Today: Undecidability, Recognition, Enumeration, all that good stuff

Definition: A Turing Machine is a 7-tuple $T = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where:

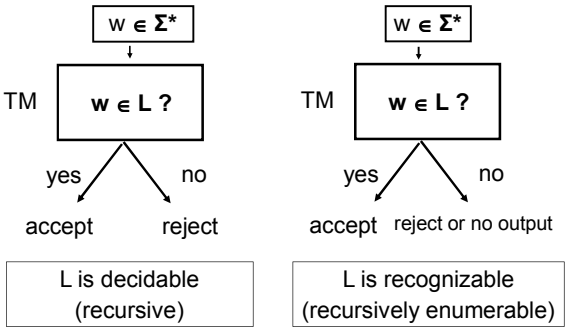
- **Q is a finite set of states**
- **Σ is the input alphabet, where $\square \notin \Sigma$**
- **Γ is the tape alphabet, where $\square \in \Gamma$ and $\Sigma \subseteq \Gamma$**
- **$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,R\}$**
- **$q_0 \in Q$ is the start state**
- **$q_{accept} \in Q$ is the accept state**
- **$q_{reject} \in Q$ is the reject state, and $q_{reject} \neq q_{accept}$**

A TM recognizes a language if it accepts all and only those strings in the language

A language is called recognizable or recursively enumerable, (or r.e.) if some TM recognizes it

A TM decides a language if it accepts all strings in the language and rejects all strings not in the language

A language is called decidable (or recursive) if some TM decides it



Theorem: L is decidable if both L and ¬L are recursively enumerable

Recall: Given $L \subseteq \Sigma^*$, define $\neg L := \Sigma^* - L$

Theorem: L is decidable iff both L and ¬L are recognizable

Given:
a TM M_1 that recognizes A and
a TM M_2 that recognizes ¬A,
we can build a new machine M that *decides* A

Simulate M_1 (x) on one tape, M_2 (x) on another. One of them must halt. If M_1 halts then accept. If M_2 halts then reject.

There are languages over {0,1} that are not decidable

Assuming the Church-Turing Thesis, this means there are problems that NO computing device can solve

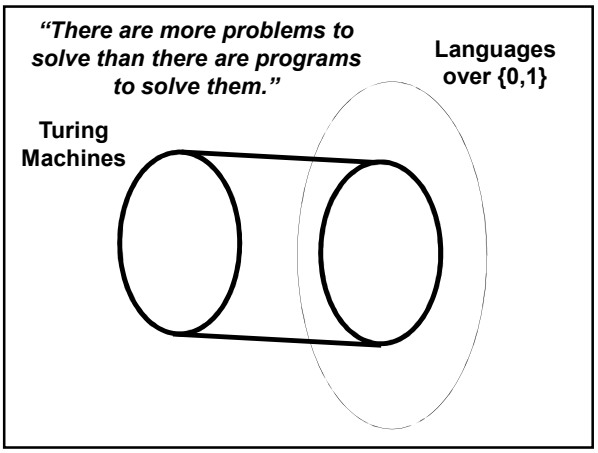
We can prove this using a counting argument. We will show there is no onto function from the set of all Turing Machines to the set of all languages over {0,1}. (But it works for any Σ)

That is, every mapping from Turing machines to languages must "miss" some language.

There are languages over {0,1} that are not *recognizable*

Proof Outline:

1. Every recognizable language can be associated with a Turing machine
2. Every TM corresponds to a bit string
3. So there is a 1-1 mapping from the set of all recognizable languages, to {0,1}*
4. But the set of *all* languages has a bijection with the POWER SET of {0,1}*
5. The power set of A is always larger than A, so there must be unrecognizable languages



Let L be any set and 2^L be the power set of L

Theorem: There is no onto map from L to 2^L

Proof: Assume, for a contradiction, that there is an onto map $f : L \to 2^L$

Let $S = \{ x \in L \mid x \notin f(x) \} \in 2^L$

If f is onto, then there is a $y \in L$ where $f(y) = S$
Suppose $y \in S$. By definition of S, $y \notin f(y) = S$.
So $y \notin S$. By definition of S, $y \in f(y) = S$.
Contradiction

MORAL:

No map from L to the power set 2^L can cover all elements in 2^L

No matter what the set L is, the power set 2^L *always* has strictly greater cardinality than L

There are undecidable (and unrecognizable) languages over {0,1}

{Recognizable languages over {0,1}}
↓
{Turing Machines} ↓ {0,1}*
| |
Set **L**

{Languages over {0,1}}
| |
{Sets of strings of 0s and 1s}
| |
Set of all subsets of L: 2^L

There are (many) unrecognizable languages

Russell's Paradox in Set Theory

In the early 1900's, logicians were trying to define consistent foundations for mathematics.

