

Turing Machines

1. Infinite tape, divided into cells.
2. Read/Write Head
3. Finite Number of States
4. In each step, head can read/write and move left/right.

Turing Machines

Example:

Suppose we want to check if the input contains same number of a's as b's.

Turing Machines

State	a	b	B	X
q0	q1, X, R	q2, X, R	qA,B,R	q0, X, R
q1	q1, a, R	q3, X, L		q1, X, R
q2	q3, X, L	q2, b, R		q2, X, R
q3	q3, a, L	q3, b, L	q0,B,R	q3, X, L
qA				

Turing Machines

1. Function Computation
2. Language Acceptance

Turing Machines

Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$.

Q : a set of states

Σ : input alphabet set

Γ : tape alphabet. $\Sigma \subseteq \Gamma$.

δ : transition function from $Q \times \Gamma$ to $Q \times \Gamma \times \{L, R\}$.

q_0 : starting state

B : blank symbol. We assume $B \in \Gamma - \Sigma$

F : set of final sets. $F \subseteq Q$.

Usually, input is given without any blanks in between.

Instantaneous Description

1. We leave out blanks on both ends.

Exception: if head is among the blanks

2. $x_0x_1 \dots x_{n-1}qx_nx_{n+1} \dots x_m$.

3. $x_0x_1 \dots x_{n-1}qx_nx_{n+1} \dots x_m \vdash$ next ID

4. \vdash^* can be defined by saying ‘zero or more steps’.

$ID_1 \vdash ID_2 \vdash \dots \vdash ID_n$, then

$ID_1 \vdash^* ID_n$.

(Here n maybe 1).

Language Accepted by Turing Machine

TM accepts x , if

$$q_0x \vdash^* \alpha q_f \beta$$

where $q_f \in F$.

$$L(M) = \{x \mid q_0x \vdash^* \alpha q_f \beta, \text{ for some } q_f \in F\}.$$

Function Computed by Turing Machine

Languages/Functions

1. A language L is said to be *recursively enumerable* (RE), if some Turing Machine accepts the language L .
2. A language L is said to be *recursive* (*decidable*), if some Turing Machine accepts the language L , and Halts on all the inputs.
3. A function f is said to be *partial recursive*, if some Turing Machine computes the function (it halts on all the inputs on which f is defined, and it does not halt on inputs on which f is not defined).
4. A function f is said to be *recursive*, if some Turing Machine computes the function, and f is defined on all elements of Σ^* .

Turing Machine and Halting

Machine may never halt.

Cannot determine if a machine will halt on a particular input

Turing Machines

Many tricks for doing the computation with TM.

Turing Machines

1. Stay where you are: 'S' move.
2. Storage in Finite Control.
3. Multiple Tracks
4. Subroutines

Modifications of Turing Machines

Semi-Infinite Tapes

Initialization:

$$\delta(q_S, (X, B)) = ((q_0, U), (X, *), S).$$

Simulation:

Below $X, Z \in \Sigma$ and $M \in \{R, L\}$.

1. If $\delta(q, X) = (q', Y, M)$:

$$\delta((q, U), (X, Z)) = ((q', U), (Y, Z), M)$$

$$\delta((q, D), (Z, X)) = ((q', D), (Z, Y), \overline{M})$$

2. If $\delta(q, X) = (q', Y, R)$:

$$\delta((q, U), (X, *)) = ((q', U), (Y, *), R)$$

$$\delta((q, D), (X, *)) = ((q', U), (Y, *), R)$$

Modifications of Turing Machines

3. If $\delta(q, X) = (q', Y, L)$:

$$\delta((q, U), (X, *)) = ((q', D), (Y, *), R)$$

$$\delta((q, D), (X, *)) = ((q', D), (Y, *), R)$$

Modifications of Turing Machines

Multi Tape Turing Machines

Modifications of Turing Machines

Nondeterministic Turing Machines

Church-Turing Thesis

Whatever can be computed by an algorithmic device (in function computation sense, or language acceptance sense) can be done by a Turing Machine.

Codings of Strings

For a string x over $\{0, 1\}^*$, let $1x$ (in binary) -1 be its code.

Codings of Turing Machines

States: q_1, q_2, \dots are the states, with q_1 being start state and q_2 the only accepting state.

Tape symbols: X_1, X_2, \dots, X_s are tape symbols. X_1 is 0, X_2 is 1 and X_3 is blank.

Directions: L is D_1 and R is D_2 .

