Jerry Lu

CS3331 - Final Notes Dec 14/19

Chapter 17 - Turing Machine and undecidability

A TM M's behaviour will only be defined on input strings that are finite and only contain characters in M's input alphabet (Σ)

How does a TM work? In each step:

1. Choose its next state
2. Write on current square
3. Move read/write head left or right one square

Since δ (transition) is a function, NOT a relation. Thus a TM is deterministic

transition on a TM (graphically) r/w/m = read/write/moveTo

A TM is NOT guaranteed to halt.

What does a Macro TM mean? You can combine smaller TMs to form a more complex one.

Example of shifting:

input: $\square u \underline{\square} w \square$

output: $\square u w \underline{\square}$

(Ans) M:

$$\begin{tikzcd} > R \arrow[r, "x \leftarrow \neg\square"] \arrow[d, "\square"] & \square L x R \arrow[l, bend right] \\ L & \end{tikzcd}$$

A language is DECIDABLE iff a TM either accept OR reject every string.

A language is SEMI-DECIDABLE iff a TM either accept OR does not accept (either reject or loop).

A non-deterministic TM, M, is DECIDABLE iff:

- ↳ M accepts $w \in \Sigma^*$ iff at least one computation accepts
- ↳ M rejects $w \in \Sigma^*$ iff all of its computations reject

A non-deterministic TM, M decides a language $L \subseteq \Sigma^*$ iff $\forall w \in \Sigma^*$
↳ There's a finite number of paths that M can follow on input w.
↳ All of those paths halt
↳ $w \in L$ iff M accepts w

A non-deterministic TM, M, SEMI-DECIDES a language $L \subseteq \Sigma^*$ iff $\forall w \in \Sigma^*$
↳ $w \in L$ iff $(s, \sqcup w)$ *starting state* yields at least one accepting configuration.

A non-deterministic TM, M, computes a function, f, iff $\forall w \in \Sigma^*$
↳ All of M's computations halt
↳ All of M's computations result in $f(w)$

Any computation by a TM with a two-way infinite tape can be simulated by a TM with a one-way tape

Universal Turing Machine:
1. A programmable TM that accepts (program, string) as input
2. It executes the program and produce a string as output

How to define an Universal Turing Machine, U.?
1. Define an encoding operation for TMs
2. Describe the operation of U given input $\langle M, w \rangle$.

How to encode the states?
↳ Let t' be the binary string assigned to state t
↳ If t is the halting state y, assign it the string yt'.
↳ If t is the halting state n, assign it the string nt'.
↳ If t is any other state, assign it the string qt'.

Ex. Let say we have 9 states, and then this is the encoding of the states:
q0000 (s), q0001, q0010, y0011, n0100, q0101, q0110, q0111, q1000
* State 3 is y & State 4 is n *

③

Jerry Lu

CS3331 - Final Notes (Continue) Dec 14/18

How to encode the tape alphabet (Γ)?

Example: Say we have a tape alphabet of $\Gamma = \{\square, a, b, c\}$. Then the encoding is:

$\square = a00$

$a = a01$

$b = a10$

$c = a11$

How to encode the transition?

Transition = (starting state, input string, destination state, ouput string, direction)

Ex. $(q000, a00, q0001, a00, \rightarrow)$

$q000 \xrightarrow{a00/a00/\rightarrow} q0001$

Theorem: There exists an infinite lexicographic enumeration of

① All syntactically valid TMs

② All syntactically valid TMs with specific input alphabet Σ.

③ All syntactically valid TMs with specific input AND tape alphabet, Σ & Γ.

Specification of the Universal TM:

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

- Halt iff M halts on w
- If M is a deciding or semi-deciding machine, then:
  - If M accepts, accept
  - If M rejects, reject
- If M computes a function, then $U(\langle M, w\rangle)$ must equal $M(w)$.

How does an Universal TM work? It uses 3 tapes as follow $U(\langle M, w\rangle)$

Tape 1: M's tape

Tape 2: $\langle M\rangle$, the "program" that U's running

Tape 3: M's state

Step ①: Copy the encoding of $\langle M, w\rangle$ onto tape 1.

Step ②: Transfer $\langle M\rangle$ from tape 1 to tape 2 (erasing it from tape 1)

Step ③: Find out the # of states in M, then let i be the binary digit of the # of states in M. Write $q0^i$ (corresponding to the start state of M) on tape 3.

