM is defined by a set of states, symbols, transition function, etc.

- Lets illustrate one possible encoding of TMs using alphabet $\{0, 1, \#\}$
  - Arbitrary TM

    $$M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$$

  - Encode state q by a binary string $\langle q \rangle$ (angle brackets represent encoding of some mathematical object that has finite description) of size $\lceil \log_2 |Q| \rceil$ bits
  - Encode Q by listing code of its elements separated by $\#$'s

    $$\langle q_0 \rangle \# \langle q_a \rangle \# \langle q_r \rangle \# \langle q_1 \rangle \# \ldots \# \langle q \rangle$$

    Where init state first, accepting state second, and so on...
  - Encode tape symbol $a \in \Gamma$ by a binary string $\langle a \rangle$ of $\lceil \log_2 |\Gamma| \rceil$ bits
  - Encode Σ by listing codes of symbols in Σ separated by $\#$'s

    $$\langle a \rangle \# \langle b \rangle \# \langle c \rangle \# \ldots \# \langle z \rangle$$

  - Encode Γ similarly, with convention that code of blank symbol is listed first.
  - Encode $x = a_1 a_2 \ldots a_k \in \Gamma^\star$ (tape strings) as

    $$\langle a_1 \rangle \langle a_2 \rangle \ldots \langle a_k \rangle = \langle x \rangle$$

  - Encode transition $\delta(q, a) = (p, b, D)$ as

    $$\langle q \rangle \# \langle a \rangle \# \langle p \rangle \# \langle b \rangle \# \langle D \rangle$$

    Where D can be encoded as 0 to go left and 1 to go right.
  - Encode entire transition function δ as

    $$\langle \text{trans 1} \rangle \#\# \langle \text{trans 2} \rangle \#\# \ldots \#\# \langle \text{trans k} \rangle$$

  - And encoding of q_0, q_a, q_r are explicit in the encoding of Q (first second third)

– Encode entire TM M as

$$\langle M \rangle = \langle Q \rangle ### \langle \Sigma \rangle ### \langle \Gamma \rangle ### \langle \delta \rangle \in \{0, 1, \#\}^\star$$

- Strings in $\{0, 1, \#\}$ that don't encode a TM as above, encode TM that rejects all strings
- Note that we can also use only $\{0, 1\}$ by using a uniary encoding (0 repeated however many times to represent a symbol) and 1 as a separator.

  In modern computers, we do not need a separator because the computer only recognizes a string of say 64-bits as words, so we have a automatic separators.

3.2 Universal TM

- M_u takes as input encoding of a TM M and its input x, and simulates M on x

  Input to M_u: $\langle M, x \rangle$

$$\langle M \rangle #### \langle x \rangle$$

- **Thm 3.2**: There is a TM M_u called the universal TM that
  - accepts $\langle M, x \rangle$ if M accepts x
  - rejects $\langle M, x \rangle$ if M rejects x
  - loops on $\langle M, x \rangle$ if M loops on x
- Describe M_u as a 3-tape TM. This can be converted to a TM.

  Tape alphabet of $M_u = \{0, 1, \#, \sqcup\}$
  - The first tape is the input tape.

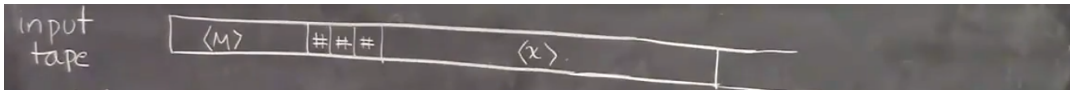


Figure 4: l34

  - The second tape is the encoded M's tape
  - The third tape is the encoded state of M

- The TM M_u does the following

    1. Takes x on the first tape and move it onto the second tape. Then puts on the third tape, the encoding of the initial state

