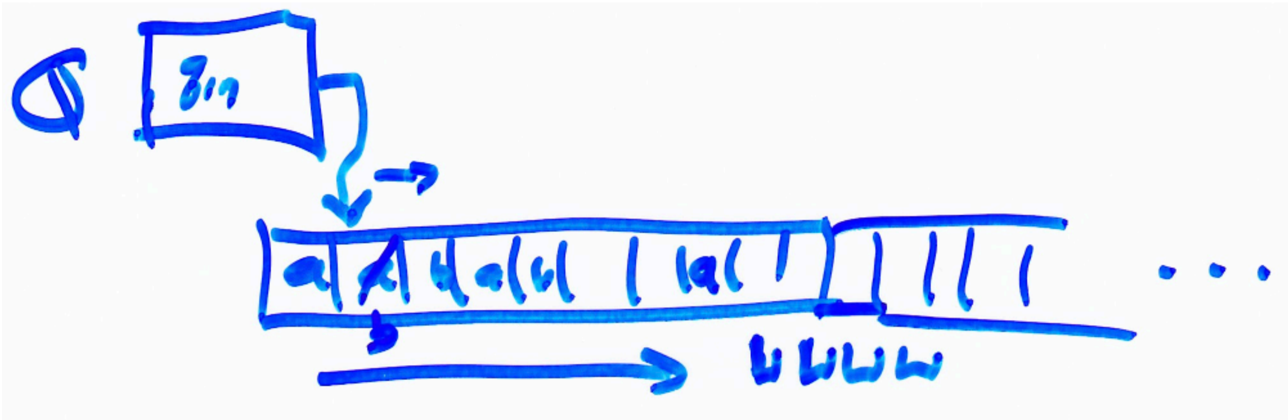
Turing Machines

Reading Assignment: Sipser Chapter 3.1, 4.2

4.1 covers algorithms for decidable problems about DFA, NFA, RegExp, CFG, and PDAs, e.g. slides 17 & 18 below. I've talked about most of this in class at one point or another, but skimming 4.1 would probably be a good review.



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

Q: finite state set

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

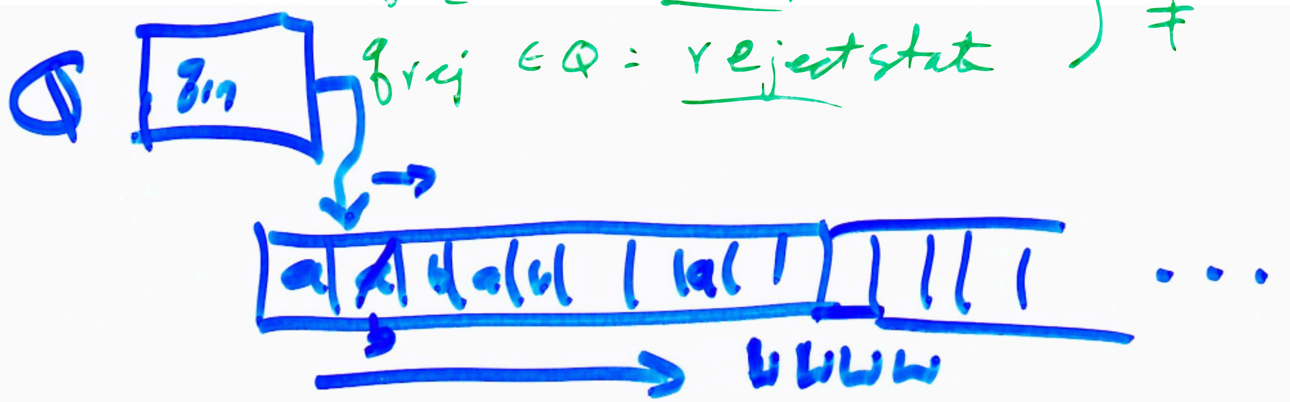
Γ: finite tape alphabet . $\Sigma \cup \{\sqcup\} \subseteq \Gamma$.