Coding Transition: $\delta(q_i, X_j) = (q_k, X_l, D_m)$, then code it using string

$0^i 10^j 10^k 10^l 10^m$.

(Note that each of i, j, k, l, m is at least 1).

Code of TM is: $C_111C_211C_3 \dots C_n$, where C_i are the codes of all the transitions in the TM.

Now we can convert the string to numbers, as above, if needed.

M_i denotes the Turing Machine with code number i .

$W_i = L(M_i)$ denotes the language accepted by Turing Machine with code number i .

A non-RE language

Let $L_d = \{w_i : w_i \notin L(M_i)\}$.

Recursive Languages

Theorem: If L is recursive, then \bar{L} is recursive.

Theorem: L is recursive iff L is RE and \bar{L} is RE.

Universal Turing Machine

$L_u = \{(M, w) : M \text{ accepts } w\}$.

We will construct a Universal TM which will accept the language L_u .

It has following tapes:

Input: M (coded) and w (over $\{0, 1\}^*$)

Tape for M : initially place w in coded form: 10 for zeros and 100 for ones in w .

State of M : initially contains 0 (head on 0).

Scratch tape

Simulating each step of M :

Undecidable Problems: Reductions

P_1 reduces to P_2 , if some recursive function f behaves as follows:

$$x \in P_1 \text{ iff } f(x) \in P_2$$

Undecidable Problems: Reductions

Suppose P_1 reduces to P_2 (notation: $P_1 \leq_m P_2$ or $P_1 \preceq P_2$).

Then,

- (a) If P_2 is recursive, then so is P_1 .
- (b) If P_2 is RE, then so is P_1 .
- (a') If P_1 is undecidable, then so is P_2 .
- (b') If P_1 is non-RE, then so is P_2 .

So, if we want to show that some problem P_2 is “hard” (not recursive, not RE), then

we pick a problem P_1 which is hard (not recursive, not RE) and then show that

$$P_1 \leq_m P_2.$$

For this, we construct a function f ,

and show that

$$x \in P_1 \text{ iff } f(x) \text{ in } P_2.$$

TMs accepting empty set.

Let $L_e = \{M : L(M) = \emptyset\}$.

Let $L_{ne} = \{M : L(M) \neq \emptyset\}$.

Theorem: L_{ne} is RE.

Theorem: L_e is not recursive.

Corollary: L_e is not RE.

We reduce $\overline{L_u}$ to L_e

Given $M\#w$ we construct M' defined as follows:

$M'(x)$

For $t = 0$ to ∞ .

If $M(w)$ accepts within t steps, then accept.

End For

Now,

$M(w)$ does not accept (i.e., $M\#w \in \overline{L_u}$) iff $L(M') = \emptyset$

and

$M(w)$ accepts (i.e., $M\#w \notin \overline{L_u}$) iff $L(M') = \Sigma^* \neq \emptyset$.

Same proof technique can be used for several variations:

$$L = \{M \mid M \text{ does not accept } a\}.$$

$$L = \{M \mid M \text{ does not accept } a \text{ or does not accept } b\}.$$

$$L = \{M \mid L(M) \text{ is finite } \}.$$

etc

Let $L_5 = \{M \mid L(M) \text{ has } \leq 5 \text{ elements} \}$

We reduce L_e to L_5 .

Rice's Theorem.

Suppose P is a property on RE languages. Then one may ask
Is $L_P = \{M \mid L(M) \text{ satisfies property } P\}$ is decidable? RE?

A property about RE languages is *non-trivial* if there exists at least one RE language which satisfies the property, and there exists at least one RE language which does not satisfy the property.

Rice's Theorem: Suppose P is a non-trivial property about RE languages. Then L_P is undecidable.

Post's Correspondence Problem.

Input: Two lists of strings $A = w_1, w_2, \dots, w_k$ and $B = x_1, x_2, \dots, x_k$.

Question: Does there exist i_1, i_2, \dots, i_m (where $m > 0$) such that

$$w_{i_1} w_{i_2} \dots w_{i_m} = x_{i_1} x_{i_2} \dots x_{i_m}$$

MPCP: What if we require that first string used should be w_1 and x_1 , that is:

Question: Does there exist i_1, i_2, \dots, i_m (where m maybe 0) such that

$$w_1 w_{i_1} w_{i_2} \dots w_{i_m} = x_1 x_{i_1} x_{i_2} \dots x_{i_m}$$

Post's Correspondence Problem.

We show that $L_u \leq_m MPCP \leq_m PCP$.