Suppose X = "universe of all possible objects"
"Frege's Axiom" Let $P : X \to \{0,1\}$.
Then {S in X | P(S) = 1} is a set.

Define $F = \{ S \in X \mid S \notin S \}$

Suppose $F \in F$. Then by definition, $F \notin F$.
So $F \notin F$ and by definition $F \in F$.
This set theory is inconsistent!

Theorem: There is no onto function from the positive integers Z^+ to the real numbers in (0, 1)
{0,1}* Power set of {0,1}*

Proof: Suppose f is such a function:

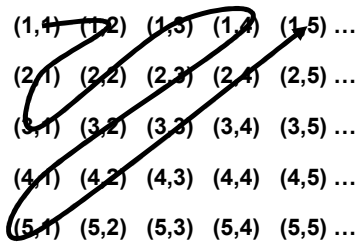
1 → 0.**2**8347279…
2 → 0.8**8**388384…
3 → 0.77**6**35284…
4 → 0.111**1**1111…
5 → 0.12345**5**678…
: :

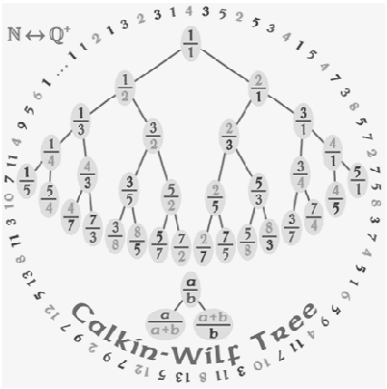
[n-th digit of r] = { **1** if [n-th digit of f(n)] ≠ 1 ; **0** otherwise

f(n) ≠ r for all n (Here, r = 0.11101...) r is never output by f

---

Let Z^+ = {1,2,3,4…}.
There is a bijection between Z^+ and $Z^+ \times Z^+$





A Concrete Undecidable Problem

A_{TM} = { (M, w) | M is a TM that accepts string w }

Theorem: A_{TM} is recognizable but NOT decidable

A_{TM} is recognizable:
Define a TM U as follows: U is a *universal TM*

On input (M, w), U runs M on w. If M ever accepts, accept. If M ever rejects, reject.

Therefore,
U accepts (M,w) ⇔ M accepts w ⇔ (M,w) ∈ A_{TM}
Therefore, U *recognizes* A_{TM}

A_{TM} = { (M,w) | M is a TM that accepts string w }

A_{TM} is undecidable: (proof by contradiction)

Assume machine H decides A_{TM}

H((M,w)) = { **Accept** if M accepts w ; **Reject** if M does not accept w

Construct a new TM D as follows: *on input M, run H on (M,M) and output the opposite of H.*

D(D) = { **Reject** if D accepts D ; **Accept** if D does not accept D

OUTPUT OF H(x,y)

	M_1	M_2	M_3	M_4 …	D
M_1	accept	accept	accept	reject	accept
M_2	reject	accept	reject	reject	reject
M_3	accept	reject	reject	accept	accept
M_4	accept	reject	reject	reject	accept
:					
D	reject	reject	accept	accept	?

OUTPUT OF D(M)

	M_1	M_2	M_3	M_4 …	D
M_1	reject	accept	accept	reject	accept
M_2	reject	reject	reject	reject	reject
M_3	accept	reject	accept	accept	accept
M_4	accept	reject	reject	accept	accept
:					
D	reject	reject	accept	accept	?

D(M) outputs the opposite of H(M,M)

A_{TM} = { (M,w) | M is a TM that accepts string w }

A_{TM} is undecidable: (constructive proof)

Assume machine H recognizes A_{TM}

$$H((M,w)) = \begin{cases} \text{Accept if M accepts w} \\ \text{Rejects or loops otherwise} \end{cases}$$

Construct a new TM D_H as follows: *on input M, run H on (M,M) and output the "opposite" of H whenever you get an answer from H.*

$$D_H(D_H) = \begin{cases} \text{Reject if } D_H \text{ accepts } D_H \\ \text{(i.e. if } H(D_H, D_H) = \text{Accept)} \\ \text{Accept if } D_H \text{ rejects } D_H \\ \text{(i.e. if } H(D_H, D_H) = \text{Reject)} \\ \text{loops if } D_H \text{ loops on } D_H \\ \text{(i.e. if } H(D_H, D_H) \text{ loops)} \end{cases}$$

Note: There is no contradiction here!