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 $\Big) \neq$



Example

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

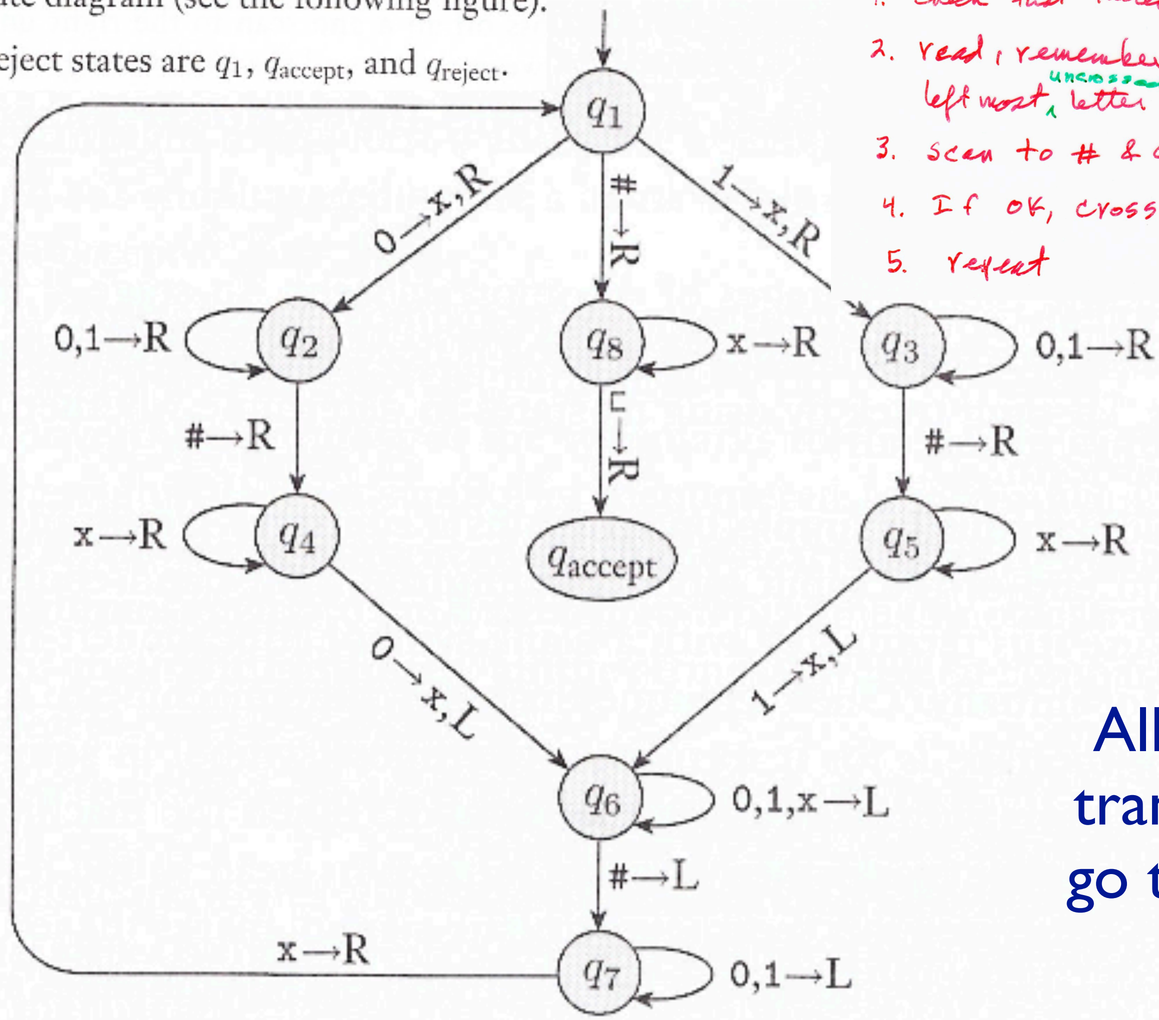
1. check that there's a single #
2. read, remember & cross off leftmost uncrossed letter
3. scan to # & compare next uncrossed letter
4. If OK, cross it off
5. repeat

- $Q = \{q_1, \ldots, q_{14}, q_{accept}, q_{reject}\}$,
- $\Sigma = \{0,1,\#\}$, 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}.

Example

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

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



All other transitions go to q_{reject}

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

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

L is *Turing recognizable* if $\exists$TM M s.t. L = L(M)

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

A key distinction!

