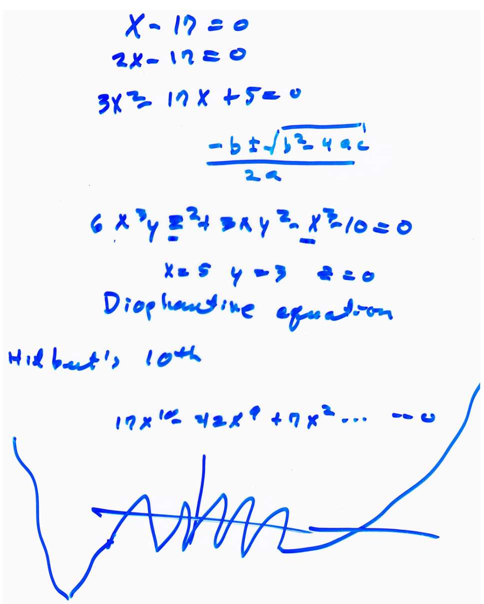
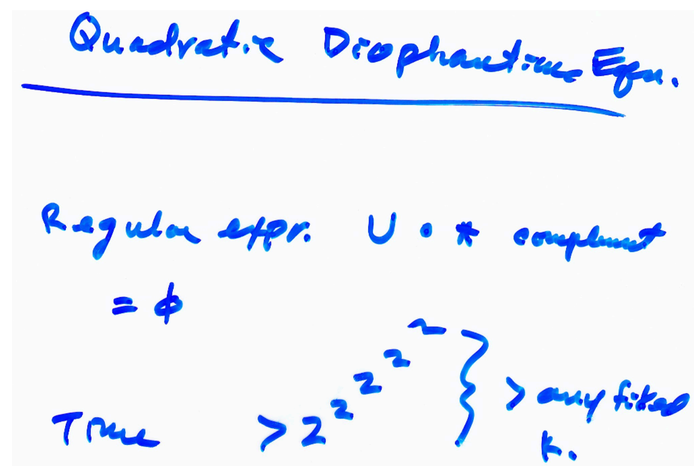
CSE 431

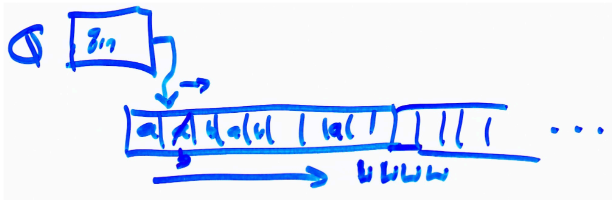
Larry Ruzzo, Spring 2010

http://www.cs.washington.edu/431

Initial Reading Assignment: Sipser Chapter 3



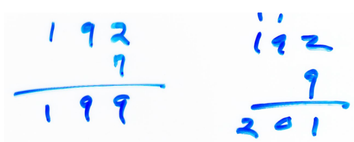




Lecture 2

Algorithms

"An *algorithm* is a finite, precise set of instructions for performing a computation"



"The Division Algorithm": $\forall$ a $\in \mathbb{Z}$, d $\in \mathbb{Z}^+$,
$\exists$ unique q, r such that $0 \leq r < d$ and $a = qr+d$

Deal with finite set of discrete objects
Finite list of instructions
Each
- simple
- unambiguous
- from a finite set of possibilities
- "clearly" solvable in finite time by "simple" agent

Overall process will finish in a finite amount of time.

Examples

$x+1$ String concatenation
$x+y$
$x*y$

Defn $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$

Q: finite state set

Σ: finite input alphabet set ; "blank" $\sqcup \notin \Sigma$

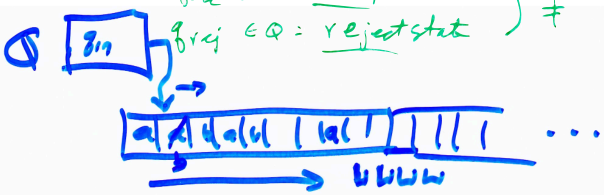
Γ: finite tape alphabet . $\Sigma \cup \{\sqcup\} \subseteq \Gamma$.

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

$q_0 \in Q$: start state

$q_{acc} \in Q$: accept state

$q_{rej} \in Q$: reject state) $\neq$



Example

$L = \{ w \# w \mid w \in \{0,1\}^* \}$

1. check that there's a single #
2. read, remember & cross off leftmost unscrossed letter
3. scan to # & compare next uncrossed letter
4. If ok, cross it off
5. repeat

Lecture 3

Defn $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$

Q: finite state set

Σ: finite input alphabet set ; "blank" $\sqcup \notin \Sigma$

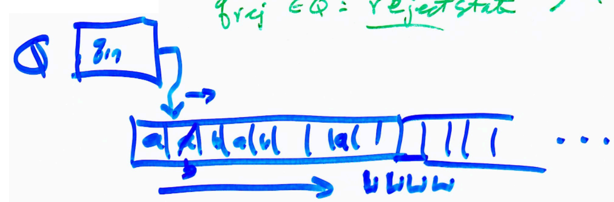
Γ: finite tape alphabet . $\Sigma \cup \{\sqcup\} \subseteq \Gamma$.

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

$q_0 \in Q$: start state

$q_{acc} \in Q$: accept state

$q_{rej} \in Q$: reject state) $\neq$



By definition, no transitions *out* of q_{acc}, q_{rej};

M *halts* if (and only if) it reaches either

M *loops* if it never halts ("loop" might suggest "simple", but non-halting computations may of course be arbitrarily complex)

M *accepts* if it reaches q_{acc},

M *rejects* by halting in q_{rej} or by looping

The language *recognized* by M:
L(M) = { w ∈ Σ* | M accepts w }

L is *Turing recognizable* if ∃TM M s.t. L = L(M)

L is *Turing decidable* if, furthermore, M halts on all inputs

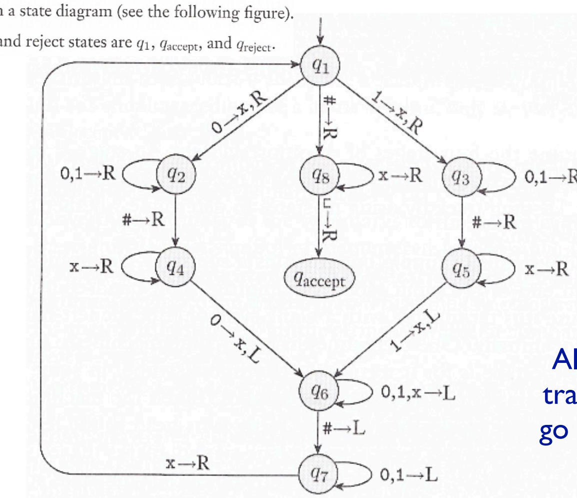
A key distinction!

Example

$L = \{ w\#w \mid w \in \{0,1\}^* \}$

1. check that there's a single #
2. read, remember & cross off left most, uncrossed letter
3. scan to # & compare next, uncrossed letter
4. If ok, cross it off
5. repeat

- $Q = \{q_1, \ldots, q_{14}, q_{\text{accept}}, q_{\text{reject}}\}$,
- $\Sigma = \{0,1,\#\}$, and $\Gamma = \{0,1,\#,x,\sqcup\}$.
- We describe δ with a state diagram (see the following figure).
- The start, accept, and reject states are q_1, q_{accept}, and q_{reject}.



All other transitions go to q_{reject}

$L = \{ w\#w \mid w \in \{0,1\}^* \}$

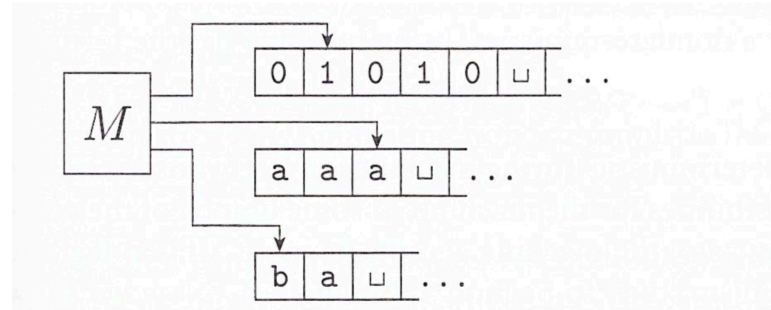
Church-Turing Thesis

TM's formally capture the intuitive notion of "algorithmically solvable"

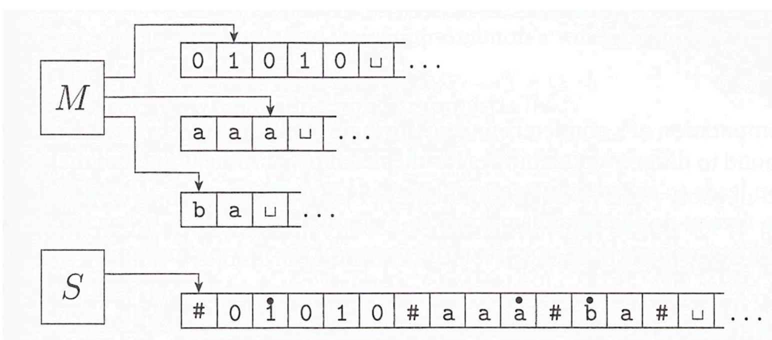
Not provable, since "intuitive" is necessarily fuzzy.

