Reducibility

- A reduction is a way of converting one problem into a second problem such that a solution to the second problem can be used to solve the first.

  if A reduces to B then a solution for B can be used to solve A

finding the area of a square reduces to finding the length of a side.

$HALT_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM that halts on } w\}$

is undecidable

Proof by Contradiction

- assume $Halt_{TM}$ is decidable R decides $Halt_{TM}$
- Construct S to decide A_{TM}

S = "On input $\langle M, w\rangle$

1. If R rejects REJECT
2. If R accepts, simulate M on w until it halts
3. If M accepts ACCEPT
   If M rejects REJECT"

$Reg_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is regular}\}$

is undecidable

- Let R be a TM that decides Reg_{TM}
- Using R we construct TM S that decides A_{TM}

S = "On input $\langle M, w \rangle$

1. Construct machine M_1,

   M_1 = "On input x

   1. If x has the form 0^n1^n accept
   2. If x does not have that form run M on input w and ACCEPT if M accepts w.

2. Run R on input $\langle M_1 \rangle$
3. If R accepts ACCEPT, If R rejects REJECT"

if M accepts w then machine M_1 accepts Σ^*

if M does not accept w then M_1 accepts 0^n1^n

$E_{TM} = \{ \langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset \}$

is undecidable

- Let R be a TM that decides E_{TM}
- Use R to construct TM S that decides A_{TM}

S = "On input $\langle M, w \rangle$

1. Construct TM M_2

   M_2 = "On input x

   1. if $x \neq w$ reject
   2. if $x = w$, run M on w and accept if M does."

2. Run R on input $\langle M_2 \rangle$
3. If R accepts REJECT, if R rejects ACCEPT."

if M accepts w M_2 accepts w $L(M_2) \neq \emptyset$

if M rejects w $L(M_2) = \emptyset$

$EQ_{TM} = \{\langle M_1, M_2 \rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$

is undecidable

- Let R decide EQ_{TM}
- Construct S using R that decides E_{TM}

S = "On input $\langle M \rangle$

1. Run R on input $\langle M, M_1 \rangle$, where M_1 is a TM that rejects all inputs.
2. IF R accepts ACCEPT, if R rejects REJECT"

$EQ_{CFG} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are CFG and } L(G) = L(H)\}$

why can't we use the same technique as EQ_{DFA}?

CFGs are not closed under intersection or complement.

Undecidability.

What problems can't be solved algorithmically?

Given a computer program and a specification can you automate the process of verifying that the program behaves correctly.

$A_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM that accepts } w \}$

A_{TM} is recognizable

U = "On input $\langle M, w \rangle$

1. Simulate M on input w
2. If M accepts w ACCEPT, if M rejects w REJECT"

why isn't this a decider for A_{TM}?

M might loop on input w and never reach the accept or reject states.

What is the difference between countably infinite and uncountably infinite?

A_{TM} is undecidable

Some languages are not Turing-recognizable

- The set of all TM is countable
  - Σ^* is countable for any alphabet Σ
    - first list all strings of length 0
    - then all strings of length 1
    - followed by 2 etc.
  - Since every TM is encoded by a string in Σ^*
  - By simply eliminating every string that isn't a valid encoding of a TM we have a countable list of all TM

- The set of all languages is uncountable
  - The set of all infinite binary sequences is uncountable
  - Languages can be represented as binary sequences

$\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000 \ldots\}$

$A = \{\quad 0, \quad 00, 01, \quad\quad 000, \ldots\}$

A_{TM} is undecidable

Proof by contradiction

- Assume A_{TM} is decidable
- Let H be a machine that decides A_{TM}

$$H(\langle M, w \rangle) = \begin{cases} \text{accept} & \text{if } M \text{ accepts } w \\ \text{reject} & \text{if } M \text{ rejects } w \end{cases}$$

- Construct machine D that uses H as a subroutine

D = "On input $\langle M \rangle$, where M is a TM

1. Run H on input $\langle M, \langle M \rangle \rangle$
2. Output the opposite of what H outputs."

$$D(\langle M \rangle) = \begin{cases} \text{accept} & \text{if } M \text{ does not accept } \langle M \rangle \\ \text{reject} & \text{if } M \text{ accepts } \langle M \rangle \end{cases}$$

- Run D on input $\langle D \rangle$

$$D(\langle D \rangle) = \begin{cases} \text{accept} & \text{if } D \text{ does not accept } \langle D \rangle \\ \text{reject} & \text{if } D \text{ does accept } \langle D \rangle \end{cases}$$

- No matter what D does it is forced to do the opposite. Which is a contradiction

Therefore TM H cannot exist

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$ ...	$\langle D \rangle$
M_1	accept	reject	accept	reject	
M_2	reject	accept	accept	reject	
M_3	accept	accept	accept	accept	
M_4	reject	reject	reject	reject	
⋮					
D	reject	reject	reject	accept ...	?