Church-Turing Thesis

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

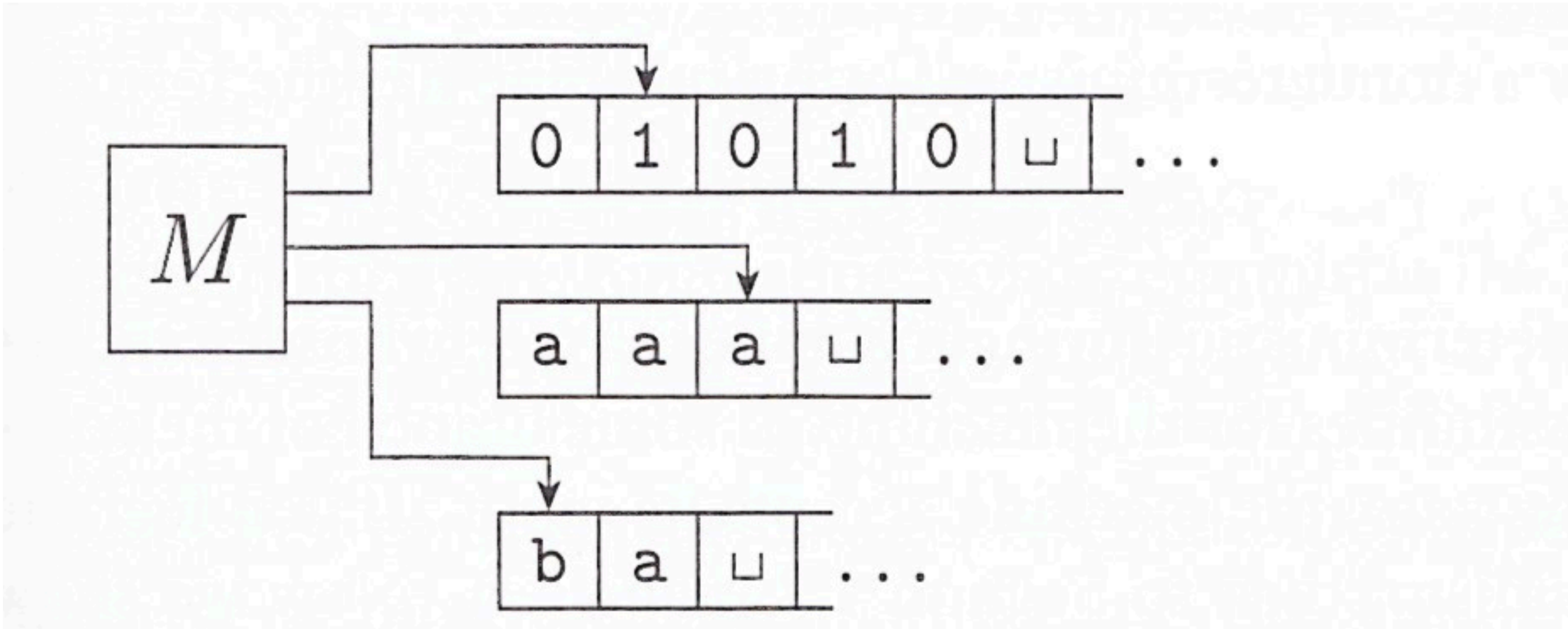
Not provable, since "intuitive" is necessarily fuzzy.

But, give support for it by showing that

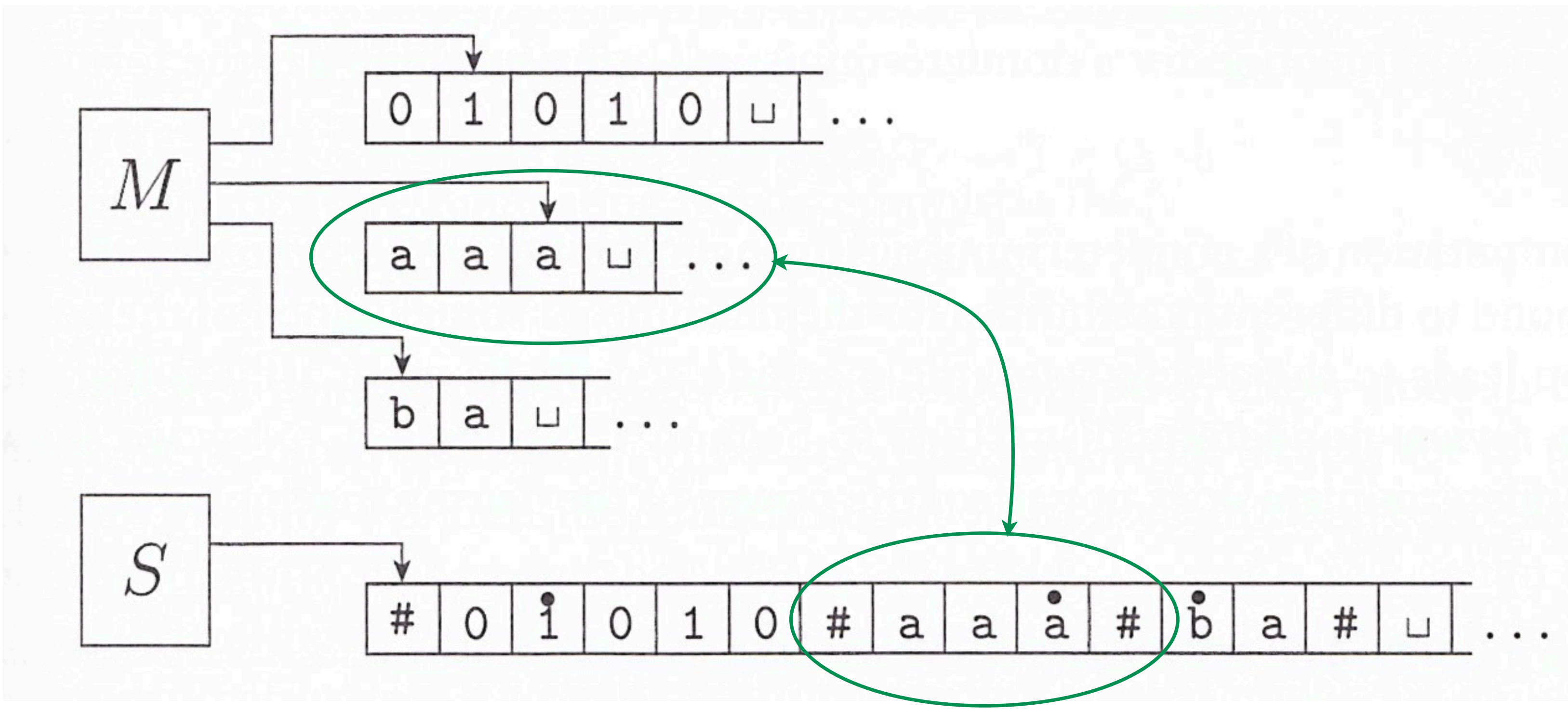
(a) other intuitively appealing (but formally defined) models are precisely equivalent, and

