Formal Language and Automata Theory (CS21004)

Course Coverage

Class: CSE 2^{nd} Year

4^{th} January, 2010 (2 hours):

Tutorial-1 +

Alphabet: $\Sigma, \Gamma, \cdots$, string over Σ, $\Sigma^0 = \{\varepsilon\}$, $\Sigma^n = \{x: x = a_1 a_2 \cdots a_n,\ a_i \in \Sigma,\ 1 \leqslant i \leqslant n\}$, $\Sigma^\star = \bigcup_{n \in \mathbb{N}} \Sigma^n$, $\Sigma^+ = \Sigma^\star \backslash \{\varepsilon\}$, $(\Sigma^\star, con, \varepsilon)$ is a *monoid*. Language L over the alphabet Σ is a subset of $\Sigma^\star$, $L \subseteq \Sigma^*$.

The size of Σ^* is *countably infinite*, so the collection of all languages over Σ, $2^{\Sigma^\star} \simeq 2^{\mathbb{N}}$, is *uncountably infinite*. So every language cannot have finite description.

5^{th} January, 2010 (1 hour):

No set is equinumerous to its power-set (*Cantor*). The proof is by *reductio ad absurdum* (reduction to a contradiction).

An *one to one* map from A to 2^A is easy to get, $a \mapsto \{a\}$.

Let A be a set and there is an *onto* map (*surjection*) f from A to 2^A. We consider the set $B = \{x \in A: x \notin f(x) \in 2^A\}$. As $f: A \to 2^A$ is a surjection, there is an element $a_0 \in A$ so that $f(a_0) = \emptyset \in 2^A$, but then $a_0 \notin f(a_0) = \emptyset$. So, $a_0 \in B$, and B is a non-empty subset of A. So, there is an element $a_1 \in A$ such that $f(a_1) = B$. Does $a_1 \in f(a_1) = B$? This leads to contradiction.

If $a_1 \in f(a_1) = B$, then $a_1 \notin B$, if $a_1 \notin f(a_1) = B$, then $a_1 \in B$ i.e. $a_1 \in f(a_1) = B$ if and only if $a_1 \notin f(a_1) = B$. So there is no surjection possible and the set 2^A is more numerous than A.

Decision problems from different areas of computing can be mapped to decision problems in formal language.

- REACHABLE $= \{<G, s, d>: G$ is a directed graph and the destination node d is reachable from the source node $s\}$.
- PRIME $= \{n \in \mathbb{N}: n \,\text{is}\, a \,\text{prime}\}$.
- EULERPATH $= \{<G>:$ there is an Eulerian walk in the undirected graph $G\}$.
- INVMAT $= \{<M>: M$ is an invertible matrix over rationals$\}$.

Let Σ be an alphabet. $2^{\Sigma^\star}$is the collection of languages over Σ. We know that both $<\Sigma^\star, conc, \varepsilon>$ and $<2^{\Sigma^\star}, conc, \{\varepsilon\}>$ are monoids. Let $L, L_1, L_2 \in 2^{\Sigma^\star}$, the set operations $L_1 \cup L_2, L_1 \cap L_2, L_1 - L_2$ are defined as usual.

Concatenation: $L_1 L_2 = \{x \in \Sigma^\star: \exists y \in L_1 \exists z \in L_2 \text{ so that } x = yz\}$, $L^0 = \{\varepsilon\}$, $L^n = L L^{n-1}$, $n > 0$.

Kleene Closure/star: $L^\star = \bigcup_{n=0}^{\infty} L^n$, $L^+ = \bigcup_{n=1}^{\infty} L^n$.

6^{th} January, 2010 (1 hour):

Right quotient and right derivative: $L_1 \backslash L_2 = \{x \in \Sigma^\star: \exists y \in L_2 \wedge xy \in L_1\}$, $\partial_y^r(L) = \{x \in \Sigma^\star: xy \in L\} = L \backslash \{y\}$.

Left quotient and left derivative: $L_2 / L_1 = \{y \in \Sigma^\star: \exists x \in L_2 \wedge xy \in L_1\}$, $\partial_x^l(L) = \{y \in \Sigma^\star: xy \in L\} = \{x\}/L$.

Reverse or mirror image: $\varepsilon^R = \varepsilon$, $(ax)^R = x^R a$, $L^R = \{x^R \in \Sigma^\star: x \in L\}$.