But, give support for it by showing that
(a) other intuitively appealing (but formally defined) models are precisely equivalent (rest of lecture), and
(b) models that are provably different are unappealing, either because they are too weak (e.g., DFA's) or too powerful (e.g., a computer with a "solve-the-halting-problem" instruction).

Multi-tape Turing Machines

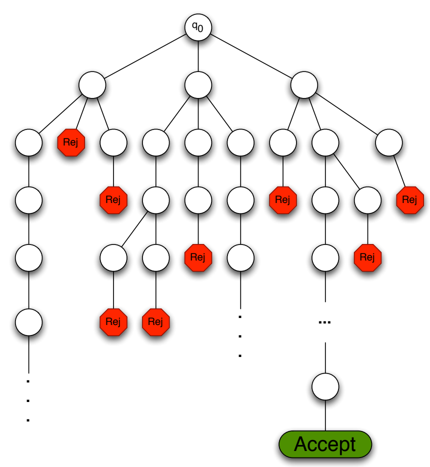


$$\delta \colon Q \times \Gamma^k \longrightarrow Q \times \Gamma^k \times \{\mathrm{L}, \mathrm{R}, \mathrm{S}\}^k$$



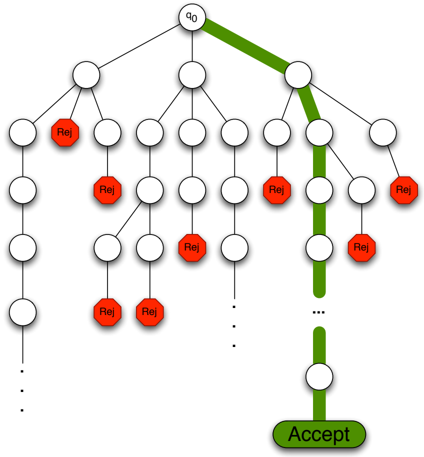
Nondeterministic Turing Machines

$$\delta \colon Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L,R\})$$



Nondeterministic Turing Machines

$\delta: Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L,R\})$



Accept if *any* path leads to q_{accept}; reject otherwise, (i.e., *all* halting paths lead to q_{reject})

Lecture 4

Announcements

Late policy

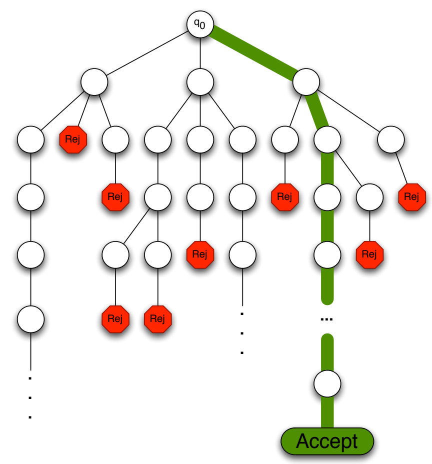
eTurnin

Office hours M 2:30, W 12:30, Th 5:00

Midterm Fri 5/7, probably

Nondeterministic Turing Machines

$\delta: Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L,R\})$

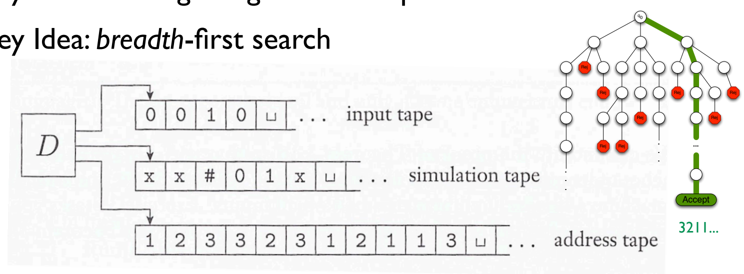


Accept if *any* path leads to q_{accept}; reject otherwise, (i.e., *all* halting paths lead to q_{reject})

Simulating an NTM

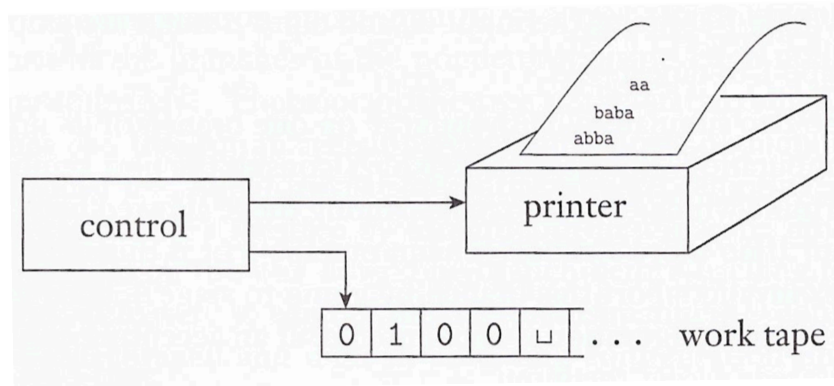
Key issue: avoid getting lost on ∞ path

Key Idea: *breadth*-first search



tree arity ≤ |Q| x | Γ| x |{L,R}| (3 in example)

A TM "Enumerator"



L Turing recognizable iff a TM enumerates it

(⇐): Run enumerator, compare each "output" to input; accept if they match (reject by not halting if input never appears)

(⇒): The "obvious" idea: enumerate Σ^*, run the recognizer on each, output those that are accepted.

[Oops, doesn't work... may not halt...]

L Turing recognizable iff a TM enumerates it

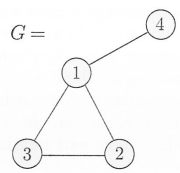
(⇒): A better idea–"dovetailing":

For i = 0, 1, 2, 3, ... :

> At stage i, run the recognizer for i steps on each of the first i strings in Σ^*, output any that are accepted.

Encoding things

$G =$



$\langle G \rangle =$

(1,2,3,4)((1,2),(2,3),(3,1),(1,4))

Σ = ?

CFG G = (V, Σ, R, S) ; <G> = ((S,A,B,...),(a,b,...), (S→aA, S→b, A→cAb, ...),S)

or <G> = ((A_0, A_1, ...),(a_0, a_1, ...), (A_0 → a_0 A_1, A_0 → a_1, A_1 → a_2 A_1 a_1 , ...), A_0)

DFA D = (Q, Σ, δ, q_0, F); <D> = (...)

TM M = (Q, Σ, Γ, δ, q_0, q_a, q_r); <M> = (...)

...

Decidability

Recall: L *decidable* means there is a TM recognizing L *that always halts*.

Example:

"The acceptance problem for DFAs"

A_{DFA} = { <D,w> | D is a DFA & w $\in$ L(D) }

Some Decidable Languages

The following are decidable:

A_{DFA} = { <D,w> | D is a DFA & w $\in$ L(D) }

pf: simulate D on w

A_{NFA} = { <N,w> | N is an NFA & w $\in$ L(N) }

pf: convert N to a DFA, then use previous as a subroutine

A_{REX} = { <R,w> | R is a regular expr & w $\in$ L(R) }

pf: convert R to an NFA, then use previous as a subroutine

$EMPTY_{DFA}$ = {<D> | D is a DFA and L(D) = $\varnothing$ }

pf: is there no path from start state to any final state?