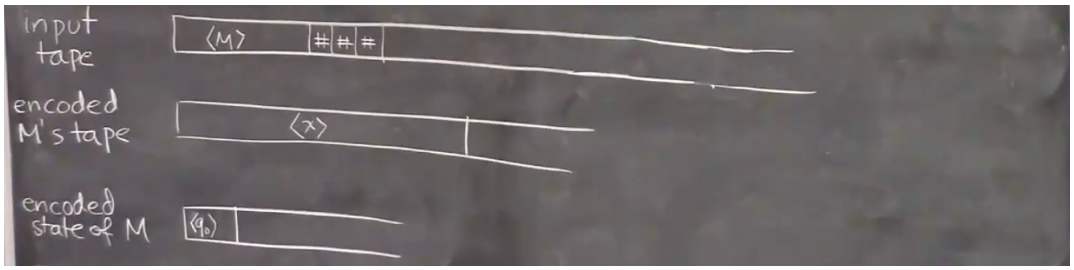


Figure 5: l35

Note that all encodings are currently over $\{0, 1, \#\}$

So say that

$$\langle x \rangle = \langle a \rangle \langle d \rangle \langle z \rangle$$

2. Then M_u looks for the string (matching $\langle q \rangle$ and $\langle a \rangle$ on the first 2 entries)

$$\langle q \rangle \# \langle a \rangle \# \langle p \rangle \# \langle b \rangle \# \langle R \rangle$$

on the first tape (input tape - repository), so we can figure out the next state and which symbol we have written

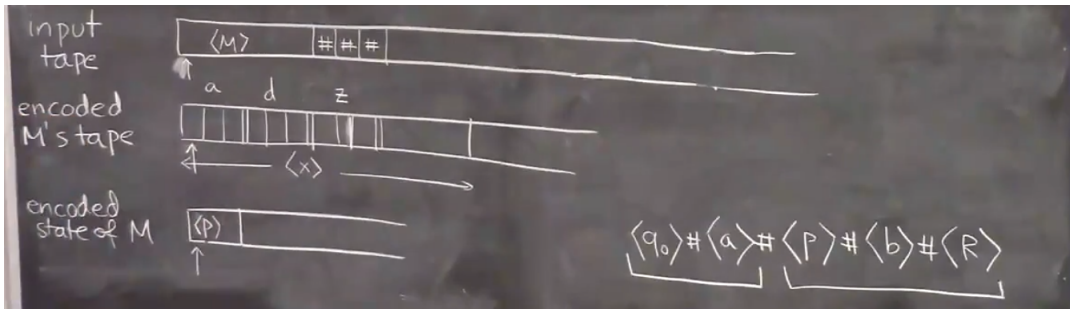


Figure 6: l36

3. Then it changes the state and symbol, then moves the head on the second tape (simulates M's tape) to the next "cell"

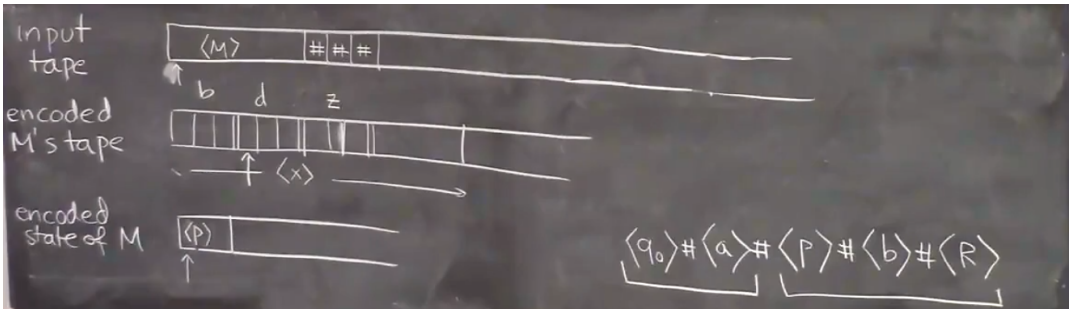


Figure 7: l37

We have simulated one move of M!

4. Then we go back to step 2 and look for the next transition function we need, until we reach an accepting or rejecting state, in which case we accept or reject

- Note that this is actually what a modern computer does!

  The modern computer loads the program and input onto a slice on memory (tape 1).