(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).

Example: Multi-tape Turing Machines



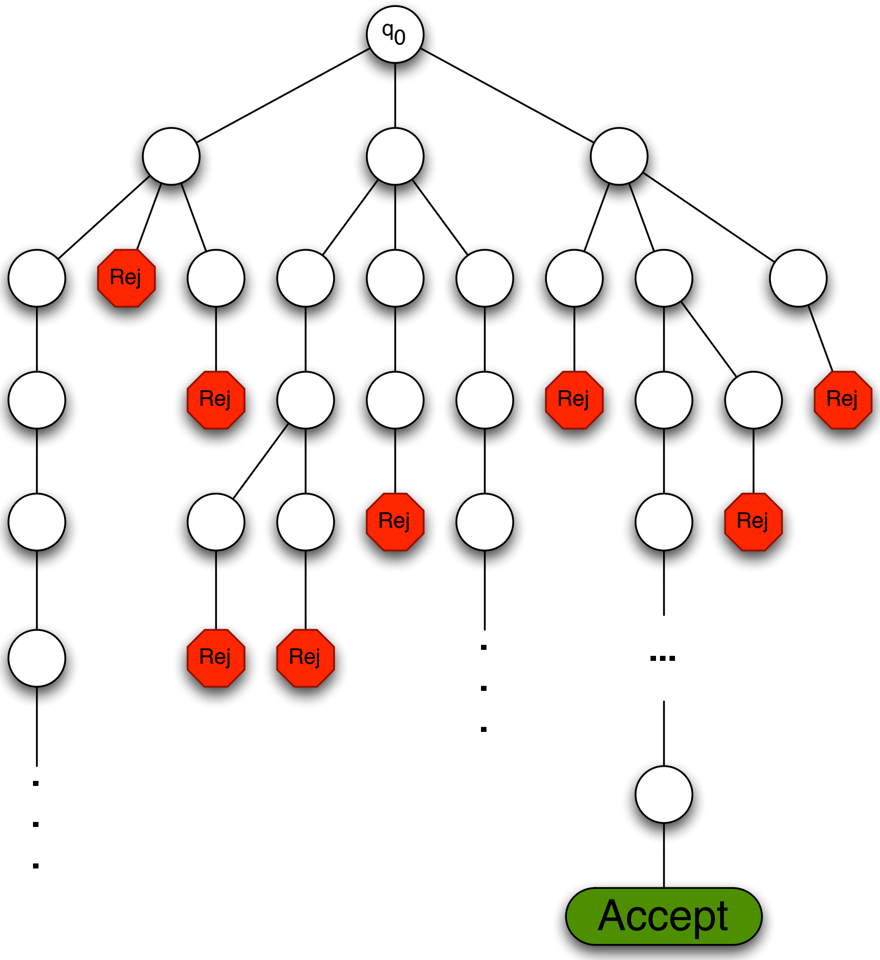
$$\delta \colon Q \times \Gamma^k \longrightarrow Q \times \Gamma^k \times \{\mathrm{L}, \mathrm{R}, \mathrm{S}\}^k$$



M
0 1 0 1 0 ⊔ ...
a a a ⊔ ...
b a ⊔ ...
S
0 1 0 1 0 # a a a # b a # ⊔ ...