EQ_{DFA} = { <A,B> | A & B are DFAs s.t. L(A)= L(B) }

pf: equal iff L(A)$\oplus$L(B) = $\varnothing$, and $x \oplus y = (x \cap y^c) \cup (x^c \cap y)$, and regular sets are closed under $\cup$, $\cap$, complement

A_{CFG} = { <G,w> | ... }

pf: see book

$EMPTY_{CFG}$ = { <G> | ... }

pf: see book

EQ_{CFG} = { <A,B> | A & B are CFGs s.t. L(A) = L(B) }

This is *NOT* decidable

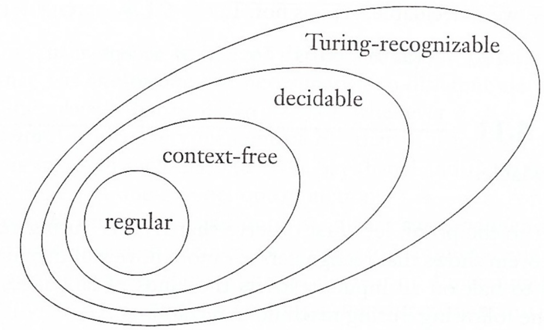


FIGURE 4.10
The relationship among classes of languages

Lecture 5

The Acceptance Problem for TMs

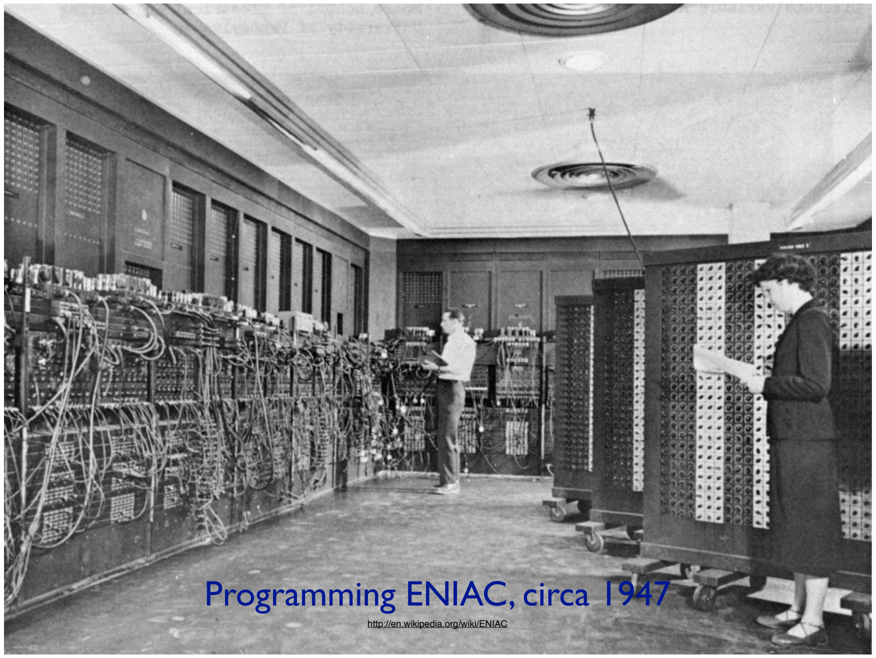
A_{TM} = { <M,w> | M is a TM & $w \in L(M)$ }

Theorem: A_{TM} is Turing recognizable

Pf: It is recognized by a TM U that, on input <M,w>, simulates M on w step by step. U accepts iff M does. □

U is called a *Universal Turing Machine*
(Ancestor of the stored-program computer)

Note that U is a recognizer, not a decider.



Cardinality

Two sets have equal cardinality if there is a bijection between them

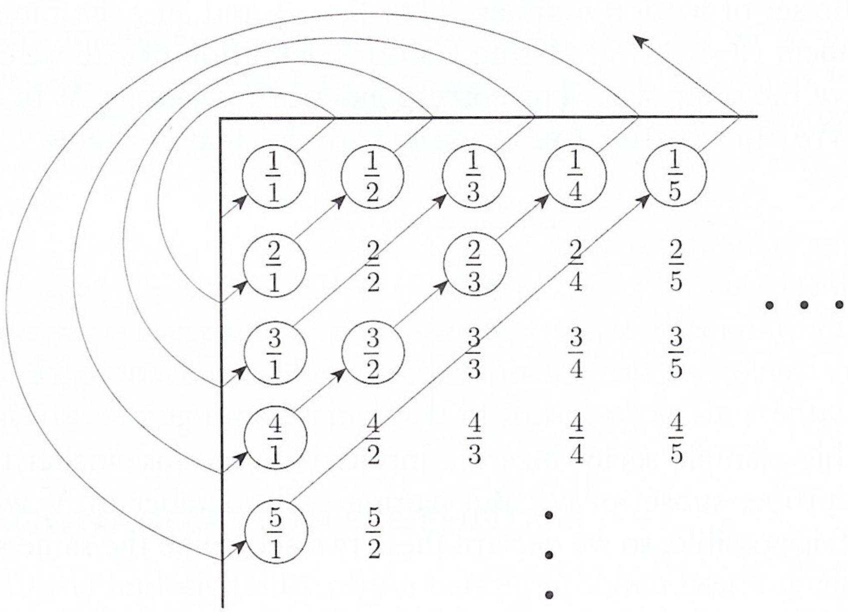
A set is *countable* if it is finite or has the same cardinality as the natural numbers

Examples:

Σ^* is countable (think of strings as base-$|\Sigma|$ numerals)

Even natural numbers are countable: $f(n) = 2n$

The Rationals are countable



More cardinality facts

If $f: A \rightarrow B$ in an injective function ("1-1", but not necessarily "onto"), then

$|A| \leq |B|$

(Intuitive: f is a bijection from A to its range, which is a subset of B, & B can't be smaller than a subset of itself.)

Theorem (Cantor-Schroeder-Bernstein):

If $|A| \leq |B|$ and $|B| \leq |A|$ then $|A| = |B|$

The Reals are Uncountable

Suppose they were

List them in order

Define X so that its i^{th} digit $\neq$ i^{th} digit of i^{th} real

Then X is *not in the list*

Contradiction

A detail: avoid .000..., .9999... in X

	int	1	2	3	3	5	
1	0.	0	0	0	0	0	
2	3.	1	4	1	5	9	
3	0.	3	3	3	3	3	
4	0.	5	0	0	0	0	...
5	2.	7	1	8	2	8	
6	41.	9	9	9	9	9	
			⋮				⋱
X	1.	2	4	1	3	8	...

Number of Languages in Σ^* is Uncountable

Suppose they were

List them in order

Define L so that
$w_i \in L \Leftrightarrow w_i \notin L_i$

Then L is *not in the list*

Contradiction

	w_1	w_2	w_3	w_4	w_5	w_6	
L_1	0	0	0	0	0	0	
L_2	1	1	1	1	1	1	
L_3	0	1	0	1	0	1	
L_4	0	1	0	0	0	0	...
L_5	1	1	1	0	0	0	
L_6	1	1	1	1	0	1	
			⋮				⋱
L	1	0	1	1	1	0	...

"Most" languages are neither Turing recognizable nor Turing decidable

Pf:

“< >” maps TMs into Σ^*, a countable set, so the set of TMs, and hence of Turing recognizable languages is also countable; Turing decidable is a subset of Turing recognizable, so also countable. But by the previous result, the set of all languages is *un*countable.

A *specific* non-Turing-recognizable language

Let M_i be the TM encoded by w_i, i.e. $<M_i> = w_i$

(M_i = some default machine, if w_i is an illegal code.)

i, j entry tells whether M_i accepts w_j

Then L_D is *not recognized by any TM*

	w_1	w_2	w_3	w_4	w_5	w_6	
$<M_1>$	0	0	0	0	0	0	
$<M_2>$	1	1	1	1	1	1	
$<M_3>$	0	1	0	1	0	1	
$<M_4>$	0	1	0	0	0	0	...
$<M_5>$	1	1	1	0	0	0	
$<M_6>$	0	1	0	0	0	1	
			⋮				⋱
L_D	1	0	1	1	1	0	...

Theorem: The class of Turing recognizable languages is *not* closed under complementation.

Proof:

The *complement* of D, *is* Turing recognizable:

On input w_i, run $<M_i>$ on w_i (= $<M_i>$); accept if it does. E.g. use a universal TM on input $<M_i,<M_i>>$

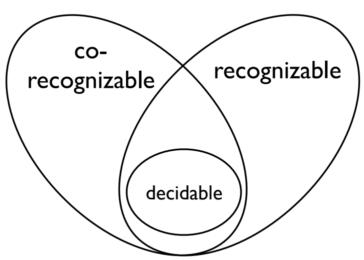
E.g., in previous example, D^c might be $L(M_6)$

Theorem: The class of Turing decidable languages *is* closed under complementation.