Substitution and homomorphism: Let Σ_a be an alphabet for each $a \in \Sigma$ and L_a be a language over Σ_a. The map $\sigma(a) = L_a$ for all $a \in \Sigma$ induces a map $\sigma: \Sigma^\star \to 2^{\Sigma^\star}$ so that $\sigma(\varepsilon) = \{\varepsilon\}$ and $\sigma(ax) = \sigma(a)\sigma(x)$ [in other words $\sigma(xy) = \sigma(x)\sigma(y)$]. The map $\sigma: \Sigma^\star \to 2^{\Sigma^\star}$ is called *substitution*. It is ε-free if no L_a has ε in it. A *substitution* is a *homomorphism* if $|L_a| = 1$ for all $a \in \Sigma$.

Finite description of languages - *phrase structure* grammar or *type-0* grammar.

$G = (N, \Sigma, P, S)$, where

i. N is a finite set of *variables* or *non-terminals*,

ii. Σ finite set of *object language* symbols called *constants* or *terminals*,

iii. $S \in N$ is a special symbol called the *start symbol* or *axiom*,

iv. P is a finite subset of $((N \cup \Sigma)^\star N (N \cup \Sigma)^\star \times (N \cup \Sigma)^\star)$ called the *production* or *transformation* or *rewriting* rules.

An element $(\alpha, \beta) \in P$ is such that $\alpha = uAv$, where $u, v, \beta \in (N \cup \Sigma)^\star$ and $A \in N$, there must be a non-terminal in the first component of a production rule. The ordered pair of the production rule is written as $\alpha \rightarrow \beta$.

11$^{\text{th}}$ January, 2010 (2 hour):

i. *Phrase-Structure Grammar (PSG)*: as defined earlier. This is also called a *unrestricted grammar* or *type-0 grammar*.

ii. *Context-Sensitive Grammar (CSG):* Each production rule is of the $\alpha A \beta \rightarrow \alpha u \beta$, where $\alpha, \beta \in (N \cup \Sigma)^\star$, $A \in N$, $u \in (N \cup \Sigma)^+$ i.e. one non-terminal from the left-side of the production rule will be replaced by a non-null string to form the right side of the production. This is also called a *type-1 grammar*.

iii. *Length Increasing Grammar (LIG)*: In each production rule the length of the right side string is not shorter than the length of the left side string i.e. if $u \rightarrow v \in P$, $|u| \leqslant |v|$.

It is clear that any *context-sensitive grammar* is a *length-increasing grammar*. But it can also be proved that for every *length-increasing grammar* there is an equivalent *context-sensitive grammar*.

iv. *Context-Free Grammar:* Each production rule is of the form $A \rightarrow \alpha$, where $A \in N$ and $\alpha \in (N \cup \Sigma)^\star$. Replacement of a non-terminal does not depends on the context. This is also called a *type-2 grammar*.

v. *Right-linear Grammar:* Each production rule is either of the following two forms, $A \rightarrow xB$ or $A \rightarrow x$, where $A \in N$ and $x \in \Sigma^\star$. Without loss of power we can take $x \in \Sigma \cup \{\varepsilon\}$. This is also called a *type-3 grammar* or *regular grammar*.

Given a grammar $G = (N, \Sigma, P, S)$,we define the binary relation '*one step derivation*' ($\Rightarrow$) on the set $(N \cup \Sigma)^\star$. If $\alpha u \beta$ and $\alpha v \beta$ are two strings of $(N \cup \Sigma)^\star$ and $u \rightarrow v \in P$, we say that $\alpha u \beta$ derives or produces $\alpha v \beta$ in the grammar G in one step, and write $\alpha u \beta \overset{G}{\Rightarrow} \alpha v \beta$. We shall drop G from $\Rightarrow$ if there is no scope of confusion.
The *reflexive-transitive closure* of '*one step derivation*' relation gives the notion of derivation in any finite number of steps (including 0), $\overset{\star}{\Rightarrow}$. We shall often drop the '$\star$' and abuse the notation $\Rightarrow$ for both.
Sentential form and *sentence*: Given a grammar G, any string that can be derived from the start symbol S in finite number of states is a sentential for, $S \overset{\star}{\Rightarrow} u$, u is a sentential form. It is a sentence if it is a string of $\Sigma^\star$.
Language: Given a grammar $G = (N, \Sigma, P, S)$, the language generated by the grammar or language described by the grammar is the collection of all sentences. $L(G) = \{x \in \Sigma^\star \colon S \overset{\star}{\Rightarrow} x\}$. The language of a *context-sensitive grammar (CSG)* is called a *context-sensitive language (CSL)*, the language of a *length-increasing grammar (LIG)* is also a *CSL* (as the grammars are equivalent). The language of a *context-free grammar (CFG)* is called a *context-free language (CFL)*. The language of a *right-linear grammar* is called a *regular set* or a *regular language*.