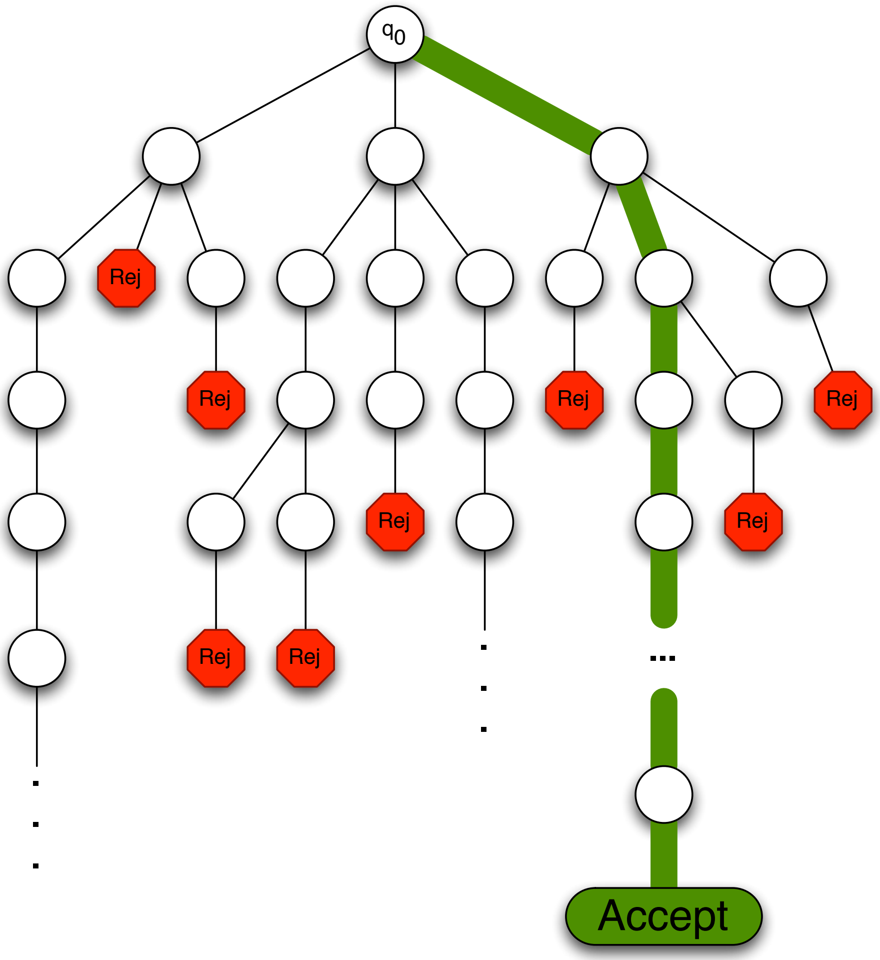
Nondeterministic Turing Machines

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



Nondeterministic Turing Machines

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

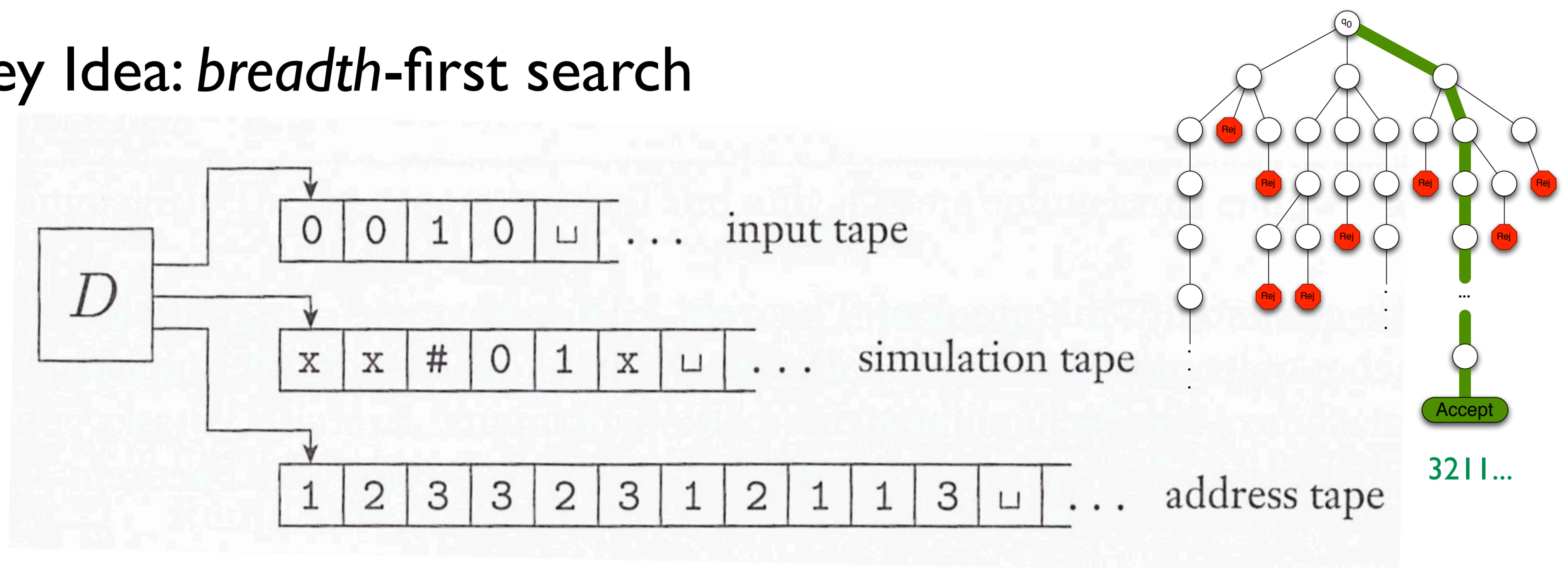


Accept if *any* path leads to q_{accept}; reject otherwise, (i.e., *all* halting paths lead to q_{reject})