Proof:

Flip q_{accept}, q_{reject}

Decidable $\subsetneq$ Recognizable



Lecture 6

The Acceptance Problem for TMs

A_{TM} = { <M,w> | M is a TM & $w \in L(M)$ }

Theorem: A_{TM} is Turing recognizable

Pf: It is recognized by a TM U that, on input <M,w>, simulates M on w step by step. U accepts iff M does. □

U is called a *Universal Turing Machine*
(Ancestor of the stored-program computer)

Note that U is a recognizer, not a decider.

A_{TM} is Undecidable

A_{TM} = { <M,w> | M is a TM & $w \in L(M)$ }

Suppose it's decidable, say by TM H. Build a new TM D:

"on input <M> (a TM), run H on <M,<M>>; when it halts, halt & do the opposite, i.e. accept if H rejects and vice versa"

D accepts <M> iff H rejects <M,<M>> (by construction)
iff M rejects <M> (H recognizes A_{TM})

D accepts <D> iff D rejects <D> (special case)

Contradiction!

A *specific* non-Turing-recognizable language

Note: The above TM D, if it existed, would recognize exactly the language L_D defined in this diagonalization proof (which we already know is not recognizable)

Let M_i be the TM encoded by w_i, i.e. $\langle M_i \rangle = w_i$

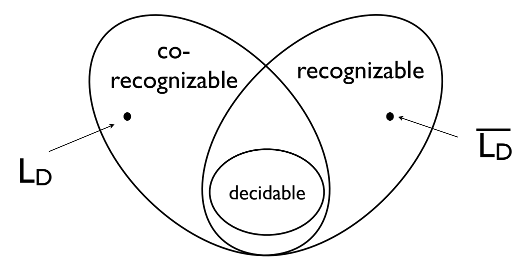
(M_i = some default machine, if w_i is an illegal code.)

i, j entry tells whether M_i accepts w_j

Then L_D is *not recognized by any TM*

	w_1	w_2	w_3	w_4	w_5	w_6	
$\langle M_1 \rangle$	0	0	0	0	0	0	
$\langle M_2 \rangle$	1	1	1	1	1	1	
$\langle M_3 \rangle$	0	1	0	1	0	1	
$\langle M_4 \rangle$	0	1	0	0	0	0	…
$\langle M_5 \rangle$	1	1	1	0	0	0	
$\langle M_6 \rangle$	0	1	0	0	0	1	
			⋮				⋱
L_D	1	0	1	1	1	0	…

Decidable $\subsetneq$ Recognizable

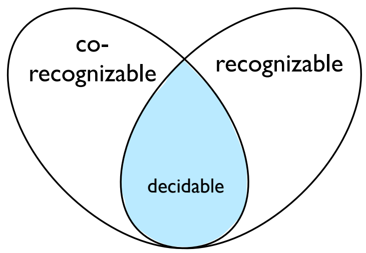


Decidable = Rec ∩ co-Rec

L decidable iff both L & L^c are recognizable

Pf:
(⇐) on any given input, dovetail a recognizer for L with one for L^c; one or the other must halt & accept, so you can halt & accept/reject appropriately.

(⇒): from last lecture, decidable languages are closed under complement (flip acc/rej)



Reduction

"A is reducible to B" means I could solve A *if* I had a subroutine for B

Ex:

Finding the max element in a list is reducible to sorting

pf: sort the list in increasing order, take the last element

(A big hammer for a small problem, but never mind...)

The Halting Problem

$HALT_{TM}$ = { <M,w> | TM M halts on input w }

Theorem: The halting problem is undecidable

Proof:

A = A_{TM}, B = $HALT_{TM}$ Suppose I can reduce A to B. We already know A is undecidable, so must be that B is, too.

Suppose TM R decides $HALT_{TM}$. Consider S:

On input <M,w>, run R on it. If it rejects, halt & reject; if it accepts, run M on w; accept/reject as it does.

Then S decides A_{TM}, which is impossible. R can't exist.

Lecture 7

Reduction

"A is reducible to B" means I could solve A *if* I had a subroutine for B

Ex:

Finding the max element in a list is reducible to sorting

pf: sort the list in increasing order, take the last element

(A big hammer for a small problem, but never mind...)

The Halting Problem

$HALT_{TM}$ = { <M,w> | TM M halts on input w }

Theorem: The halting problem is undecidable

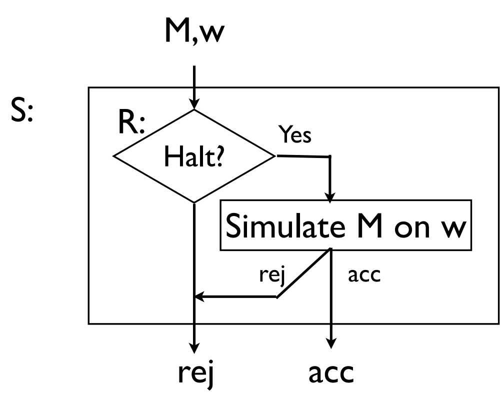
Proof:

A = A_{TM}, B = $HALT_{TM}$ Suppose I can reduce A to B. We already know A is undecidable, so must be that B is, too.

Suppose TM R decides $HALT_{TM}$. Consider S:

> On input <M,w>, run R on it. If it rejects, halt & reject; if it accepts, run M on w; accept/reject as it does.

Then S decides A_{TM}, which is impossible. R can't exist.



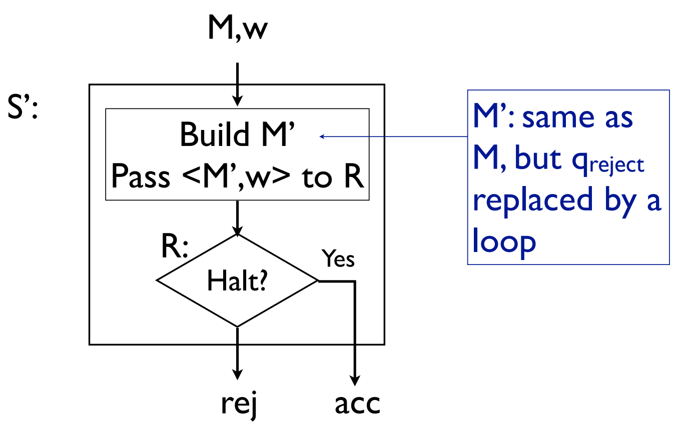
Another Way

Rather than running R on <M,w>, and manipulating that answer, manipulate the input to build a new M' so that R's answer about <M',w> *directly* answers the question of interest.

Specifically, build M' as a clone of M, but modified so that if M halts-and-rejects, M' instead rejects by looping.

Then halt/not-halt for M' == accept/reject for M

Again, this reduces A_{TM} to $HALT_{TM}$



Reduction

Notation (not in book, but common):

$A \leq_T B$ means "A is Turing Reducible to B"

I.e., if I had a TM deciding B, I could use it as a subroutine to solve A

Facts:

$A \leq_T B$ & B decidable implies A decidable (definition)

$A \leq_T B$ & A *un*decidable implies B undecidable (contrapositive)

$A \leq_T B$ & $B \leq_T C$ implies $A \leq_T C$

$EMPTY_{TM}$ is undecidable

$EMPTY_{TM}$ = { <M> | M is a TM s.t. L(M) = ∅ }

Pf: To show: $A_{TM} \leq_T EMPTY_{TM}$

On input <M,w> build M' :

M' on input x:
1. erase x
2. write w
3. run M on w
4. if M accepts w, then accept x
5. otherwise, reject x

Do *not* run M or M'. (That whole "halting thing" means we might not learn much if we did.) But note that L(M') is/is not empty exactly when M does not/does accept w, so knowing whether L(M') = ∅ answers whether <M,w> is in A_{TM}. And our hypothetical "$EMPTY_{TM}$" subroutine applied to M' tells us just that. I.e., $A_{TM} \leq_T EMPTY_{TM}$

$$L(M') = \begin{cases} \Sigma^*, & \text{if M accepts w} \\ \varnothing, & \text{if M rejects w} \end{cases}$$

$REGULAR_{TM}$ is undecidable

$REGULAR_{TM}$ = { <M> | M is a TM s.t. L(M) is regular }

Pf: To show: $A_{TM} \leq_T REGULAR_{TM}$

On input <M,w> build M' :

M' on input x:
1. if $x \in \{0^n 1^n | n \geq 0\}$, accept x
2. otherwise, erase x
3. write w
4. run M on w
5. if M accepts w, then accept x
6. otherwise, reject x

Do *not* run M or M'. (That whole "halting thing" ...) But note that L(M') is/is not regular exactly when M does/does not accept w, so knowing whether L(M') is regular answers whether <M,w> is in A_{TM}. The hypothetical "$REGULAR_{TM}$" subroutine applied to M' tells us just that. I.e., $A_{TM} \leq_T EMPTY_{TM}$

$$L(M') = \begin{cases} \Sigma^*, & \text{if M accepts w} \\ \{0^n 1^n | n \geq 0\}, & \text{otherwise} \end{cases}$$

Lecture 8

Announcements

re HW#1, Aeron says "If I made a comment, even if I didn't take off points *this* time, people should pay attention because I will take off points for the same mistake in the future..."