D_H must run forever on D_H

We can effectively construct an instance (D_H, D_H) which does not belong to A_{TM} but H fails to tell us that: $H(D_H, D_H)$ runs forever!

That is:

Given any machine H recognizing A_{TM}

(H is a potential decider for A_{TM})

we can effectively construct an instance which does not belong to A_{TM} (namely, (D_H, D_H))

but H runs forever on the input (D_H, D_H)

So H cannot decide A_{TM}

Given any program that recognizes the Acceptance Problem, we can efficiently construct an input where the program hangs!

Theorem: A_{TM} is recognizable but NOT decidable

Corollary: $\neg A_{TM}$ is not recognizable!

$\neg A_{TM}$ = { (M,w) | M does not accept string w }

Proof: Suppose $\neg A_{TM}$ is recognizable.
Then $\neg A_{TM}$ and A_{TM} are both recognizable…
But that would mean they're decidable…

The Halting Problem

$HALT_{TM}$ = { (M,w) | M is a TM that halts on string w }

Theorem: $HALT_{TM}$ is undecidable

Proof: Assume (for a contradiction) there is a TM H that decides $HALT_{TM}$

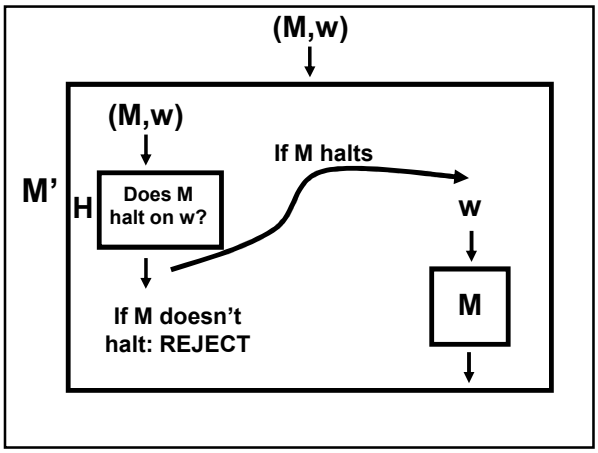
We use H to construct a TM M' that *decides* A_{TM}

M'(M,w): run H(M,w)

If H rejects then reject

If H accepts, run M on w until it halts:

Accept if M accepts and
Reject if M rejects



The Emptiness Problem

$EMPTY_{TM}$ = { M | M is a TM such that L(M) = ∅}

Given a program, does it always reject?

Theorem: $EMPTY_{TM}$ is undecidable

Proof: Assume (for a contradiction) there is a TM E that decides $EMPTY_{TM}$
We'll use it to get a decider D for A_{TM}

D(M,w) := Build a TM M' with the behavior:
"M'(x) := if x ≠ w reject else sim M(w)"
Run E(M').
If E accepts, *reject.* If rejects, *accept.*

The Complement of Emptiness

$\neg EMPTY_{TM}$ = { M | M is a TM such that L(M) ≠ ∅}

Given a program, does it accept some input?

Theorem: $\neg EMPTY_{TM}$ is recognizable

Proof:
M'(M) :=
For all pairs of positive integers (i, t),
Let x be the ith string in lex order.
Determine if M(x) accepts within t steps
If *yes* then *accept.*

L(M) ≠ ∅ if and only if M' halts and accepts M

The Emptiness Problem

Theorem: $EMPTY_{TM}$ is undecidable

Theorem: $\neg EMPTY_{TM}$ is recognizable

Corollary: $EMPTY_{TM}$ is *unrecognizable*

Proof: If $\neg EMPTY_{TM}$ and $EMPTY_{TM}$ recognizable, that would imply $EMPTY_{TM}$ is decidable

One can often show that a language L is undecidable by showing that if L is decidable, then so is A_{TM}

We reduce A_{TM} to the language L

$$A_{TM} \leq L$$

We showed: $A_{TM} \leq HALT_{TM}$
and $A_{TM} \leq \neg EMPTY_{TM}$

Mapping Reductions

$f : \Sigma^* \rightarrow \Sigma^*$ is a computable function if there is a Turing machine M that halts with just f(w) written on its tape, for every input w.

A language A is *mapping reducible* to language B, written $A \leq_m B$, if there is a computable $f : \Sigma^* \rightarrow \Sigma^*$, such that for every w,

$$w \in A \Leftrightarrow f(w) \in B$$

f is called a mapping reduction
(or many-one reduction) from A to B

Next Episode:

Undecidability and Reductions Between Problems