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 states. $F \subseteq Q$.
Usually, input is given without any blanks in between.

Instantaneous Description

- We leave out blanks on both ends.
  Exception: if head is among the blanks .....
- $x_0 x_1 \ldots x_{n-1} q x_n x_{n+1} \ldots x_m$.
- $x_0 x_1 \ldots x_{n-1} q x_n x_{n+1} \ldots x_m \vdash$ next ID
- $\vdash^*$ can be defined by saying 'zero or more steps'.
  $ID_1 \vdash ID_2 \vdash \ldots \vdash ID_n$, then
  $ID_1 \vdash^* ID_n$.
  (Here n maybe 1).

Language Accepted by Turing Machine

TM accepts x , if

$$q_0 x \vdash^* \alpha q_f \beta$$

where $q_f \in F$.
$L(M) = \{x \mid q_0 x \vdash^* \alpha q_f \beta$, for some $q_f \in F\}$.

Function Computed by Turing Machine

1. Require that the machine halt on input x iff $f(x)$ is defined.
2. Interpret the content of the tape(s) of the Turing machine, after it halts, as the output of f. There are various ways of doing this, for example, by just considering the non-blank content of the first (only) tape.

Languages/Functions

- A language L is said to be *recursively enumerable* (RE), (computably enumerable, CE) if some Turing Machine accepts the language L.
- A language L is said to be *recursive* (*decidable*), if some Turing Machine accepts the language L, and Halts on all the inputs.
- A function f is said to be *partial recursive* (partially computable), 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).
- A function f is said to be *recursive* (computable), 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 $X, Z \in \Gamma$ and $m \in \{R, L\}$.

$\overline{m}$ denotes L if $m = R$; otherwise $\overline{m} = R$.

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

Multi Tape Turing Machines

Nondeterministic Turing Machines

— $\delta(q, a)$ is a finite set of possibilities.
— For any current ID, there maybe finite number of possible next ID
— Acceptance, if there exists an accepting state q_f such that
$q_0 x \vdash^* \alpha q_f \beta$.

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. w_i denotes the string with code i.

Codings of Turing Machines

- States: $q_1, q_2, \ldots$ are the states, with q_1 being start state and q_2 the only accepting state.
- Tape symbols: $X_1, X_2, \ldots, X_s$ 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
  $0^i 1 0^j 1 0^k 1 0^l 1 0^m$.
  (Note that each of i, j, k, l, m is at least 1).
- Code of TM is: $C_1 11 C_2 11 C_3 \ldots C_n$, where C_i are the codes of all the transitions in the TM.

Codings of Turing Machines

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 $L_d = \{w_i : w_i \notin L(M_i)\}$.
Then L_d is not RE.
Proof:
Suppose any M_j is given. We will show that $L(M_j) \neq L_d$.
Case 1: $w_j \in L(M_j)$. Then, $w_j \notin L_d$ by definition of L_d.
Thus, $L(M_j) \neq L_d$.
Case 2: $w_j \notin L(M_j)$. Then, $w_j \in L_d$ by definition of L_d.
Thus, $L(M_j) \neq L_d$.
Hence, in both cases, $L(M_j) \neq L_d$. Since this applies for any M_j, we have that L_d is not accepted by any Turing Machine.
Hence L_d is not RE.

Recursive Languages

Theorem: If L is recursive, then $\overline{L}$ is recursive.
Proof: Suppose $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ accepts L and halts on all the inputs. Then, modify M as follows to form a new machine M':
(i) assume without loss of generality that there is only one accepting state (q_{acc}) in M and that there is no transition in M from an accepting state
(ii) create a new state q_{new}
(iii) for any non-accepting state q and any letter a of the alphabet, if $\delta(q, a)$ is not defined in M, then let $\delta(q, a) = q_{new}$ in M'. Other transitions in M' are as in M.
(iv) Let q_{new} be the only accepting state of M' (thus q_{acc} is a non-accepting state of M').

Then, M' accepts $\overline{L}$. To see this, note that if M accepts x, then it eventually reaches the state q_{acc} and then never leaves it. Thus, M' on input x also eventually reaches q_{acc} and never leaves it (without ever going through state q_{new}). On the other hand, if M does not accept x, then it never goes through q_{acc} and eventually halts by having no transition at some point of time. Thus, M' on input x eventually reaches state q_{new} and then never leaves it. Thus, M' accepts x.

Theorem: L is recursive iff L is RE and $\overline{L}$ is RE.
Proof: If L is recursive, then $\overline{L}$ is also recursive. Thus, both L and $\overline{L}$ are RE.
For the other direction, suppose M accepts L and M' accepts $\overline{L}$. Then consider the Turing Machine M'' which works as follows:
$M''(x)$ first copies input x into two different tapes. Then, in parallel, it runs $M(x)$ on the first tape, and $M'(x)$ on the second tape. Note that running in parallel can be done in a way similar to that done for DFAs.
Then, if at any point of time $M(x)$ accepts, then M'' accepts. If $M'(x)$ accepts, then M'' halts and rejects.