EQ_{TM} is undecidable

$EQ_{TM} = \{ <M_1, M_2> \mid M_i \text{ are TMs s.t. } L(M_1) = L(M_2)\}$

EQ_{TM} is undecidable

$EQ_{TM} = \{ <M_1, M_2> \mid M_i \text{ are TMs s.t. } L(M_1) = L(M_2)\}$

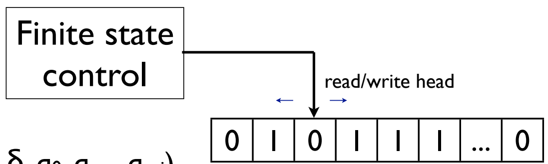
Pf: Will show $EMPTY_{TM} \leq_T EQ_{TM}$

Suppose EQ_{TM} were decidable. Let $M_\varnothing$ be a TM that accepts nothing, say one whose start state = q_{reject}. Consider the TM E that, given $<M>$, builds $<M, M_\varnothing>$, then calls the hypothetical subroutine for EQ_{TM} on it, accepting/rejecting as it does. Now, $<M, M_\varnothing> \in EQ_{TM}$ if and only if M accepts $\varnothing$, so, E decides whether $M \in EMPTY_{TM}$, which we know to be impossible. Contradiction

Linear Bounded Automata

Like a (1-tape) TM, but tape only long enough for input

(head stays put if try to move off either end of tape)



$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$

$L(M) = \{ x \in \Sigma^* \mid M \text{ accepts } x \}$

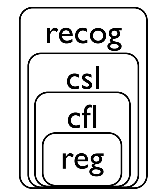
An Aside: The Chomsky Hierarchy

TM = phrase structure grammars $\alpha A \beta \to \alpha \gamma \beta$

LBA = context-sensitive grammars $\alpha A \beta \to \alpha \gamma \beta,\ \gamma \neq \varepsilon$

PDA = context-free grammars $A \to \gamma$

DFA = regular grammars $A \to abcB$



A_{LBA} is decidable

$A_{LBA} = \{ \langle M, w \rangle \mid M \text{ is an LBA and } w \in L(M) \}$

Key fact: the number of distinct configurations of an LBA on any input of length n is *bounded*, namely

$$\leq n\,|Q|\,|\Gamma|^n$$

If M runs for more than that many steps, it is looping

Decision procedure for A_{LBA}:

Simulate M on w and count steps; if it halts and accepts/rejects, do the same; if it exceeds that time bound, halt and reject (it's looping).

$EMPTY_{LBA}$ is undecidable

Why is this hard, when the acceptance problem is not?

Loosely, it's about infinitely many inputs, not just one

Can we exploit that, say to decide A_{TM}?

An idea. An LBA is a TM, so can it simulate M on w?

Only if M doesn't use too much tape.

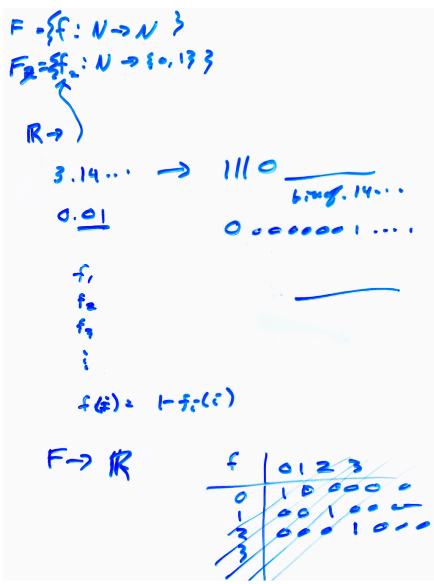
What about simulating M on w############ ?

Given M, build LBA M’ that, on input w # # # # … #, simulates M on w, treating # as a blank. If M halts, do the same. If M tries to move off the right end of the tape, reject.

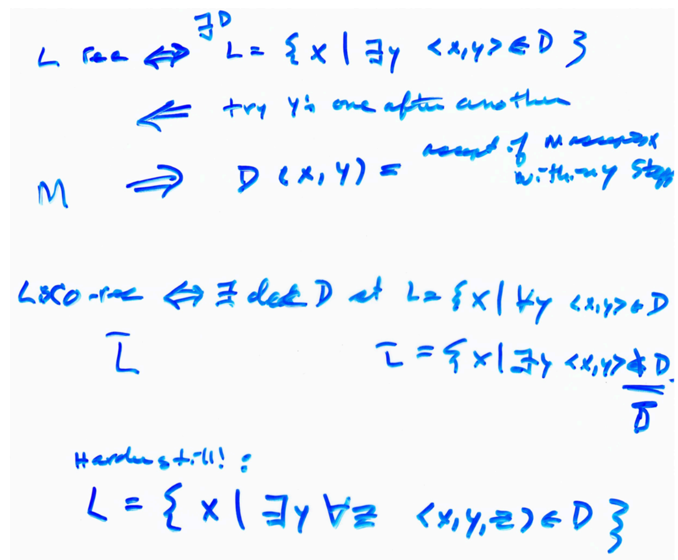
$L(M') = \{ w\#^k \mid M \text{ accepts } w \text{ using} \leq | w\#^k | \text{ tape cells} \}$

Key point:

- if M rejects w, M’ rejects $w\#^k$ for all k, $\therefore L(M') = \varnothing$
- if M accepts w, some k will be big enough, $\therefore L(M') \neq \varnothing$



Lecture 9



rec. set of Deciders

	ε	a	b	ab	...
M_1	1	1	1	1	
M_2	0	0	0	0	
M_3					
L	0	1	...		

L decidable? Yes: on i^{th} input, run enumerator until i^{th} TM is output, then run it on i, then do opposite.

Lecture 10

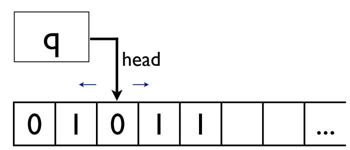
$\text{EMPTY}_{\text{LBA}}$ is undecidable

An alternate proof, using a new technique –

Computation histories

Computation Histories

Configuration:
state, head, tape



Encoding Configs:

0 1 q 0 1 1

A string in $\Gamma^* Q \Gamma^*$ (trailing blanks optional)

Accepting (Rejecting) *History*: $C_1, C_2, \ldots, C_n$ s.t.

1. C_1 is M's initial configuration
2. C_n is an accepting (rejecting, resp.) config, and
3. For each $1 \leq i < n$, C_i moves to C_{i+1} in one step

Checking Histories

Many proofs require *checking* that a string, say

$\# C_1 \# C_2 \# \ldots \# C_n \#$ in $(\{\#\} \cup Q \cup \Gamma)^*$

is/is not an accepting history:

1. C_1 is M's initial configuration:

   $C_1 \in q_0 \Sigma^*$

2. C_n is an accepting config:

   it contains q_{accept}

3. For each $1 \leq i < n$, C_i moves to C_{i+1} in one step

...

"C_i moves to C_{i+1} in one step of M"

$\#a_1a_2\ldots\boxed{a_k p a_{k+1}}a_{k+2}\ldots a_n\#b_1b_2\ldots\boxed{b_j q b_{j+1}}b_{j+2}\ldots b_m\#$

No change:

$a_i = b_i \in \Gamma$

$p, q \in Q$,

$j = k \pm 1$

$n = m$

Except for adjustments, all near the head, reflecting the move:

$\delta(p, a_{k+1}) = (q, b_{k+1}, L/R)$,

$j = k+1$ if R else $\max(k-1,0)$

and injecting blanks on the right as needed:

if $n = k$, then "a_{k+1}" = blank

$m = n+1$, ...