Simulating an NTM

Key issue: avoid getting lost on ∞ path

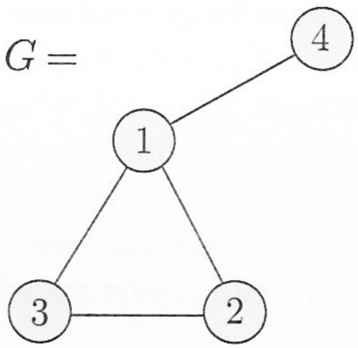
Key Idea: *breadth*-first search



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

Encoding things

$G =$



$\langle G \rangle =$

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

$\Sigma = ?$

CFG $G = (V, \Sigma, R, S)$; $<G> = ((S,A,B,\ldots),(a,b,\ldots), (S \rightarrow aA, S \rightarrow b, A \rightarrow cAb, \ldots),S)$

or $<G> = ((A_0, A_1, \ldots),(a_0, a_1, \ldots), (A_0 \rightarrow a_0 A_1, A_0 \rightarrow a_1, A_1 \rightarrow a_2 A_1 a_1, \ldots), A_0)$

DFA $D = (Q, \Sigma, \delta, q_0, F)$; $<D> = (\ldots)$

TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$; $<M> = (\ldots)$

...

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

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

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

$EQ_{DFA} = \{ <A,B> \mid A \text{ \& } B \text{ are DFAs s.t. } L(A)= L(B) \}$

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

$A_{CFG} = \{ <G,w> \mid \ldots \}$

pf: see book

$EMPTY_{CFG} = \{ <G> \mid \ldots \}$

pf: see book

$EQ_{CFG} = \{ \langle A,B \rangle \mid A \text{ \& } B \text{ are CFGs s.t. } L(A) = L(B) \}$

