COMS3261:
Computer Science Theory

Fall 2013

Mihalis Yannakakis

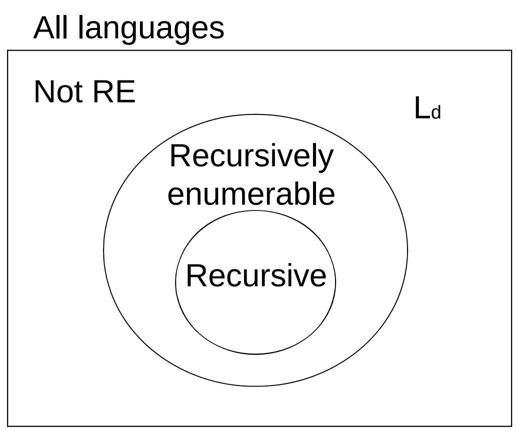
Lecture 21, 11/20/13

Recursive vs RE languages (reminder)

- Can assume wlog that TM halts when it accepts
- But TM may reject either by running forever or by halting at a nonaccepting state.
- Recursive language L: ∃ TM M that halts on every input and L=L(M) (M accepts w ⇔ w ∈L) (M =algorithm for L)
- Recursively enumerable language L: L=L(M) for some TM M (M may not halt on some inputs not in L)
- Decidable problem ⇔ recursive language
- Undecidable problem: Not a recursive language; could be recursively enumerable

- Diagonalization language L_d = { <M> | TM M rejects <M>} is not r.e.

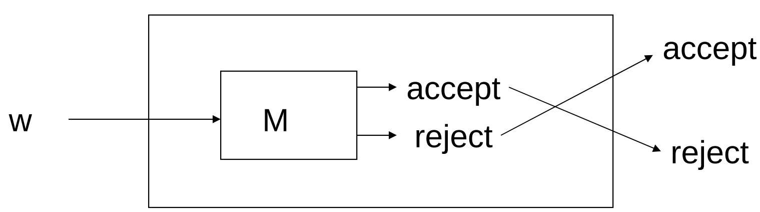
Relationship



Recursive languages & complementation

- L recursive $\Rightarrow$ L^c recursive

  Proof: Take halting TM M for L. When M halts, if rejecting state then go to a new accepting state, else not $\rightarrow$ TM M' always halts and accepts L^c

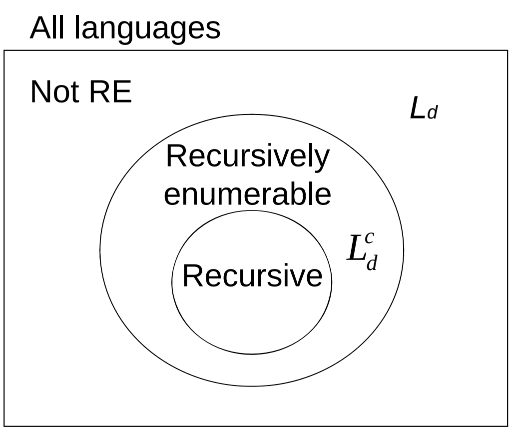


- Not true for recursively enumerable languages:

  Problem: rejected inputs where M runs forever

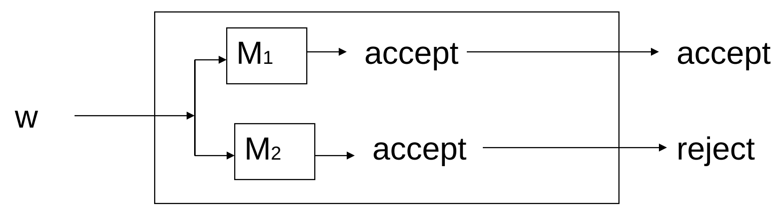
Example

The complement L_d^c of L_d is recursively enumerable but not recursive. If it was recursive, then its complement L_d would also be recursive.



Recursive languages & complementation

- Theorem: If both L and L^c are recursively enumerable, then L is recursive (and so is L^c)
- Proof: Let M_1 be a TM for L, M_2 a TM for L^c
- Combine: run both in parallel. One of them will halt and accept.
- For example, if they are 1-tape TMs, then combined TM M has 1 tape for each, state of M keeps track of both states.



Universal Language L_u

- $L_u = \{ \langle M,w \rangle \in \{0,1\}^* \mid w \in L(M) \}$
- <M,w> encoding of a TM M, and an input string w for M. Can separate them with 111 (<M> does not have three 1's in row)

Theorem: L_u is recursively enumerable

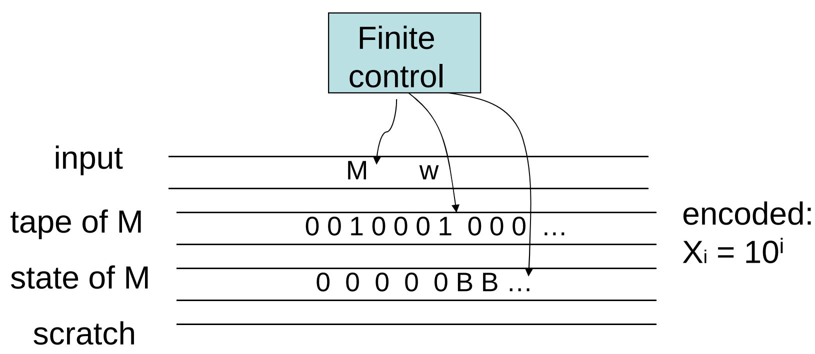
i.e. there is a TM whose language is L_u

Universal TM: Takes as input a TM M (i.e. a program) and an input w (the data), and determines if M accepts w.