  Loads input onto another piece of memory (tape 2) and looks for instructions on the first tape as to what to do next.

  tape 3 is sort of like a program counter, keeping track of where the instruction pointer is (if it points to a return statement, then halts, etc. . .)

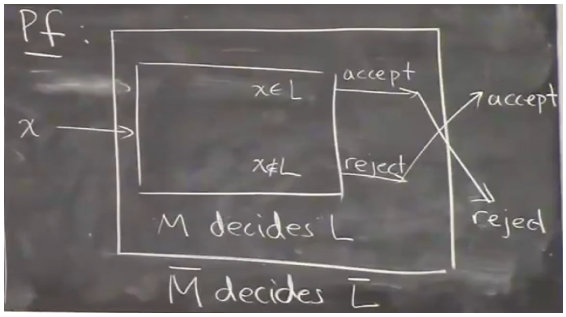
  This is **not** a coincidence, the modern computer is built from the universal TM

4 Undecidable/Unrecognizable Languages

4.1 Thm 3.3

- **Thm 3.3**: If L is a decidable language (if there is a TM that decides the language, on all inputs gives yes/no), then $\bar{L}$ is also decidable (The set of decidable languages is closed under complementation)

  proof:



If M accepts all strings in L, then M rejects all strings in $\bar{L}$, which is accepted by $\bar{M}$, vice versa

- This is not true if we replace "decidable" with "recognizable"
- Fix Σ (input string) to include enough symbols to encode TMs (ex. $\Sigma = \{0, 1, \#\}$)

  Consider languages over Σ

  - $u = \{\langle M, x\rangle : \text{ M accepts x}\}$ (**universal language**)
    Sets of encodings for machine and input such that M accepts x
    or A_{TM} = the set of all strings accepted by TMs
  - $D = \{\langle M\rangle : \text{ M does not accept } \langle M\rangle \}$
    Where D stands for diagonal (trouble)

4.2 Thm 3.4

- **Thm 3.4**: D is not recognizable

  proof:

  - Suppose for contra, that D is recognizable (there is a turing machine M_D that recognizes D).
    Then $\forall$ TM M, M_D accepts $\langle M\rangle \iff M$ does not accept $\langle M\rangle$ [by def of M_D]
  - Take $M = M_D$:
    M_D accepts $\langle M_D\rangle \iff M_D$ does not accept $\langle M_D\rangle$, which is a contradiction

- We can visualize the diagonalization as follows:

  enumerate all TMs and their inputs, 1 if machine accepts string, 0 otherwise

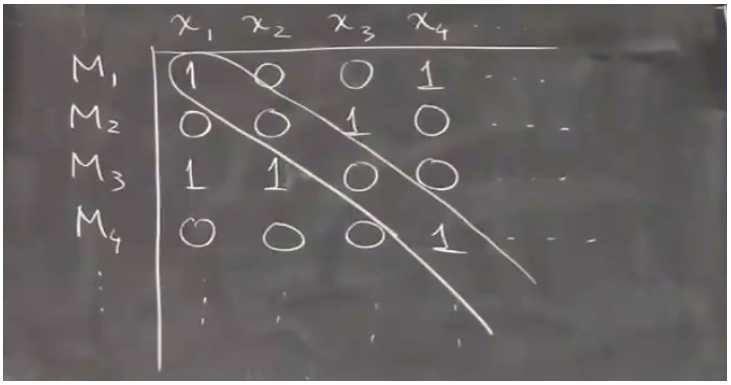


Figure 8: l39

Look at the diagonal - whether machine i accepts string i

- Lets compliment the infinite string on the diagonal (gives language of D)

  Then we get 0110 ...

  Note that none of the TMs in the enumeration has this sequence of 0's and 1's, or the characteristic vector (we have seen a similar diagonalization argument), where the TM rejects the 1st string, accepts second, etc...