Example 1. Following is a length-increasing grammar for the language $L = \{a^n b^{2n} c^n \colon n \geqslant 1\}$
$G_1 = (\{S, B\}, \{a, b, c\}, P, S)$, the production rules are,

$$\begin{aligned} S &\rightarrow aSBBc \\ S &\rightarrow abbc \\ cB &\rightarrow Bc \\ bB &\rightarrow bb \end{aligned}$$

The grammar is not context-sensitive due to presence of the rule $cB \rightarrow Bc$. We replace it by three context-sensitive rules and get the context-sensitive grammar of the same language. In doing so we first replace the terminal 'c' by a new non-terminal D.

$$\begin{aligned} S &\rightarrow aSBBD \\ S &\rightarrow abbD \\ DB &\rightarrow DE \\ DE &\rightarrow BE \\ BE &\rightarrow BD \\ bB &\rightarrow bb \\ D &\rightarrow c \end{aligned}$$

Following is a context-free grammar for the language $L = \{x \in \Sigma^\star : |x|_a = |x|_b\}$.

$G_2 = (\{S\}, \{a, b\}, P, S)$, the production rules are

$$\begin{aligned} S &\rightarrow aSb \\ S &\rightarrow bSa \\ S &\rightarrow SS \\ S &\rightarrow \varepsilon \end{aligned}$$

Following is a right-linear grammar, what is the language?

$G_3 = (\{S, A\}, \{a, b\}, P, S)$, the production rules are

$$\begin{aligned} S &\rightarrow aA \\ S &\rightarrow bS \\ S &\rightarrow \varepsilon \\ A &\rightarrow aS \\ A &\rightarrow bA \end{aligned}$$

12

12^{th} January, 2010 (1 hour): Tutorial II

13^{th} January, 2010 (1 hour): We first prove that for every length-increasing grammar G there is a context-sensitive grammar G', so that they are equivalent i.e. $L(G) = L(G')$. Without any loss of generality we take the rules of LIG in any one of the following form:

$$\begin{aligned} &A \rightarrow a \\ &A_1 A_2 \cdots\cdots A_m \rightarrow B_1 B_2 \cdots\cdots B_n \end{aligned}, \text{ where } A, A_1, \cdots, A_m, B_1, \cdots, B_n \in N$$

and $x \in \Sigma$, and $m \leqslant n$. We have replaced every terminal 'a' from the productions by new non-terminal A' and add a production $A' \rightarrow a$.

Example 2. Consider the grammar $G_1 = (\{S, B\}, \{a, b, c\}, P, S)$, where the production rule P is

$$\begin{aligned} S &\rightarrow aSBBc \\ S &\rightarrow abbc \\ cB &\rightarrow Bc \\ bB &\rightarrow bb \end{aligned}$$

The transformed grammar is $G_1' = (\{S, B, A', B', C'\}, \{a, b, c\}, P', S)$, where the production rules are

$$\begin{aligned} S &\rightarrow A' SBBC \\ S &\rightarrow A' B' B' C' \\ C'B &\rightarrow BC' \\ B' B &\rightarrow B' B' \\ A' &\rightarrow a \\ B' &\rightarrow b \\ C' &\rightarrow c \end{aligned}$$

The rules of first type and the second type rule with $m = 1$ are context-sensitive rules. So we are interested about the second type of rule where $m \geqslant 2$. We replace $A_1A_2\cdots\cdots A_m \rightarrow B_1B_2\cdots\cdots B_n$ by the following set of $2m$ rules,

$$
\begin{array}{l}
A_1A_2\cdots\cdots A_m \rightarrow C_1A_2\cdots\cdots A_m\\
C_1A_2\cdots\cdots A_m \rightarrow C_1C_2\cdots\cdots A_m\\
\qquad\qquad\vdots\\
C_1A_2\cdots\cdots A_{m-1}A_m \rightarrow C_1C_2\cdots\cdots C_{m-1}A_m\\
C_1A_2\cdots\cdots C_{m-1}A_m \rightarrow C_1C_2\cdots\cdots C_{m-1}C_mB_{m+1}\cdots B_n\\
C_1C_2\cdots\cdots C_{m-1}C_mB_{m+1}\cdots B_n \rightarrow B_1C_2\cdots\cdots C_{m-1}C_mB_{m+1}\cdots B_n\\
B_1C_2\cdots\cdots C_{m-1}C_mB_{m+1}\cdots B_n \rightarrow B_1B_2\cdots\cdots C_{m-1}C_mB_{m+1}\cdots B_n\\
\qquad\qquad\vdots\\
B_1B_2\cdots\cdots B_{m-1}C_mB_{m+1}\cdots B_n \rightarrow B_1B_2\cdots\cdots B_{m-1}B_mB_{m+1}\cdots B_n
\end{array}
$$

All these rules are context-sensitive in nature.