Aside: one reason TM's have been so useful for computation theory is that they make questions like this very simple; "config" and "move" are much messier for "real" computers.

$A_{TM} \leq_T EMPTY_{LBA}$

Given $\langle M, w\rangle$, build an LBA $L_{M,w}$ that recognizes

$AH_{M,w} = \{ x \mid x = \# C_1 \# C_2 \# \ldots \# C_n \#$, an *A*ccepting computation *H*istory of M on w $\}$

Then pass $\langle L_{M,w}\rangle$ to the hypothetical subr for $EMPTY_{LBA}$

Specifically, $L_{M,w}$ operates by checking that:

1. Its input is of the form $\# C_1 \# C_2 \# \ldots \# C_n \#$
2. C_1 is the initial config of M on w
3. C_n has M's accept state, and
4. For each $1 \leq i < n$, C_i moves to C_{i+1} in one step of M
   (ziz-zag across adjacent pairs, checking as on prev slide)

Correctness

$L(L_{M,w}) = AH_{M,w} = \{ x \mid x = \# C_1 \# C_2 \# \ldots \# C_n \#$, an accepting computation history of M on w $\}$

Empty if M rejects w – no such x

Non-empty if M accepts w – there is one such history

So, "M accepts w" is equivalent to (non-)emptyness of $AH_{M,w}$

$\therefore A_{TM} \leq_T EMPTY_{LBA}$ QED

Notes

Similar ideas can be used to give reductions like

$$A_{TM} \leq_T EMPTY_X$$

for any machine or language class X expressive enough that we can easily, given M & w, represent $AH_{M,w}$ in X

A nice thing about histories is that they are so transparent that this is easy, even for more restricted models than LBA's

(One example in homework; another below)

ALL_{CFL} is Undecidable

$$ALL_{CFG} = \{ \langle G \rangle \mid G \text{ is a CFG with } L(G) = \Sigma^* \}$$

A variant on the above proof, but instead of using $AH_{M,w}$, (the set of accepting histories of M on w), we use its *complement:*

$$NH_{M,w} = \{ x \mid x \text{ is } \textit{not} \text{ an accepting computation history}^* \text{ of M on w } \}$$

* and change the representation of a history so that alternate configs are reversed:

$$\# C_1 \# C_2^R \# C_3 \# C_4^R \#\ldots\# C_n^{(R?)} \#$$

$A_{TM} \leq_T ALL_{CFG}$

Given M, w, build a PDA P that, on input x, accepts if x does *not* start and end with #; otherwise, let

$$x = \# C_1 \# C_2^R \# C_3 \# C_4^R \#\ldots\# C_n^{(R?)} \#$$

and nondeterministically do one of:

1. accept if C_1 is *not* M's initial config on w
2. accept if C_n is *not* accepting, or
3. nondeterministically pick i and verify that C_i does *not* yield C_{i+1} in one step. (Push 1st; pop & compare to 2nd, with the necessary changes near the head.)

From P, build equiv CFG G; ask the hypothetical ALL_{CFG} subr if G generates all of $(\{\#\} \cup Q \cup \Gamma)^*$

Computable Functions

In addition to language recognition, we are also interested in computable functions.

Defn: a function $f: \Sigma^* \rightarrow \Sigma^*$ is *computable* if $\exists$ a TM M s.t. given *any* input $w \in \Sigma^*$, M <u>*halts*</u> with just f(w) on its tape. (Note: domain(f) = Σ^*; crucial that M always halt, else value undefined.)

Ex 1: $f(n) = n^2$ is computable

Ex 2: $g(\langle M,w \rangle) = \langle L_{M,w} \rangle$ (as in the $EMPTY_{LBA}$ pf) is computable

Ex 2: $h(\langle M,w \rangle)$ = "1 if M acc w else 0" is *un*computable (Why? Reduce A_{TM} to it.)

Reducibility

"A reducible to B" means could solve A *if* had subr for B

Can use B in arbitrary ways–call it repeatedly, use its answers to form new calls, etc. E.g.,

$$\text{WHACKY} \le_T A_{TM}$$

where WHACKY = { <M,w_1, w_2, ...,w_n> | M accepts

$a_1 \cdots a_n$, where a_i = 0 if M rejects w_i, 1 if accepts w_i }

BUT in "practice," *reductions rarely exploit this generality* and a more refined version is better for some purposes

Reduction

Notation (not in book, but common):

$A \le_T B$ means "A is Turing Reducible to B"

I.e., if I had a TM deciding B, I could use it as a subroutine to solve A

Facts:

$A \le_T B$ & B decidable implies A decidable (definition)

$A \le_T B$ & A *un*decidable implies B undecidable (contrapositive)

$A \le_T B$ & $B \le_T C$ implies $A \le_T C$

Mapping Reducibility

Defn: A is *mapping reducible* to B ($A \le_m B$) if there is computable function *f* such that $w \in A \Leftrightarrow f(w) \in B$

A special case of $\le_T$:
Call subr only once; its answer is *the* answer

Facts:

$A \le_m B$ & B decidable $\Rightarrow$ A is too

$A \le_m B$ & A *un*decidable $\Rightarrow$ B is too

$A \le_m B$ & $B \le_m C \Rightarrow A \le_m C$

Mapping Reducibility

Defn: A is *mapping reducible* to B ($A \le_m B$) if there is computable function *f* such that $w \in A \Leftrightarrow f(w) \in B$

A special case of $\le_T$:
Call subr only once; its answer is *the* answer

Theorem:

$A \le_m B$ & B decidable (recognizable) $\Rightarrow$ A is too

$A \le_m B$ & A *un*decidable (*un*recognizable) $\Rightarrow$ B is too

$A \le_m B$ & $B \le_m C \Rightarrow A \le_m C$

Most reductions we've seen were actually $\le_m$ reductions.

Lecture 11

Mapping Reducibility

Defn: A is *mapping reducible* to B ($A \leq_m B$) if there is computable function *f* such that $w \in A \Leftrightarrow f(w) \in B$

A special case of $\leq_T$:

Call subr only once; its answer is *the* answer

Facts:

$A \leq_m B$ & B decidable $\Rightarrow$ A is too

$A \leq_m B$ & A *un*decidable $\Rightarrow$ B is too

$A \leq_m B$ & $B \leq_m C \Rightarrow A \leq_m C$

Mapping Reducibility

Defn: A is *mapping reducible* to B ($A \leq_m B$) if there is computable function *f* such that $w \in A \Leftrightarrow f(w) \in B$

A special case of $\leq_T$:

Call subr only once; its answer is *the* answer

Theorem:

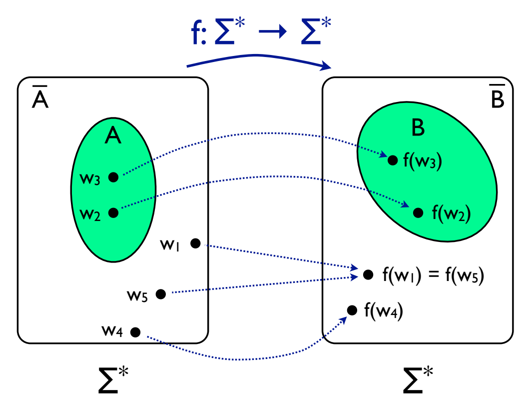
$A \leq_m B$ & B decidable (recognizable) $\Rightarrow$ A is too

$A \leq_m B$ & A *un*decidable (*un*recognizable) $\Rightarrow$ B is too

$A \leq_m B$ & $B \leq_m C \Rightarrow A \leq_m C$

Most reductions we've seen were actually $\leq_m$ reductions.