So the complement of the diagonal is the characteristic vector of M_D

So M_D cannot exist in our enumeration of TMs

$\implies$ M_D doesn't exist

4.3 Thm 3.5

- **Thm 3.5**: $u = \{\langle M, x\rangle : M \text{ accepts } x\}$ is a) recognizable but b) not decidable

  Proof a): M_u recognizes u, note that M_u accepts $\langle M, x\rangle$ if M accepts x

  Proof b):

  - Suppose for contra, that u is decidable

    Let M_1 be a TM that decides u

    Can use M_1 to decide $\bar{D}$, where $\bar{D} = \{\langle M\rangle : M \text{ accepts } \langle M\rangle\}$

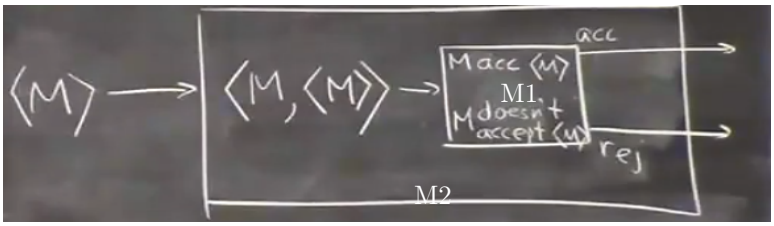


Figure 9: l3x1

  - M_2 = on input $\langle M\rangle$
    1. construct $\langle M, \langle M\rangle\rangle$
    2. Run M_1 on $\langle M, \langle M\rangle\rangle$
    3. If M_1 accepts, accept
    4. Else reject
  - Then M_2 is a $\bar{D}$ decider.
  - By Thm 3.3, if a language is decidable, its complement is also decidable

    $\implies$ $\bar{\bar{D}} = D$ is decidable, which contradicts Thm 3.4, and we have a contradiction
  - So M_2 is not a $\bar{D}$ decider, then we have a contradiction again, so M_1 does not decide u, and u is undecidable

- The proof we have used to prove that u is not decidable is a **proof by reduction**

  P reduces to $Q \iff \exists$ algo to solve P that uses an assumed algo for Q (as a "black box")

- We reduced $\bar{D}$ to u, $\bar{D} \leq u$, and since $\bar{D}$ is not decidable, then u is also not decidable

4.4 Thm 3.6

- **Thm 3.6**: If L and $\bar{L}$ are both recognizable, then L and $\bar{L}$ is decidable

  Proof: suppose $L, \bar{L}$ both recognizable,

  Let M_1 be TM that recognizes L

Let M_2 be TM that recognizes $\bar{L}$

Construct M that decides L

- M simulates M_1 for one step,

  M simulates M_2 for one step,

  and so on, until one or the other accept. If M_1 accepts, then accept, if M_2 accepts, then reject

4.5 Cor 3.7

- $\bar{u} = \{\langle M, x\rangle : M \text{ does not accept } x\}$ is not recognizable

  Proof: If $\bar{u}$ is recognizable, then both u and $\bar{u}$ will be recognizable, which by thm 3.6, u is decidable. But since u is not decidable, then $\bar{u}$ cannot be recognizable

4.6 Cor 3.8

- The set of recognizable languages is <u>not</u> closed under complementation (contrasted to thm 3.3). This follows from cor 3.7.

  Consider the relationship (not drawn to scale)

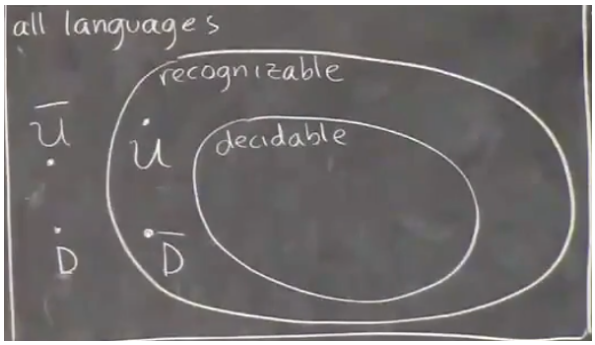


Figure 10: l3x2

Note that the set of all languages is uncountable.

The set of recognizable languages is countable, because each recognizable language can be recognized by a TM. And the set of TMs is countable because we can encode every TM with finite strings

Then the set of decidable languages is also countable