CSC 320 - Lecture 13

#enumerators #turing-machines #TM #church-turing-thesis #yes-no-problems #high-level #halting-problem #decider #decidable-languages

---

Enumerators

They are a variant of Turing Machine. An **enumerator** E is a TM that outputs each string of its language $L(E)$.

Note. An enumerator may output the same string(s) many times (even an infinite amount of times).

A Language Is Turing-Recognizable If And Only If Some Enumerator Outputs It

Proof.

1. Given enumerator E, build TM M that recognizes $L(E)$
2. Given TM M, design enumerator E that outputs $L(M)$

Given Enumerator E, Build TM M That Recognizes $L(E)$

Given enumerator E that enumerates language A, i.e., $L(E) = A$. Build (Multitape) TM M that works as follows... $M =$ "On input w:

- Run E
  - Every time E outputs a string s, compare it with w
    - if $w = s$ then accept".

Given TM M, Design Enumerator E That Outputs $L(M)$

Given TM M recognizing language A. Construct enumerator E for A...

Let $s_1, s_2, s_3, \ldots$ be list of all possible strings in Σ... $E =$ "Ignore input.

- Repeat for $i = 1, 2, 3, \ldots$
  - Run M for i steps on each input s_k from $s_1, s_2, s_3, \ldots, s_i$
    - If s_k is accepted by M then print s_k

Note. We can't allow M to enter an infinite loop.

Corollary

A language L is Turing-Recognizable **IFF** some TM recognizes L **IFF** some Multitape TM recognizes L **IFF** some Nondeterministic TM recognizes L **IFF** some Enumerator outputs L.

The Church Turing Thesis

Algorithms & Turing Machines.

Recall. Definition of algorithm: Finite number of exact instructions (each instruction is of finite length). Produces the desired result in a finite number of steps. (An algorithm needs to stop).

Thesis. Algorithms = Deciders

Thus, a problem can be solved following an algorithm (while ignoring resource limitations) if and only if it is computable by a Turing Machine.

Does There Exists An Algorithm / TM That Decides The Halting Problem?

Halting Problem. Does there exist Algorithm / TM that, when given any code / program with any input, determines if the program halts.

Note. We will discuss when we see undecidable.

Does There Exist A Decider For The Halting Problem?

To figure this out we first study the High-Level descriptions of Turing Machines and Prove that certain problems / languages are decidable using High-Level descriptions of deciders.

Describing Turing Machines

New Focus. Turing Machines (i.e., deciders) as algorithms. TM serves as precise model for definition of algorithm.

Formal Description. Spells out in full TM's states, transition function, ...

Implementation Description. Use English prose to describe was that TM moves head(s) and way that it stores data on its tape(s). At this level no details of states or transition function.

High-Level Description. English prose to describe algorithm, ignoring implementation details. At this level no need to mention how TM manages its tape or head.

High-Level Descriptions of TM's

English prose to describe algorithm. No need to mention how TM manages tape or head. Format and Notation for describing TM.

- Input to TM: String
    - Objects other than string as input must first be represented as string.

Notation for encoding of an object O into its representation as string: $\langle O \rangle$; several objects $O_1, O_2, \ldots, O_k$: $\langle O_1, O_2, \ldots, O_k \rangle$.

Example

Language A consisting of **all strings representing undirected graphs that are connected**.

Remember. A graph is **connected** if every node can be reached from every node by travelling along the edges of the graph.

We Write. $A = \{\langle G \rangle | G \text{ is a connected undirected graph}\}$. Where $\langle G \rangle$ is the encoding of G as string.

Recall. Connection between problems and solutions. A (decision or yes/no) problem is a mapping from a set of problem instances to Yes/No (called yes-instances and no-instances). Languages are abstract representation of problems. For a problem Π, the associated language L_Π is...

$L_\Pi = \{x \in \Sigma^* | x \text{ is a yes-instance of } \Pi\}$

Examples - Yes-No-Problems and Their Languages

Sorted Sequence

- **Input**: A list of n comparable elements $e_1, e_2, \ldots, e_n$.
- **Question**: Are the elements, as given, in sorted order? That is: is it true that $e_1 \leq e_2 \leq \ldots \leq e_n$?

$L_{\text{SORTED SEQUENCE}} = \{\text{list } l \text{ of comparable elements} \mid \text{elements of } l \text{ are in sorted order}\}$.

As language suitable for TM: $L_{\text{SORTED SEQUENCE}} = \{\langle l \rangle | l \text{ list of comparable elements } \& \text{ elements of } l \text{ are in sorted order}\}$. (Encoding of list of comparable elements as string).

Connected Graph

- **Input**: A simple, undirected graph $G = (V, E)$.
- **Question**: Is G connected? That is: for any pairs of vertices $x, y \in V$, does there exists a path from x to y in G?

$L_{\text{CONNECTED GRAPH}} = \{G = (V, E) | G \text{ is a simple, undirected connected graph}\}$.

As language suitable for TM:
$L_{\text{CONNECTED GRAPH}} = \{\langle G \rangle | G \text{ is a simple, undirected connected graph}\}$.