= General purpose computer

Big conceptual breakthrough of Turing

Universal TM

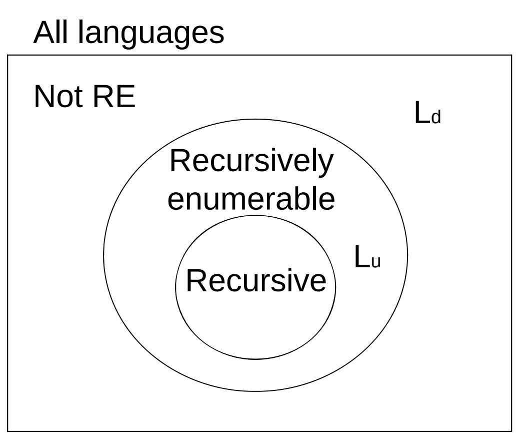


- Assume wlog M is a 1-tape TM (holds also for multitape, but U has to do the encoding of the many tapes into 1)
- Copy w to `tape of M', initialize state tape, and simulate every move of M: compare current symbol & state with code of M to determine transition

Simulation of Universal TM

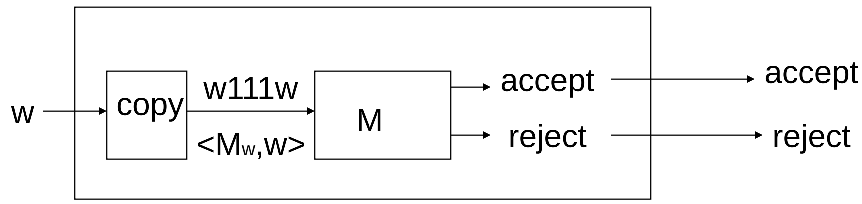
- Initialization: 1. examine input to test if legal TM code; if not halt and reject.
- If yes, then copy w to tape 2 encoding the input symbols (0→10,1→100). Leave B's but when head moves to B, replace with 1000 (code for B). Put 0 (code(q_0)) on tape 3 and position heads
- Loop (move simulation). Search code of M on input tape to find a transition that matches current state and tape symbol. If none then halt.
- If yes, then update state of M and tape symbol, and shift over accordingly contents of tape of M, using scratch tape.
- If new state of M accepting then halt and accept, else repeat.

Relationship



Undecidability of L_u

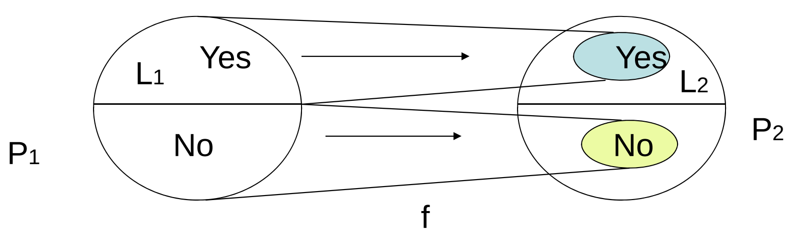
- Theorem: L_u^c is not recursively enumerable ($\Rightarrow$ L_u is not recursive).
- Proof: Reduction from L_d: Show can use a TM M for L_u^c to construct a TM M' for L_d.



$w \in L_d \Leftrightarrow w \notin L(\text{TM with code } w) \Leftrightarrow w111w \notin L_u \Leftrightarrow w111w \in L_u^c$

Reductions

- Many-one (or mapping) Reduction $\leq_m$ from one language L_1 (decision problem P_1) to another language L_2 (problem P_2)



- Mapping f maps every instance x of problem P_1 to an instance f(x) of problem P_2
- f can be computed by a TM, i.e. TM with input x, halts with f(x) on its tape
- $x \in L1 \Leftrightarrow f(x) \in L2$

Properties of Reductions

- Suppose $L1 \leq_m L2$:
- Then: If L2 is r.e. then also L1 is r.e.
- $\Rightarrow$ If L1 is not r.e. then L2 is not r.e.
- If L1 is not recursive (P1 is undecidable) then L2 is not recursive (P2 is undecidable)
- (L_d^c is r.e. but not recursive, like L_u)
- Complement: If $L1 \leq_m L2$ then $L1^c \leq_m L2^c$ by same reduction
- Transitivity: If $L1 \leq_m L2$ and $L2 \leq_m L3$ then $L1 \leq_m L3$

- Things to watch – common mistakes:
- Reducing in the wrong direction. Want from problem known to be undecidable or not r.e. to problem we want to show
- Confusing a language and its complement

Turing Reductions

- There is another reduction, Turing reduction = algorithm for P1 that uses a subroutine for P2, i.e., to answer one instance of P1, may use many calls to P2 on various instances, or maybe one call and sometimes reverse the decision.

- Then P1 undecidable $\Rightarrow$ P2 undecidable, but could be not r.e. or complement not r.e. or both. (i.e. L1 not r.e. does not necessarily imply L2 not r.e.)