Mapping Reducibility



$\boxed{w \in A \Leftrightarrow f(w) \in B}$

Mapping Reducibility

Defn: A is *mapping reducible* to B ($A \le_m B$) if there is computable function *f* such that $w \in A \Leftrightarrow f(w) \in B$

Theorem:

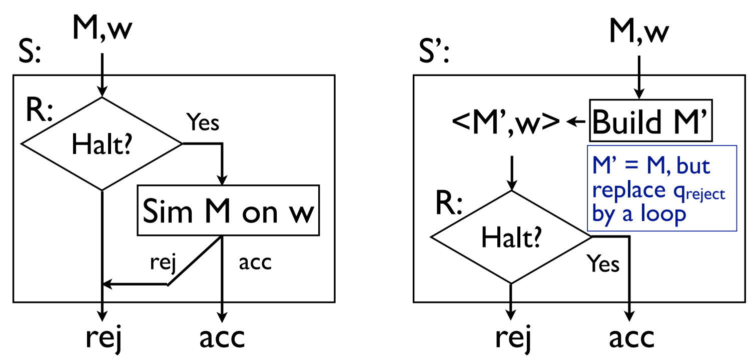
1) $A \le_m B$ & B decidable (recognizable) $\Rightarrow$ A is too
2) $A \le_m B$ & A *un*decidable (*un*recognizable) $\Rightarrow$ B is too
3) $A \le_m B$ & $B \le_m C \Rightarrow A \le_m C$

Proof:

1) To decide (recognize) w in A compute f(w), then use decider (recognizer, resp) for B on f(w).
2) Contrapositive
3) Given f for $A \rightarrow B$, g for $B \rightarrow C$; then $w \in A \Leftrightarrow g(f(w)) \in C$

A_{TM} ($\le_T$ vs $\le_m$) $HALT_{TM}$

$\boxed{f(\langle M,w \rangle) = \langle M',w \rangle}$



From Lecture 07

Other Examples of $\le_m$

$A_{TM} \le_m REGULAR_{TM}$ $\quad f(\langle M,w \rangle) = \langle M_2 \rangle$

Build M_2 so $L(M_2) = \Sigma^* / \{ 0^n 1^n \}$, as M accept/rejects w

$EMPTY_{TM} \le_m EQ_{TM}$ $\quad f(\langle M \rangle) = \langle M, M_{reject} \rangle$

$L(M_{reject}) = \varnothing$, so equiv to M iff $L(M) = \varnothing$

$\left.\begin{array}{l} A_{TM} \le_m MPCP \\ MPCP \le_m PCP \end{array}\right\}$ 5.2

$A_{TM} \le_m \overline{EMPTY_{TM}}$ $\quad f(\langle M,w \rangle) = \langle M_1 \rangle$

Build M_1 so $L(M_1) = \{w\} / \varnothing$, as M accept/rejects w

Lecture 12

Why TM's? Programs are OK too

Fix Σ = printable ASCII

Programming language with ints, strings & function calls

"Computable function" = always returns something

"Decider" = computable function always returning 0 / 1

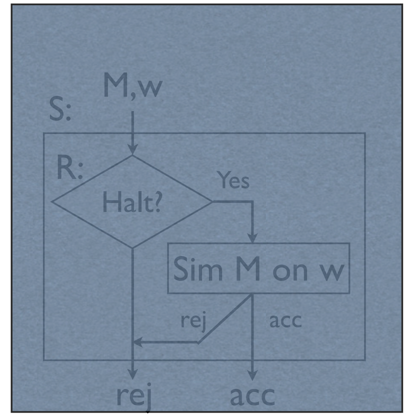
"Acceptor" = accept if return 1; reject if ≠1 or loop

A_{Prog} = {<P,w> | program P returns 1 on input w }

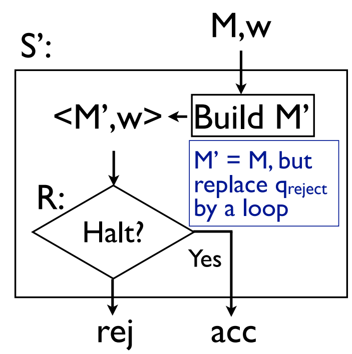
$HALT_{Prog}$ = {<P,w> | prog P returns *something* on w }

...

A_{TM} ($\leq_T$ vs $\leq_m$) $HALT_{TM}$



f(<M,w>) = <M',w>



From Lecture 07

$A_{Prog} \leq_m HALT_{Prog}$

f(<P,w>) = <P',w>

```
sub f(P,w){
  // build P'
  pn = ...//(find P's name)
  pp = "sub " + pn + "prime(x){"
  pp += P
  pp += "if "+pn+"(x) return 1;"
  pp += "while True {;}}"
  val = "<" + pp + "," + w + ">"
  return val
```

```
sub Pprime(x){
   sub P(y){
   ...          (copy
                 of P)
   }
   if P(x) return 1;
   while True { ; }
}
```

Programs vs TMs

Everything we've done re TMs can be rephrased re programs

From the Church-Turing thesis (hopefully made concrete in earlier HW) we know they are equivalent.

Above ex. shows some things are perhaps easier with programs.

Others get harder (e.g., "Universal TM" is a Java interpreter written in Java; "configurations" and "computation histories" are much messier)

TMs are convenient to use here since they strike a good balance

Hopefully you can mentally translate between the two; decidability/undecidability of various properties of programs are obviously more directly relevant.

Mapping Reducibility

Defn: A is *mapping reducible* to B ($A \leq_m B$) if there is computable function *f* such that $w \in A \Leftrightarrow f(w) \in B$

A special case of $\leq_T$:
  Call subr only once; its answer is *the* answer

Theorem:

$A \leq_m B$ & B   decidable   (recognizable) $\Rightarrow$ A is too

$A \leq_m B$ & A *un*decidable (*un*recognizable) $\Rightarrow$ B is too

$A \leq_m B$ & $B \leq_m C \Rightarrow A \leq_m C$

Most reductions we've seen were actually $\leq_m$ reductions.

Other Examples of $\leq_m$

$A_{TM} \leq_m REGULAR_{TM}$ $f(<M,w>) = <M_2>$

Build M_2 so $L(M_2) = \Sigma^* / \{ 0^n 1^n \}$, as M accept/rejects w

$EMPTY_{TM} \leq_m EQ_{TM}$ $f(<M>) = <M, M_{reject}>$

$L(M_{reject}) = \varnothing$, so equiv to M iff $L(M) = \varnothing$

$A_{TM} \leq_m MPCP$
$MPCP \leq_m PCP$
} 5.2

$A_{TM} \leq_m \overline{EMPTY_{TM}}$ $f(<M,w>) = <M_1>$

Build M_1 so $L(M_1) = \{w\} / \varnothing$, as M accept/rejects w

$EMPTY_{TM}$ is undecidable

From Lecture 07

$EMPTY_{TM} = \{ <M> \mid M \text{ is a TM s.t. } L(M) = \varnothing \}$

Pf: To show: $A_{TM} \leq_T EMPTY_{TM}$

On input <M,w> build M' :

M' on input x:
1. erase x
2. write w
3. run M on w
4. if M accepts w, then accept x
5. otherwise, reject x

Do *not* run M or M'. (That whole "halting thing" means we might not learn much if we did.) But note that L(M') is/is not empty exactly when M does not/does accept w, so knowing whether L(M') = $\varnothing$ answers whether <M,w> is in A_{TM}. And our hypothetical "$EMPTY_{TM}$" subroutine applied to M' tells us just that. I.e., $A_{TM} \leq_T EMPTY_{TM}$

$$L(M') = \begin{cases} \Sigma^*, & \text{if M accepts w} \\ \varnothing, & \text{if M rejects w} \end{cases}$$

NB: it shows $A_{TM} \leq_m \overline{(EMPTY_{TM})}$

$REGULAR_{TM}$ is undecidable

From Lecture 07

$REGULAR_{TM} = \{ <M> \mid M \text{ is a TM s.t. } L(M) \text{ is regular} \}$

Pf: To show: $A_{TM} \leq_T REGULAR_{TM}$

On input <M,w> build M' :

M' on input x:
1. if $x \in \{0^n 1^n | n \geq 0\}$, accept x
2. otherwise, erase x
3. write w
4. run M on w
5. if M accepts w, then accept x
6. otherwise, reject x

Do *not* run M or M'. (That whole "halting thing" ...) But note that L(M') is/is not regular exactly when M does/does not accept w, so knowing whether L(M') is regular answers whether <M,w> is in A_{TM}. The hypothetical "$REGULAR_{TM}$" subroutine applied to M' tells us just that. I.e., $A_{TM} \leq_T REGULAR_{TM}$

$$L(M') = \begin{cases} \Sigma^*, & \text{if M accepts w} \\ \{0^n 1^n | n \geq 0\}, & \text{otherwise} \end{cases}$$

Exercise: Is it $A_{TM} \leq_m REGULAR_{TM}$? If not, could it be changed?