Soundness and *Completeness*: Given a language L and a grammar G we have to establish that $L = L(G)$. There are two parts of the process - we have to prove that the grammar does not generate any string outside L i.e. $L(G) \subseteq L$ - the grammar is *sound*. Every string of the language is generated by the grammar, $L \subseteq L(G)$ - the grammar is *complete*.

18$^{\text{th}}$ January, 2010 (2 hour): No class due to Death of Jyoti Basu.

19$^{\text{th}}$ January, 2010 (1 hour): Rooted tree, Parse or derivation tree. Ambiguously derived string and ambiguous grammar. Inherently ambiguous language. Simplification of a CFG - removal of useless symbol.

25$^{\text{th}}$ January, 2010 (2 hour): 1 hour tutorial +

Elimination of ε-production and elimination of unit-production. *Deterministic finite automaton* (DFA) - $M = (Q, \Sigma, \delta, s, F)$, state transition function δ: $Q \times \Sigma \rightarrow Q$, state transition diagram, state transition table, $\hat{\delta}$: $Q \times \Sigma^\star \rightarrow Q$, string accepted by M, language of M, $L(M) = \{x \in \Sigma^\star : \hat{\delta}(s, x) \in F\}$.

25$^{\text{th}}$ January, 2010 (1.5 hour)(compensation for 18$^{\text{th}}$): Examples of DFA, *Non-deterministic finite automaton* (NFA) - $N = (Q, \Sigma, \delta, s, F)$, state transition function δ: $Q \times \Sigma \rightarrow 2^Q$, $\hat{\delta}$: $Q \times \Sigma^\star \rightarrow 2^Q$, $\delta(P, a)$, where $P \subseteq Q$. Equivalence of DFA and NFA - subset construction.

27$^{\text{th}}$ January, 2010 (1 hour): Subset construction, NFA with ε-transition and its equivalence with NFA without ε-transition (not done properly).

1$^{\text{st}}$ February, 2010 (2 hour): 1 hour tutorial +

NFA with ε-transition, equivalence of NFA with ε-transition and NFA without ε-transition, regular expression and its language.

2$^{\text{nd}}$ February, 2010 (1 hour): ε-NFA from regular expression, L_x, derivative of L with respect to x, if L is regular then so is L_x. Unique solution of $X = \mathrm{A}X + B$ when $\varepsilon \notin A$, $X = A^\star B$. Regular expression from DFA - solution of simultaneous set equations.

3$^{\text{rd}}$ February, 2010 (1 hour): Regular expression from DFA using state equations. Closure properties.

8$^{\text{th}}$ February, 2010 (2 hour): 1 hour tutorial +

Closure properties of regular languages: closure under boolean operations, concatenation, Kleene-star, reversal, homomorphism, inverse homomorphism.

9$^{\text{th}}$ February, 2010 (1 hour): Closure properties, pumping theorem - proving a language non-regular, decidability results.

10th February, 2010 (1 hour): Myhill-Nerode Theorem - identification of regular languages as union of equivalence classes of a right invariant equivalence relation of finite index. Regular language - a countable boolean algebra. Given a finite state transition diagram with k states on an alphabet Σ, we can define 2^k DFAs (set of final states may be any subset of k states). These 2^k languages forms a boolean algebra.

15th February, 2010 (2 hour): 1 hour tutorial +

Minimisation of DFA, Minimisation algorithm and equivalence of two DFAs, Finite Automata with output - Moore and Mealy machine.

16th February, 2010 (1 hour) - ??? - definition of a PDA

2nd March, 2010 (1 hour) - Definition of a PDA - acceptance of a string by empty stack of a PDA M, the language $N(M)$, acceptance of a string at a final state by a PDA M, the language $T(M)$. A language $L = N(M_1)$ for a PDA M_1 if and only if $L = T(M_2)$ for some PDA M_2. The equivalence is not true in case of DPDA. Every CFL L is accepted by a PDA M - one state PDA simulates left-most derivation.