Now, if $x \in L$, then $M(x)$ eventually accepts, and thus M'' halts and accepts. If $x \notin L$, then $M'(x)$ eventually accepts, and thus M'' halts and rejects.
Thus, L is recursive.

Universal Turing Machine

$L_u = \{(M, w) : M$ 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

At any time during the simulation, before/after simulating a step of M, we will have
— the state of the M is in the tape called state of M (in form 0^s),
— the tape "tape for M" contains the content of M's tape (in coded form, as mentioned earlier, 10 for zero, 100 for one, etc), where the trailing blanks need not be coded.
— the head of UTM on the tape corresponding to the tape for M is at the 1 corresponding to the code for the content of the cell scanned by M.

Simulating each step of M:

1. suppose the tape containing state of M has 0^i
2. suppose the head of UTM as on the tape "tape for M" is at the start of 10^j1 (or 10^jB).
3. Search in the code of M for an entry of the form $0^i10^j10^k10^\ell10^m$, where on both sides we have either blanks or 11, and i and j are as in steps 1 and 2 above. Note that this can be done using comparison of various tape contents.
4. Then change the content of the tape containing state of TM to 0^k (this can be done using copying).

5.1 To write 0^ℓ on the tape, first mark the current "cell" being read on the 'Tape for M' by using a special symbol $*$ (the $*$ is used to replace the 1 in 10^j).
5.2 Copy the Tape for M into scratch tape, where instead of copying the portion $*0^j1$ write $*0^\ell1$ in the scratch tape (where the last 1 might be a B, in case the current cell used by M is the last written cell). Additionally, convert one blank at the end by using 10^3.
5.3 Copy back the scratch tape into the tape "Tape for M".
5.4. Movement of the head can be easily implemented now by reaching the marked portion $*$, replacing it by 1, and then moving the head to the next 1 on the left/right as the case maybe depending on the value of m.

$\overline{L_u}$ is not RE.
To see this suppose by way of contradiction that M accepts $\overline{L_u}$. Then, one can construct a machine M' for accepting L_d as follows:
$M'(w_i)$:
1. Extract the code i from w_i.
2. Run M on (M_i, w_i) (where M_i is coded appropriately).
3. Accept iff M accepts in the above execution.

Thus, as L_d is not RE, we have that $\overline{L_u}$ is not RE.

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 $P_1 \preceq_m P_2$). 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
$x \in P_1$ iff $f(x)$ 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.

Note: We have used M in sets above. Strictly speaking we should use i, w_i or 1^i or something similar (where i is a code for M).

Theorem: L_{ne} is RE.
Below is an informal description of the Turing Machine M' to accept L_{ne}.
M' on input M (in coded form) works as follows:

For $t = 0$ to ∞
For $i = 0$ to t
If $M(w_i)$ accepts within t steps, then accept.
Endfor
Endfor

Theorem: L_e is not RE.
We reduce $\overline{L_u}$ to L_e
Given $M\#w$ we construct M' defined as follows (that is the reduction function f maps $M\#w$ to M'):

$M'(x)$
For $t = 0$ to ∞.
If $M(w)$ accepts within t steps, then accept.
End For
End M'

Note that the mapping f: $M\#w \rightarrow M'$ is recursive.
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 $L(M') = \Sigma^* \neq \emptyset$.
Thus, $\overline{L_u} \leq_m L_e$.

Same proof technique can be used for several variations:
$L = \{M \mid M$ does not accept $a\}$.
$L = \{M \mid M$ does not accept a or does not accept $b\}$.
$L = \{M \mid L(M)$ is finite $\}$.
etc

Let $L_5 = \{M \mid L(M)$ has ≤ 5 elements $\}$
We reduce L_e to L_5.
$f : M \rightarrow M'$
$M'(x)$
For $t = 0$ to ∞
For $i = 0$ to t
If $M(w_i)$ accepts within t steps, then accept
Endfor
Endfor

Now, if $L(M) = \emptyset$, then $L(M') = \emptyset$, and thus $M' \in L_5$
If $L(M) \neq \emptyset$, then $L(M') = \Sigma^*$ and thus $M' \notin L_5$
Thus, $L_e \leq_m L_5$.

Rice's Theorem.

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

Proof: Suppose P is a non-trivial property about RE languages.
Without loss of generality assume $\emptyset$ satisfies P (otherwise, switch P and $\overline{P}$).
Suppose L is an RE language that does not satisfy P. Let M'' be the machine which accepts L.
Define f as follows. $f(M) = M'$ such that M' is defined as follows.