Turing Unrecognizable

Theorem

A language is decidable iff it is Turing-recognizable and its complement is Turing-recognizable

Proof 1

- if A is decidable then A is recognizable. The complement of a decidable language is decidable.

Proof 2

- If A and $\bar{A}$ are Turing-recognizable then A is decidable.
  - Let M_1 be a recognizer for A
  - Let M_2 be a recognizer for $\bar{A}$
  - Construct M that decides A

    M = " On input w:

    1. Run M_1 and M_2 on input w in parallel.
    2. IF M_1 accepts Accept
       if M_2 accepts REJECT

$\overline{A_{TM}}$ is ~~undecidable~~ not recognizable

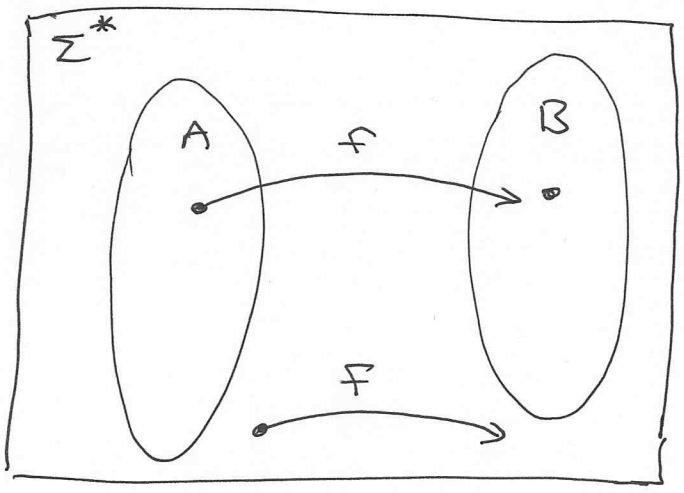
- A_{TM} is recognizable.

Mapping reductions $A \leq_m B$

Language A is mapping reducible to language B if there is a computible function $F: \Sigma^* \to \Sigma^*$ where for every w

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

The function f is called the reduction from A to B



We can use this to show Language A is not more difficult to compute than Language B.

Theorem

If $A \leq_m B$ and B is decidable, then A is decidable.

Proof

- Let M be a decider for B
- Let f be the reduction from A to B
- Describe TM M' that decides A.

M' = On input w:

1. Compute $f(w)$
2. Run M on input $f(w)$: if M accepts Accept, if M rejects REJECT."

If $w \in A$ then $f(w) \in B$

if $w \notin A$ then $f(w) \notin B$

If $A \leq_m B$ and A is undecidable then B is undecidable

$A_{TM} \leq_m Halt_{TM}$

Halting problem is at least as difficult as A_{TM}

$\exists$ a mapping reduction from A_{TM} to $Halt_{TM}$

-Determine a computable function f such that

$$\langle M, w \rangle \in A_{TM} \text{ iff } \langle M', w' \rangle \in Halt_{TM}$$

F = "On input $\langle M, w \rangle$

1. Construct M' where

   M' = "On input x

   1. Run M on x
   2. If M accepts ACCEPT
   3. If M rejects, enter a loop."

2. Output $\langle M', w \rangle$"

If M accepts w M' halts ~~and accepts w~~

If M rejects w M' does not halt

If M loop on w M' does not halt

$A_{TM} \leq_m Reg_{TM}$

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

$\langle M, w \rangle \in A_{TM}$ iff $L(M')$ is regular

M' = "On input x:

1. if $x = 0^n 1^n$ then accept
2. otherwise Run M on input w

if M accepts w ACCEPT x

if M rejects w Reject x"

if M accepts w then $\langle M, w \rangle \in A_{TM}$

and $L(M') = \{0,1\}^*$

if M does not accept w the $\langle M, w \rangle \notin A_{TM}$

and $L(M') = \{0^n 1^n \mid n \geq 0\}$

Assume there exists a decider D for Reg_{TM}

$D(\langle M' \rangle)$ accept $\Leftrightarrow L(M') = \{0,1\}^* \Leftrightarrow \langle M, w \rangle \in A_{TM}$