3rd March, 2010 (1 hour) - Any regular set L is accepted by a DPDA in final state. The regular language $\{0\}^\star$ is accepted by a DPDA in final state, but is not accepted by a DPDA in empty stack. If $L = N(M_1)$ for a DPDA, then there is a DPDA M_2 so that $L = T(M_2)$. But the reverse is not true. If a language is accepted by a PDA, then it is a CFL - example from the PDA of $\{a^n b^n : n \geqslant 1\}$.

8th March, 2010 (2 hour): 1 hour tutorial +

Pumping Lemma for CFL, $\{a^n b^n c^n : n \geqslant 1\}$ is not a CFL, substitution of a language, the collection of context free language is closed under substitution, finite union, concatenation, Kleene closure and homomorphism, the collection of CFL is not closed under intersection.

9th March, 2010 (1 hour): Closure under substitution, (?)

10th March, 2010 (1 hour): The class of context-free languages is closed under inverse homomorphism. Decision problems of context-free languages - language is empty, language is finite, language is infinite, $x \in L(G)$ - CYK algorithm.

15th March, 2010 (2 hour): 1 hour tutorial +

Intersection of a CFL and a regular language is a CFL, Turing machine - as an acceptor, as a computor and as a enumerator.

16th March, 2010 (1 hour): Turing machine - formal description.

If L is a CFL over the alphabet $\{a\}^\star$, the L is regular.

Proof: If L is finite then L is regular. So we assume that L is infinite and the CFL pumping constant is k. We partition $L = L_1 \cup L_2$, where $L_1 = \{x \in L : |x| < k\}$ and $L_2 = \{x \in L : |x| \geqslant k\}$. L_1 being finite is regular. We shall prove that L_2 is also regular.

Let $w \in L$ and $|w| \geqslant k$, so by the pumping lemma we can write $w = uvxyz$, such that

i. $|vy| > 0$,

ii. $|vxy| \leqslant k$,

iii. for all $i \geqslant 0$, $uv^i xy^i z \in L$

The monoid $\{a\}^\star$ is commutative so (iii) implies that for all $i \geqslant 0$, $uxz(vy)^i \in L$. If $|vy| = p$, then for all $i \geqslant 0$ $uxzvy(vy)^i = w(a^p)^i \in L$. Let $\alpha = k!$, so $(a^\alpha)^m = (a^j)^{\frac{m \times k!}{j}} = (a^j)^{m \times \beta}$, where $\beta = \frac{k!}{j}$. So, $w \in L$ and $|w| \geqslant k$ implies that for all $i \geqslant 0$, $w(a^p)^i \in L$ implies that for all $m \geqslant 0$, $w(a^\alpha)^m \in L$.

We see that each $w \in L$ and $|w| \geqslant k$ is an element of the set $a^{k+i}(a^\alpha)^\star$, for some i, $0 \leqslant i < \alpha$, i.e. $L_2 \subseteq \bigcup_{0\leqslant i<\alpha} a^{k+i}(a^\alpha)^\star$. Let w_i be the least element of the set $L \cap a^{k+i}(a^\alpha)^\star$, so for all $m \geqslant 0$ $w_i(a^\alpha)^m \in L$ and each such element belongs to $a^{k+i}(a^\alpha)^\star$ as $w_i = a^{k+i}(a^\alpha)^{m_i}$. So all these elements starting from w_i can be represented by the *regular expression* $w_i(a^\alpha)^\star$.

We also claim that for some i, $0 \leqslant i < \alpha$, there is no other element in $a^{k+i}(a^\alpha)^\star$ belonging to L. If there is some such element $w_i' = a^{k+i}(a^\alpha)^{l_i}$, then $|l_i| > |m_i|$ as w_i is the least element. Let $|l_i| - |m_i| = d$, so $w_i' = a^{k+i}(a^\alpha)^{m_i+d} = w_i(a^\alpha)^d$ belonging to the chain of w_i.

So we conclude that $w_i(a^\alpha)^\star = L \cap a^{k+i}(a^\alpha)^\star$ and $L_2 = (w_0 + w_1 + \ldots + w_{\alpha-1})(a^\alpha)^\star$ is a regular language.

17th March, 2010 (1 hour): Design of DTM, remembering information in a state - a state may be an n-tuple e.g. (q, a) and (q, b), a tape symbol may be an n-tuple and one component may be modified e.g. (a, b, a) is changed to (a, b, b). Equivalence of singly-infinite tape and doubly-infinite tape Turing machine.