This is *NOT* decidable

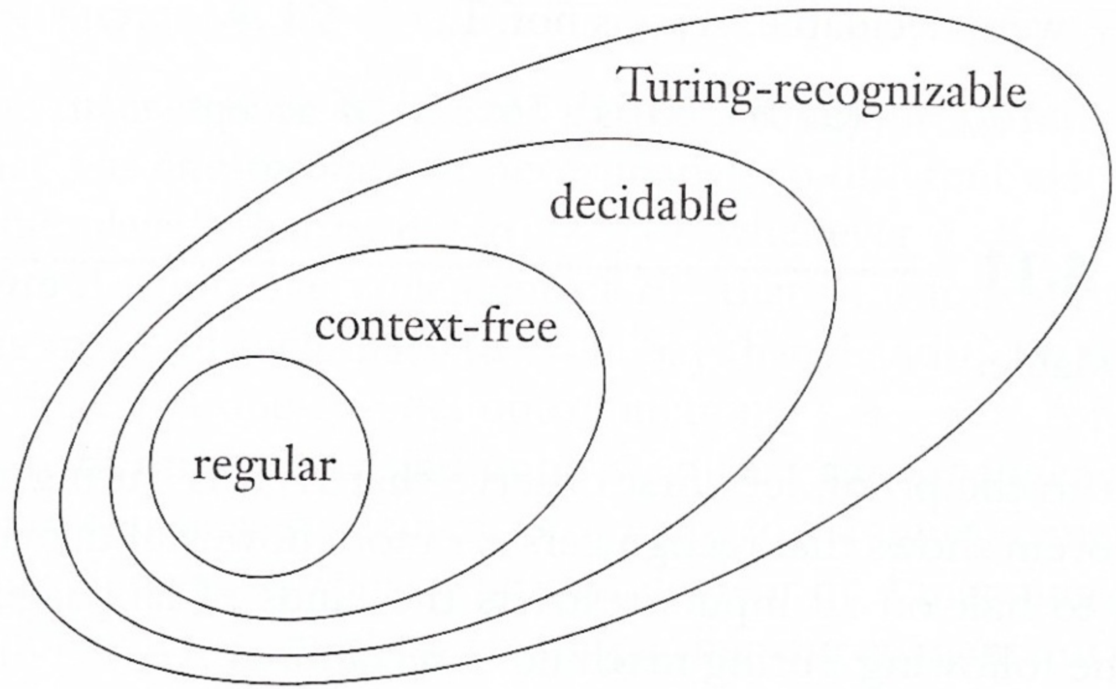


FIGURE 4.10
The relationship among classes of languages

The Acceptance Problem for TMs

$A_{TM} = \{ \langle M,w \rangle \mid M \text{ is a TM } \& \ w \in L(M) \}$

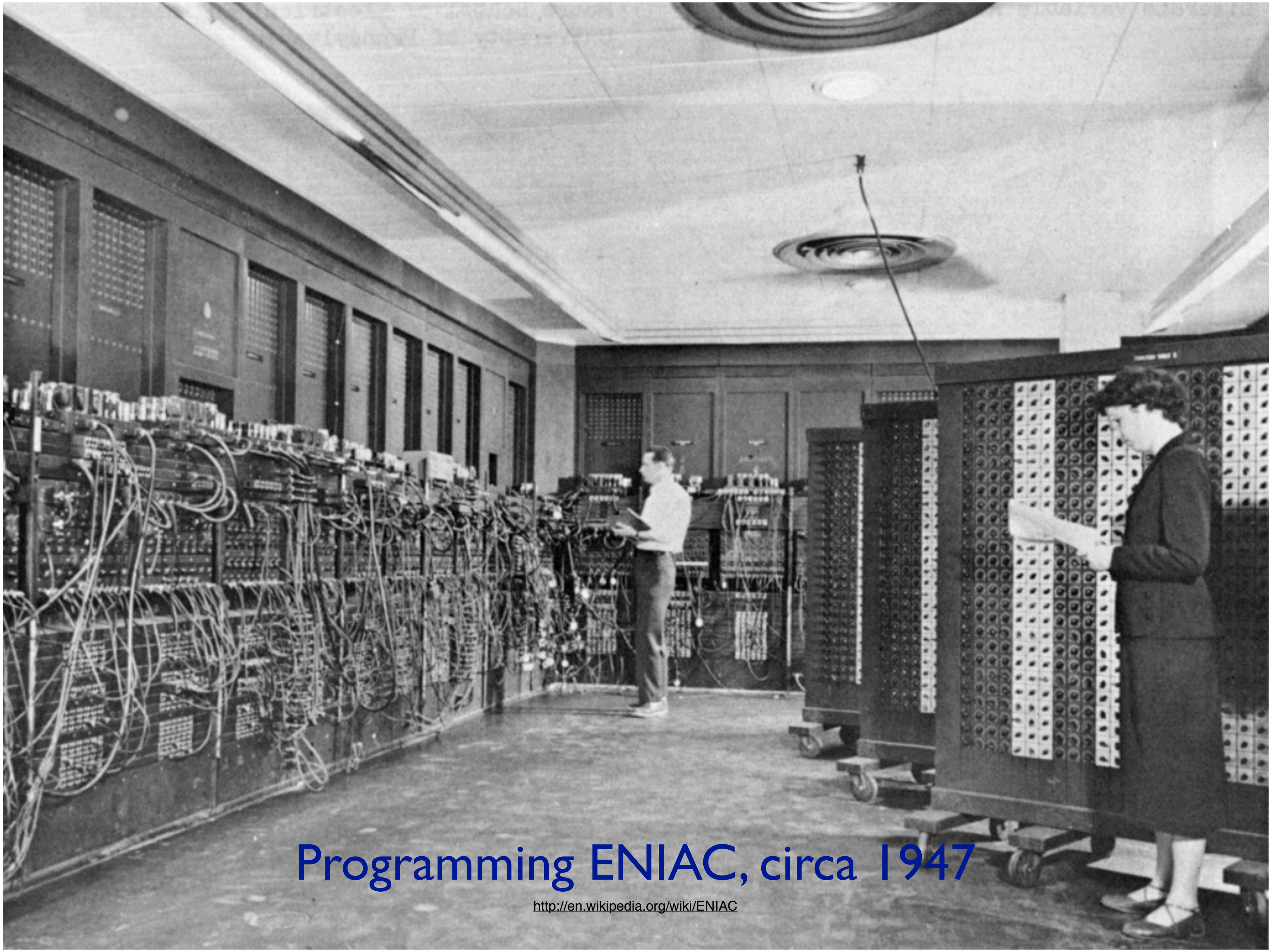
Theorem: A_{TM} is Turing recognizable

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.



Programming ENIAC, circa 1947

http://en.wikipedia.org/wiki/ENIAC

The Set 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

Proof idea:

"$\langle\ \rangle$" 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. $\langle M_i \rangle = w_i$

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

i, j entry $= 1 \Leftrightarrow M_i$ accepts w_j

$L_D = \{ w_i \mid i,i \text{ entry} = 0\}$

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	
			$\vdots$				$\ddots$

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 Idea:

Flip q_{accept}, q_{reject}, (just like we did with DFAs)

The Acceptance Problem for TMs

$A_{TM} = \{ \langle M,w \rangle \mid M$ 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. $\langle M_i \rangle = w_i$