More on $\le_T$ vs $\le_m$

Theorem: For *any* L, $L \le_T \overline{L}$

The same is not true of $\le_m$:

Theorem: L recognizable and $L \le_m \overline{L} \Rightarrow$ L is decidable.

Proof: on input x, dovetail recognizers for $x \in L$ & $f(x) \in L$

$(x \in L \Leftrightarrow f(x) \in \overline{L}$, so $x \notin L \Leftrightarrow f(x) \notin \overline{L} \Leftrightarrow f(x) \in L)$

Corr: $A_{TM} \le_T \overline{A_{TM}}$ but *not* $A_{TM} \le_m \overline{A_{TM}}$

Theorem: $A \le_m B$ iff $\overline{A} \le_m \overline{B}$

Theorem: If L is not recognizable and both $L \le_m B$ and $L \le_m \overline{B}$, then neither B nor $\overline{B}$ are recognizable

EQ_{TM} is neither recognizable nor co-recognizable

M_0: on any input x, reject x. $L(M_0) = \varnothing$

M_1: on any input x, accept x. $L(M_1) = \Sigma^*$

For any <M,w>, let h(<M,w>) = M_2 be the TM that, on input x,

1. runs M on w
2. if M accepts w, then accept x.

Claim: $L(M_2) = \Sigma^*$ (if $\langle M,w\rangle \in A_{TM}$), else $= \varnothing$ & h computable

Then $\overline{A_{TM}} \le_m EQ_{TM}$ via $g(\langle M,w\rangle) = \langle M_0,M_2\rangle$

And $\overline{A_{TM}} \le_m \overline{EQ_{TM}}$ via $f(\langle M,w\rangle) = \langle M_1,M_2\rangle$ (& $A_{TM} \le_m EQ_{TM}$)

Lecture 13

EQ_{TM} is neither recognizable nor co-recognizable

M_0: on any input x, reject x. $L(M_0) = \varnothing$

M_1: on any input x, accept x. $L(M_1) = \Sigma^*$

For any <M,w>, let h(<M,w>) = M_2 be the TM that, on input x,

1. runs M on w
2. if M accepts w, then accept x.

Claim: $L(M_2) = \Sigma^*$ (if $\langle M,w\rangle \in A_{TM}$), else $= \varnothing$ & h computable

Then $\overline{A_{TM}} \le_m EQ_{TM}$ via $g(\langle M,w\rangle) = \langle M_0,M_2\rangle$

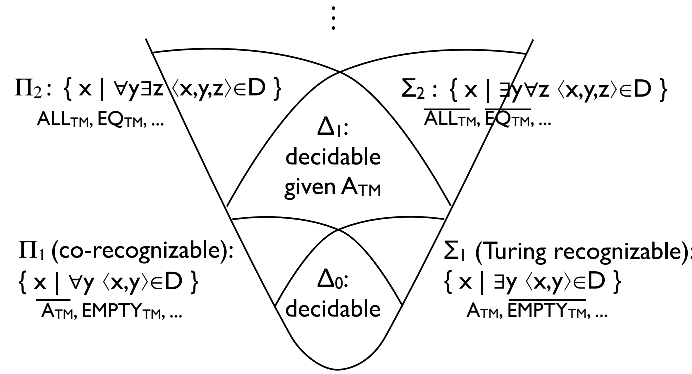
And $\overline{A_{TM}} \le_m \overline{EQ_{TM}}$ via $f(\langle M,w\rangle) = \langle M_1,M_2\rangle$ (& $A_{TM} \le_m EQ_{TM}$)

Defining Inequivalence

"If two TMs are not equivalent, there is some input w where they differ, and if they differ there is some time t such that one accepts within t steps, but the other will not accept no matter how long you run it."

$\overline{EQ_{TM}} = \{ x \mid \exists y \; \forall z \; \langle x,y,z \rangle \in D \}$ where the decidable set $D = \{$ triples $\langle x,y,z \rangle$ such that x is a pair of TMs, y is a pair w,t, and one machine accepts w within t steps but the other has not accepted w within z steps $\}$

The "Arithmetical Hierarchy"



Potential Utility: It is often easy to give such a quantifier-based characterization of a language; doing so suggests (but doesn't prove) whether it is decidable, recognizable, etc. and suggests candidates for reducing to it.

"The human mind seems limited in its ability to understand and visualize beyond four or five alternations of quantifier. Indeed, it can be argued that the inventions, subtheories, and central lemmas of various parts of mathematics are devices for assisting the mind in dealing with one or two additional alternations of quantifier."

H. Rogers, *The Theory of Recursive Functions and Effective Computability,* McGraw-Hill, 1967, pp 322-323.

Decidability Questions

Questions about a single TM:

- Detail questions: about *operation* or *structure* of a TM
  - useless state, does head move left, does it take >100 steps, ...
- Bottom-line questions: ask about a TM's *language*
  - Is L(M) empty? Infinite? Is 42 in L(M)? ...
  - About L(M), not M, *per se. Same answer for M' if L(M)=L(M')*

Other: Questions about ⟨M,w⟩, 2 TMs, grammars, ...

Language Properties

We formalize language *properties* simply as sets of languages

E.g., the "infiniteness" property is just the set of infinite languages.

A property is *non-trivial* if there is at least one language with the property and one without.

E.g., "emptiness" is nontrivial: $L_1=\varnothing$ has it; $L_2=\{42\}$ doesn't.

E.g., "countable" is trivial: every subset of Σ^* is countable

Rice's Theorem

Theorem:

For every nontrivial property $\mathcal{P}$ of the Turing recognizable languages, it is undecidable whether a TM recognizes a language having property $\mathcal{P}$. I.e.,

$$\mathcal{P}_{TM} = \{ \langle M \rangle \mid L(M) \in \mathcal{P} \}$$

is undecidable.

Corr:

$EMPTY_{TM}$, $INFINITE_{TM}$, $REGULAR_{TM}$, ... all undecidable

Rice's Theorem

$\mathcal{P}_{TM} = \{ <M> \mid M$ is a TM s.t. $L(M) \in \mathcal{P} \}$

M' on input x:
1. save x
2. write w
3. run M on w
4. if M accepts w, then run M_1 on x

L(M') = **?**

Rice's Theorem

$\mathcal{P}_{TM} = \{ <M> \mid M$ is a TM s.t. $L(M) \in \mathcal{P} \}$

M' on input x:
1. save x
2. write w
3. run M on w
4. if M accepts w, then run M_1 on x

$$L(M') = \begin{cases} L(M_1), & \text{if } M \text{ accepts } w \\ \varnothing, & \text{if } M \text{ rejects } w \end{cases}$$

Rice's Theorem

$\mathcal{P}_{TM}$ = { <M> | M is a TM s.t. $L(M) \in \mathcal{P}$ }

Pf: To show: $A_{TM} \leq_T \mathcal{P}_{TM}$. WLOG, $\varnothing \notin \mathcal{P}$; M_1 is a TM s.t. $L(M_1) \in \mathcal{P}$
On input <M,w> build M' :
Do *not* run M or M'. (That whole "halting thing" means we might not learn much if we did.) But note that L(M') is/is not in $\mathcal{P}$ exactly when M does/does not accept w, so knowing whether $L(M') \in \mathcal{P}$ answers whether <M,w> is in A_{TM}.
I.e., $A_{TM} \leq_T EMPTY_{TM}$

M' on input x:
1. save x
2. write w
3. run M on w
4. if M accepts w, then run M_1 on x

$$L(M') = \begin{cases} L(M_1), & \text{if M accepts w} \\ \varnothing, & \text{if M rejects w} \end{cases}$$

NB: it shows $A_{TM} \leq_m \mathcal{P}_{TM}$ or $\overline{\mathcal{P}_{TM}}$

Programs, in general, are opaque, inscrutable, confusing, complex, obscure, and generally yucky...

(If you've been a 142 TA, you might have observed this yourself...)

Decidability Questions

Questions about a single TM:

- Detail questions: about *operation* or *structure* of a TM
  - useless state, does head move left, does it take >100 steps, ...
- Bottom-line questions: ask about a TM's *language*
  - Is L(M) empty? Infinite? Is 42 in L(M)? ...
  - About L(M), not M, *per se*. *Same answer for M' if L(M)=L(M')*
- Other: Questions about ⟨M,w⟩, 2 TMs, grammars, ...

Rice's theorem doesn't (directly) answer these

But it says all these are undecidable (or trivial)