22nd March, 2010 (2 hour): 1 hour tutorial +
Equivalence of singly-infinite DTM and doubly-infinite DTM. Parikh's theorem -

23rd March, 2010 (1 hour): Multi-tape DTM, non-deterministic Turing machine, their equivalence with DTM. A Recursively enumerable and recursive languages.

24th March, 2010 (1 hour): A language is Turing recognisable if and only if it is generated by a unrestricted grammar.

29th March, 2010 (2 hour): 1 hour tutorial +
Continuation of equivalence of Turing machine and unrestricted grammar. The collection of recursive sets is a countable Boolean algebra. Any DTM over the $\Sigma = \{0, 1\}$ can be simulated by a DTM with tape symbols $\Gamma = \{0, 1, \sqcup\}$, where $\sqcup$ is the blank symbol.

30th March, 2010 (1 hour): Encoding of a DTM over $\{0, 1\}$ and with tape symbols $\{0, 1, \sqcup\}$. A DTM may be viewed as a binary numeral of a natural number. Binary representation of every natural number do not encode a DTM. We define such a numeral as a code of a DTM recognising a *null set*.

Let M_1, M_2, $\cdots$ be an enumeration of DTM where M_i is the binary representation of i. Let x_1, x_2, $\cdots$ be the enumerations of strings over $\{0, 1\}$. We define the diagonal language $L_d = \{x_i\colon M_i \,\text{does not accepts}\, x_i\}$.

We claim that L_d is not recursively-enumerable. If it is, then there is a DTM $T_d = T_i$ that recognises L_d. But that leads to contradiction as T_i recognises x_i if and only if T_i does not recognises x_i. So L_d is not Turing recognisable or recursively-enumerable.

There is a *Universal Turing Machine* that take the encoding of a DTM $< M >$ (including itself) an input x to M as input and simulates the behaviour of M on x. Let the language recognised by U is $L_u = \{< M, x > : M \,\text{is a DTM accepts}\, x\}$.

We claim that $\overline{L_d} = \{x_i\colon M_i \,\text{accepts}\, x_i\}$ is *recursively enumerable* but not *recursive*. It is not recursive as that makes L_d recursive. But we know that L_d is not even recursively enumerable. Following machine recognises $\overline{L_d}$.

$\overline{M_d}$:
Input: x

1. Enumerate strings over $\{0, 1\}$, x_1, x_2, $\cdots$ and compare each enumerated string with x. Stop, if they are equal. Let $x = x_j$.
2. Consider the binary representation of j, $< j >$. If it is not a valid encoding of DTM, reject x. Any invalid binary string represents a DTM whose language is empty, so it does not accept $x = x_j$.
3. If $< j >$ is a valid machine, run the universal machine U on input $< M_j, x_j >$.
4. If U reaches the final state i.e. M_j reaches the final state on x_j, then accept x.

5. If U reaches a non-final state and halts, then let $\overline{M_d}$ also halt at a non-final state and reject x.

6. If the simulation goes in an infinite loop, $\overline{M_d}$ also does the same.

It is clear that the language recognised by $\overline{M_d}$ is $\overline{L_d}$.

The language L_u of a Universal TM is certainly recursively-enumerable. But it cannot be recursive as a decider for L_u makes a decider for $\overline{L_d}$ (in the construction of $\overline{M_d}$, we shall replace the U by this decider) and that will make L_d also recursive. But we have already proved otherwise. This is called problem reduction - **we reduce the decision-problem of $\boldsymbol{\overline{L_d}}$ to the decision-problem of $\boldsymbol{L_u}$.** As $\overline{L_d}$ is known to be undecidable, then so is L_u.

Again the language $\overline{L_u} = \{ < M, x >: M \text{ does not accept } x\}$ cannot be recursively-enumerable as that will make both L_u and $\overline{L_u}$ recursive. So we have two languages and their complements - $\overline{L_d}$ and L_u - recursively-enumerable, and L_d and $\overline{L_u}$ are not even recursively-enumerable.

Problem reduction is a method of converting a decision-problem of a language A, to the decision-problem of a language B, so that a solution to the decision-problem of B results a solution to the decision-problem of A. As an example consider the construction of the previous machine $\overline{M_d}$. In a sense it reduces the decision-problem of $\overline{L_d}$ (A) to the decision-problem of L_u (B).

31$^{\text{st}}$ March, 2010 (1 hour):