Post's Correspondence Problem.

Proof for $MPCP \leq_m PCP$

Suppose $A = w_1, w_2, \dots, w_k$ and $B = x_1, x_2, \dots, x_k$.

Then form

- (a) w'_i by inserting $*$ after each symbol in w_i .
- (b) x'_i by inserting $*$ before each symbol in x_i .
- (c) $w'_0 = *w'_1, x'_0 = x'_1$
- (d) $w'_{k+1} = \$$
- (e) $x'_{k+1} = *\$$

Post's Correspondence Problem.

$L_u \leq_m MPCP$.

Assume without loss of generality that machine for accepting L_u never writes a blank and never moves to the left of the leftmost symbol of input. Below $X, Y, Z \in \Gamma$ and $p, q \in Q$.

List A	List B	
#	# $q_0 w$ #	
X	X	
#	#	
qX	Y p	if $\delta(q, X) = (p, Y, R)$
ZqX	pZY	if $\delta(q, X) = (p, Y, L)$
q#	Yp#	if $\delta(q, B) = (p, Y, R)$
Zq#	pZY#	if $\delta(q, B) = (p, Y, L)$
XqY	q	if q is accepting state
Xq	q	if q is accepting state
qX	q	if q is accepting state
q##	#	if q is accepting state

Ambiguous Grammars

Suppose PCP problem is $A = w_1 \dots w_k$ and $B = x_1 \dots x_k$.

Let a_1, \dots, a_k be letters different from those used in A, B .

Consider the grammar

$$S \rightarrow A$$

$$S \rightarrow B$$

$$A \rightarrow w_i A a_i, \text{ for } 1 \leq i \leq k$$

$$A \rightarrow w_i a_i, \text{ for } 1 \leq i \leq k$$

$$B \rightarrow x_i B a_i, \text{ for } 1 \leq i \leq k$$

$$B \rightarrow x_i a_i, \text{ for } 1 \leq i \leq k$$

Grammar is ambiguous iff $L(G_A)$ and $L(G_B)$ intersect, iff *PCP* has a solution.

(Here G_A is the grammar for the productions involving A above, and G_B is the grammar for the productions involving B above).

Further Undecidable Problems

Note that $\overline{L(G_A)}$, $\overline{L(G_B)}$ are context free.

Following are undecidable:

1. $L(G_1) \cap L(G_2) = \emptyset$, for CFGs.

Proof: by taking $G_1 = G_A$ and $G_2 = G_B$.

2. $L(G_1) = L(G_2)$, for CFGs

By taking $L(G_1) = \overline{L_A} \cup \overline{L_B} = \overline{L_A \cap L_B}$, and G_2 to be a grammar for $(\Sigma \cup I)^*$.

3. $L(G) = L(R)$, for CFG G and regular language R .

4. $L(G) = T^*$, for CFG G and alphabet T .

5. $L(G_2) \subseteq L(G_1)$

By taking G_1, G_2 as in 2 above.

6. $L(R) \subseteq L(G)$

Unrestricted Grammars

An *unrestricted grammar* is a 4-tuple $G = (N, \Sigma, S, P)$.

N is the alphabet of non-terminals.

Σ is the alphabet of terminals with $N \cap \Sigma = \emptyset$.

S is start symbol.

P is a finite set of productions of form $\alpha \rightarrow \beta$, where $\alpha \in (N \cup \Sigma)^+$ and $\beta \in (N \cup \Sigma)^*$.

Example:

$$S \rightarrow aSBC \mid aBC$$

$$CB \rightarrow BC$$

$$aB \rightarrow ab$$

$$bB \rightarrow bb$$

$$bC \rightarrow bc$$

$$cC \rightarrow cc$$

Theorem: If G is an unrestricted grammar, then $L(G)$ is RE.

Theorem: If L is RE, then there exists a grammar G such that $L = L(G)$.

Suppose L is accepted by TM $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$

Start:

$$S \rightarrow BS|SB|A_1A_2$$

$$A_2 \rightarrow aA_2|A_2a|q, \text{ for all } a \in \Sigma, q \in F.$$

Transitions:

$$bq_j \rightarrow q_ia, \text{ if } \delta(q_i, a) = (q_j, b, R)$$

$$q_jcb \rightarrow cq_ia, \text{ if } \delta(q_i, a) = (q_j, b, L)$$

Cleanup:

$$A_1q_0 \rightarrow \epsilon$$

$$B \rightarrow \epsilon$$