$M'(x)$:
For $t = 0$ to ∞ do:
For $i = 0$ to t do:
  If $M(w_i)$ accepts within t steps and
  $M''(x)$ accepts within t steps, then accept x.
EndFor
EndFor
If $L(M) = \emptyset$, then $L(M') = \emptyset$.
If $L(M) \neq \emptyset$, then $L(M') = L$.
Thus, f reduces L_e to L_P.
As L_e is not recursive, we have that L_P is not recursive.

Post's Correspondence Problem

Input: Two lists of strings $A = w_1, w_2, \ldots, w_k$ and $B = x_1, x_2, \ldots, x_k$.
Question: Do there exist $i_1, i_2, \ldots, i_m$ (where $m > 0$) such that

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

MPCP: What if we require that first string used should be some particular combination (say w_1 and x_1), that is:
Question: Does there exist $i_1, i_2, \ldots, i_m$ (where m maybe 0) such that

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

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

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) $x'_{k+1} = *\$$

Here, $*, \$$ are special symbols not in the alphabet of w_i, x_i.

Note that if the PCP problem has a solution, then the MPCP problem created above has a solution (by using the same indices, where $k+1$ is dropped, and 0 is changed to 1).
Here note that the PCP problem solution must start with the index 0.
On the other hand if the MPCP problem above has a solution then the PCP problem has a solution by replacing the first index 1 by 0, using the rest of the indices as it is, and then ending with the index $k+1$.

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 $A = w_1 \ldots w_k$ and $B = x_1 \ldots x_k$.
Let $a_1, \ldots, a_k$ be letters different from those used in A, B.
Consider the grammar
$S \to A$
$S \to B$
$A \to w_i A a_i$, for $1 \leq i \leq k$
$A \to w_i a_i$, for $1 \leq i \leq k$
$B \to x_i B a_i$, for $1 \leq i \leq k$
$B \to x_i a_i$, 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.

To see this, let

I be set of indices.

A_i generate the set of proper prefixes of w_i.

A'_i generate the set of strings which are not prefixes of w_i, and are of length at most $|w_i|$.

The following grammar generates complement of $L(G_A)$:
$S \to \epsilon \mid C \mid D \mid E \mid F$
(Comment: C is for generating strings where elements of Σ follow elements of I.)
$C \to (\Sigma \cup I)C \mid C(\Sigma \cup I) \mid I\Sigma$
(Comment: D is for generating strings when there is a mismatch between indices and corresponding strings)
$D \to w_i D a_i \mid A'_i D_1 a_i$
$D_1 \to \Sigma D_1 \mid D_1 I \mid \epsilon$
(Comment: E is for generating strings when there is excess elements from Σ).
$E \to w_i E a_i \mid E_1$
$E_1 \to \Sigma E_1 \mid \Sigma$

(Comment: F is for generating strings when there are excess indices).

$F \rightarrow w_i F a_i \mid A_i F_1 a_i$

$F_1 \rightarrow I F_1 \mid \epsilon$

Further Undecidable Problems

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

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

Where, $I = \{a_1, a_2, \ldots\}$.

3. Similarly: $L(G) = L(R)$ is undecidable, for CFG G and regular expression R.

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

$L(G_2) \subseteq L(G_1)$ is undecidable, for CFG G_1, G_2.

$L(R) \subseteq L(G)$ is undecidable, for CFG G and regular expression R.

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 $\alpha \to \beta$, where $\alpha \in (N \cup \Sigma)^* N (N \cup \Sigma)^*$ and $\beta \in (N \cup \Sigma)^*$.
In some cases, people also let $\alpha \in (N \cup \Sigma)^+$ (i.e., do not require presence of a non-terminal on the left hand side).

If additionally $|\alpha| \leq |\beta|$ always for a production $\alpha \to \beta$ in P, then the grammar is said to be context sensitive grammar.

$\gamma\alpha\gamma' \Rightarrow \gamma\beta\gamma'$, if $\alpha \to \beta$ is a production.

$L(G) = \{w \in \Sigma^* : S \Rightarrow^* w\}$

Example:

$\{a^n b^n c^n : n \geq 1\}$

$S \to aSBC \mid aBC$

$CB \to BC$

$aB \to ab$

$bB \to bb$

$bC \to bc$

$cC \to cc$

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 $L = L(G)$.
Assume M doesn't write a B on the tape.
Suppose L is accepted by TM $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$
Start:
$S \to BS \mid SB \mid A_1A_2$
$A_2 \to aA_2 \mid A_2a \mid q$, for all $a \in \Gamma$, $q \in F$.
Transitions: for all $a, b, c \in \Gamma$, $q_i, q_j \in Q$,
$bq_j \to q_ia$, if $\delta(q_i, a) = (q_j, b, R)$
$q_jcb \to cq_ia$, if $\delta(q_i, a) = (q_j, b, L)$
Cleanup:
$A_1q_0 \to \epsilon$
$B \to \epsilon$