	□	□	□	□	<w	---	--	-w>	
	0	0	0	0	1	0	0	0	
□	<M	--	--	-M>	□	□	□	□	□
	1	0	0	0	0	0	0	0	
	q	0	0	0	□	□	□	□	
	1	0	0	0	0	0	0	0	

How long does U take to simulate the computation of M? (If M halts in k steps)
↳ Ans: $O(|M| \cdot k)$ steps
↳ Explanation: Since U has to loop k times and each time U also has to loop through the entire <M> to find the corresponding computation. Hence $|M| \cdot k$

Chapter 19

$L_1 = \{<M, w> : $ TM M halts on input string $w\}$

For a string x to be in L_1, it must:
↳ Be syntactically well-formed
↳ Encode a machine M and a string w such that M halts when started on w.

Theorem: $H = \{<M, w> : $ TM M halts on input string $w\}$
↳ Is semi-decidable AND not decidable

Theorem: If H were in D, then every SD language would be in D.
↳ Proof: Suppose L is any SD language, then there's M_L that's semi-decidable.
Suppose H were in D, then there's O (oracle) that's decidable.
Pass $<M_L, w>$ as input into O if M_L halts on w.
$M'(w: \text{string}) =$
1. Run O on $<M_L, w>$
2. If O accepts (which it will iff M_L halts on w), then:
   2.1 Run M_L on w
   2.2 If M_L accepts, then accept. Else reject
3. Else Reject

(5)

Jerry Lu

CS3331 - Final Notes

Dec 14/18

Chapter 20

Theorem: The set of context-free languages is a proper subset of D.

3 relations between SD and D:

1. D is a subset of SD. In other words, every decidable language is also semi-decidable.
2. There exists at least one language that is in SD/D (Eg. H)
3. There exists languages that are not in SD.

Theorem: The class D is closed under complement

↳ What it means: If a language is decidable, then so is its complement

Theorem: The class SD is NOT closed under complement

Theorem: A language is in D iff both it and its complement are in SD

Theorem: $\neg H$ is not in SD

↳ Proof: We know that H is in SD, then if $\neg H$ is also in SD, this would mean H is also in D but we know that H is NOT in D. Therefore $\neg H$ cannot be in SD.

Theorem: A language is in SD iff it is Turing-Enumerable

↳ Proof (Right-to-Left): If a language, L, is Turing-Enumerable, then there's some TM that enumerates it. We convert M to M' that SD L.

M'(w: string) =

1. Save input w on a second tape
2. Run M. If a match is found from the result of "Running M" to w, then halt and accept.

* Running M means enumerate all strings in L.

∵ M' is SD.

∴ L is SD.

↳ Proof (Left-to-Right): If L is in SD, then there's a SD TM, M. We now construct M' using M to enumerate L in a lexicongraphical order.

M'() =

1. Enumerate all $w \in \Sigma^*$ lexicongraphically. For each string w_i:
   1.1 Start a copy of M with w_i as input
   1.2 Run M_i that are initiated, except those that are halted.
2. Whenever M_i accepts, output w_i

Theorem : A language is in D iff it is lexicongraphically Turing - enumerable

Chapter 21

How to use Reduction as a part of proof by contradiction that SD/D?

A Reduction , R, from L_1 to L_2 consists of one or more TMs with the following properties :
1. If there exists a TM, Oracle , that decides or semi-decides L_2,
2. Then the TMs in R can be composed with Oracle to build a decidable TM for L_1.

$(L_1 \leq L_2) \wedge (L_2 \text{ is in } D) \rightarrow (L_1 \text{ is in } D)$

* L_1 is reducable to L_2 *

Step 1 : Assume Oracle that decides L_2 exists

Step 2 : Choose a language L_1 :
- ↳ That is known to not be in D, and
- ↳ Can be reduced to L_2

Step 3 : Define Reduction R

Step 4 : Describe the composition C of R with Oracle :

$$C(x) = Oracle(R(x))$$

Step 5 : Show that C does correctly decide L_1 if Oracle exists . We do this by showing :
- R can be implemented by Turing Machines,
- C is correct :
  - ↳ If $x \in L_1$, then $C(x)$ accepts , and
  - ↳ If $x \notin L_1$, then $C(x)$ rejects

Rice Theorem

Any languages that are : $\{\langle M \rangle : P(L(M)) = True\}$

Non-trivial Property :
- ↳ true of at least one language
- ↳ false of at least one language