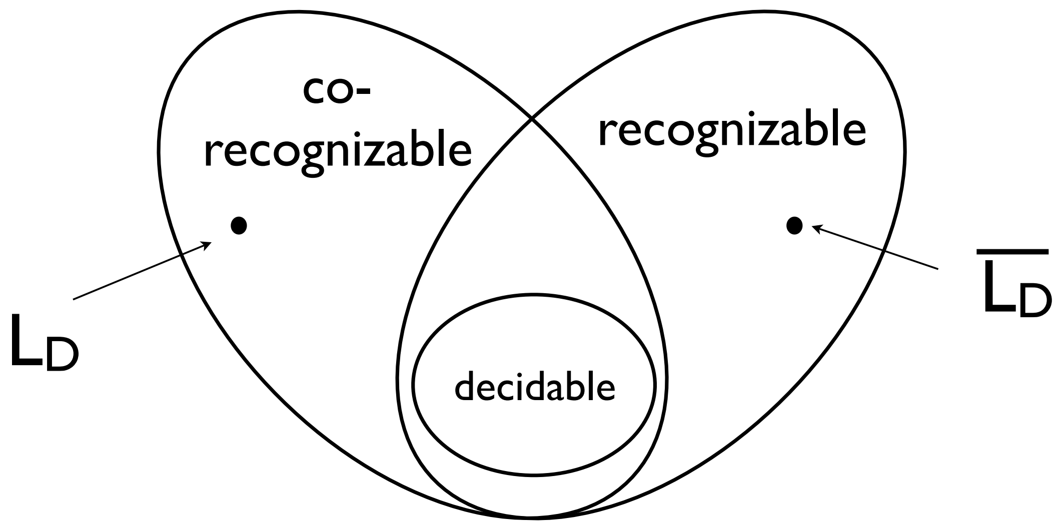
(M_i = some default machine if w_i is an illegal code.)

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

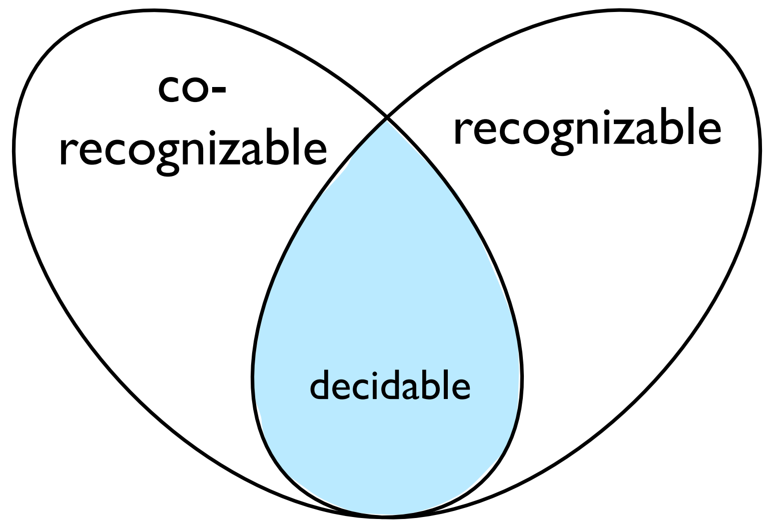


Decidable = Rec ∩ co-Rec

L decidable iff both L & L^c are recognizable

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

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



The Halting Problem

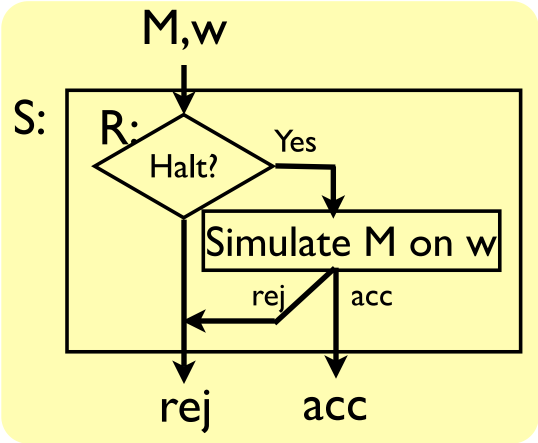
$HALT_{TM}$ = { <M,w> | TM M halts on input w }

Theorem: The halting problem is undecidable

Proof:

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.

Programs vs TMs

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

From the Church-Turing thesis, we expect them to be equivalent, and it's not hard to prove that they are

Some things are perhaps easier with programs.

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

TMs are convenient to use here since they strike a good balance between simplicity and versatility

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

Programs vs TMs

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 $\neq$1 or loop

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

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

...

Many Undecidable Problems

About Turing Machines

$HALT_{TM}$ EQ_{TM} $EMPTY_{TM}$ $REGULAR_{TM}$ ...

About programs

Ditto! *And*: array-out-of-bounds, unreachability, loop termination, assertion-checking, correctness, ...

About Other Things

$EMPTY_{LBA}$ ALL_{CFG} EQ_{CFG} PCP DiophantineEqns ...

Summary

Turing Machines

A simple model of "mechanical computation"

Church-Turing Thesis

All "reasonable" models are alike in capturing the intuitive notion of "mechanically computable"

Decidable/Recognizable – Key distinction: Does it halt

Undecidability – counting, diagonalization, reduction

A_{TM} = { <M,w> | TM M accepts w }

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

Want More?

Check out CSE 431
"Intro Computability & Complexity"