Short Spanning Tree

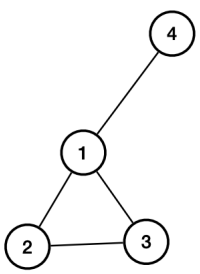
- **Input**: A simple, undirected, edge-weighted graph $G = (V, E)$ where each edge $e \in E$ is assigned a positive integer weight $w(e)$, and integer k.
- **Question**: Does there exist a spanning tree $T = (V, E_T)$ for G where T has weight at most k? That is: T is a tree, $E_T \subseteq E$, and $\Sigma_{e \in E_T} w(e) \leq k$?

$L_{\text{SHORT SPANNING TREE}} = \{(G = (V, E), k) | k \text{ is a positive integer and } G$
$\text{is a simple, undirected, edge-weighted graph has a spanning tree of weight at most } k\}$.

As language suitable for TM:
$L_{\text{SHORT SPANNING TREE}} = \{\langle G, k \rangle | k \text{ is a positive integer and } G$
$\text{is a simple, undirected, edge-weighted graph has a spanning tree of weight at most } k\}$.

Example - Graph And Possible Encoding

$A = \{\langle G \rangle | G \text{ is a connected undericted graph}\}$



- $\langle G \rangle$ = (1,2,3,4)((1,2),(2,3),(1,3),(1,4))
  vertices edges

High-Level Description of TM Deciding A

M = "On input $\langle G \rangle$:

1. Select first node of G; mark it
2. Repeat following step until no new nodes are marked
    - For each node v in G: mark v if incident to an edge where the other endpoint is already marked.
3. Scan all nodes of G: determine whether or not they all are marked.
    - If they are: accept
    - Otherwise: reject."

$\langle G \rangle = (1, 2, 3, 4)((2, 4), (2, 3), (3, 4), (1, 4))$

In Contrast: Implementation-Level Description A

M for input $\langle G \rangle$:

Step 0. Check whether input is proper encoding of graph.

- M scans for proper form (i.e., two lists: vertices, edges)
    - first list (node list): distinct decimal numbers
    - second list (edge list): pairs of decimal numbers
- M verifies that
    - node list contains no repetitions
        - use TM procedure for recognizing language $\{\#x_1\#x_2\#\dots\#x_l | each\ x_i \in \{0,1\}^* \text{ and } x_i \neq x_j \text{ for each } i \neq j\}$
    - every node appearing on edge list also appears on node list

Step 1. March first node with dot on leftmost digit.

Step 2.

1. find undotted node n_1, underline n_1
2. find dotted node n_2, underline n_2
3. test whether (n_1, n_2) is edge in graph
    - If (n_1, n_2) is in edge list: dot n_1, remove underlines; got to 1
    - Else (n_1, n_2) is no edge in graph
        - move underline on n_2 to next dotted node (if such node exists), call it n_2
        - If there are no more dotted nodes: n_1 is not attached to any dotted nodes.

- update underlines: n_1 is next undotted node, n_2 is first dotted node

4. If there are no more undotted nodes: continue with Step 3.

Step 3. M scans list of nodes to determine whether all are dotted.

- If yes: M accepts
  - If no: M rejects

$$
\begin{array}{l}
(1,2,3,4)((2,4),(2,3),(3,4),(1,4)) \\
(\dot{1},2,3,4)((2,4),(2,3),(3,4),(1,4)) \\
(\dot{\underline{1}},\underline{2},3,4)((2,4),(2,3),(3,4),(1,4)) \\
(\dot{\underline{1}},2,\underline{3},4)((2,4),(2,3),(3,4),(1,4)) \\
(\dot{\underline{1}},2,3,\underline{4})((2,4),(2,3),(3,4),(1,4)) \\
(\dot{1},2,3,\dot{4})((2,4),(2,3),(3,4),(1,4)) \\
(\dot{\underline{1}},\underline{2},3,\dot{4})((2,4),(2,3),(3,4),(1,4)) \\
(\dot{1},\underline{2},3,\dot{\underline{4}})((2,4),(2,3),(3,4),(1,4)) \\
(\dot{1},\dot{2},3,\dot{4})((2,4),(2,3),(3,4),(1,4)) \\
(\dot{\underline{1}},\dot{2},\underline{3},\dot{4})((2,4),(2,3),(3,4),(1,4)) \\
(\dot{1},\dot{\underline{2}},\underline{3},\dot{4})((2,4),(2,3),(3,4),(1,4)) \\
(\dot{1},\dot{2},\dot{3},\dot{4})((2,4),(2,3),(3,4),(1,4))
\end{array}
$$

Example – Other Decidable Problems / Languages

- $A_{DFA} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}$
- $A_{NFA} = \{\langle B, w\rangle \mid B \text{ is a NFA that accepts input string } w\}$
- $A_{REX} = \{\langle R, w\rangle \mid R \text{ is a regular expression that generates string } w\}$
- $E_{DFA} = \{\langle A\rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$
- $EQ_{DFA} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$

Question. Is the following language decidable?

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

Previous Lecture

Lecture12

Next Lecture

Lecture14