$A_{TM} \leq_m Reg_{TM}$

Reg_{TM} is undecidable

$SS_{TM} = \{ \langle M \rangle \mid M \text{ is a TM and } L(M) = \{001\} \}$

$A_{TM} \leq_m SS_{TM}$

want $f(\langle M, w \rangle) = \langle M' \rangle$

$\langle M, w \rangle \in A_{TM}$ iff $L(M') = \{001\}$

M' = "On input x:

1. if $x \neq 001$ then ~~ACCEPT~~ REJECT
2. Run M on input w
   if M accepts ACCEPT
   if M rejects REJECT"

If M accepts w, $\langle M, w \rangle \in A_{TM}$
and $L(M') = \{001\}$

if M does not accept w $\langle M, w \rangle \notin A_{TM}$
and $L(M') = \emptyset$

The same general approach will work for any single string.

$Infinite_{TM} = \{ \langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is infinite} \}$

$A_{TM} \leq_m Infinite_{TM}$

Want $f(\langle M, w \rangle) = \langle M' \rangle$

~~if~~ M accepts $w \iff L(M')$ is infinite

M' = On input x

1. Run M on w
2. if M accepts w, Accept x
3. if M rejects w, reject x"

if M accepts w then $\langle M, w \rangle \in A_{TM}$
and $L(M') = \Sigma^*$

if M does not accept w then $\langle M, w \rangle \notin A_{TM}$
and $L(M') = \emptyset$

Facts about Mapping Reductions $\leq_m$

1. if $A \leq_m B$ and B is decidable then A is decidable
   ↳ if $A \leq_m B$ and A is undecidable then B is undecidable

2. if $A \leq_m B$ and B is recognizable then A is recognizable
   ↳ " " " A is not recognizable then B is not recognizable

3. if $A \leq_m B$ then $\overline{A} \leq_m \overline{B}$

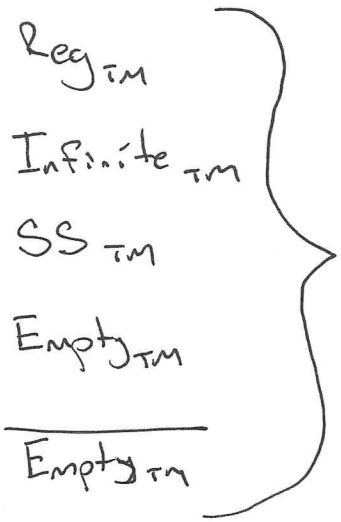
$$A_{TM} \leq_m Reg_{TM}$$

$$\overline{A_{TM}} \leq_m \overline{Reg_{TM}} \Rightarrow \overline{Reg_{TM}} \text{ is not recognizable}$$

also

$$A_{TM} \leq_m \overline{Reg_{TM}}$$

$$\overline{A_{TM}} \leq_m Reg_{TM} \Rightarrow Reg_{TM} \text{ is not recognizable}$$



all undecidable

all properties of the language of the TM

Any attempt to partition the set of TMs based on some property of the language will be undecidable.

Rice's Theorem

There does not exist an algorithm that can partition the set of all TMs based solely on properties of their language.

Def: A property of a language is a predicate

$$P: \mathcal{P}(\Sigma^*) \rightarrow \{0,1\}$$

P: "This language is regular"

$$P(L) = 1 \text{ iff } L \text{ is regular}$$

Def: A property of Turing-recognizable languages is a predicate of language L s.t. $\exists$ TM M with $L(M) = L$

$$\hookrightarrow L_P = \{\langle M \rangle \mid M \text{ is a TM and } P(L(M)) = 1\}$$

A property of Turing-recognizable languages is

<u>non-trivial</u> if $L_P \neq \emptyset$ and $L_{\bar{P}} \neq \emptyset$

Rice's Theorem:

If P is a non-trivial property of Turing-recognizable languages then:

$$L_P = \{ \langle M \rangle \mid M \text{ is a TM and } P(L(M)) = 1 \}$$

is undecidable

✓ P_M: "This TM has the same language as TM M_0"

↳ $P(L(M)) = 1$ iff $L(M) = L(M_0)$

✗ P_{repeat}: "This TM, on every input, never visits the initial state more than twice."

↳ Not a property of the language

✓ $P_{\text{decidable}}$: "The language of this TM is decidable"

↳ $P(L(M)) = 1$ iff $\exists$ TM D that decides $L(M)$

✗ $P_{\text{recognizable}}$: "The language of this TM is recognizable"

↳ Trivial (Property of Turing-recognizable language)

Proof

Show that $A_{TM} \leq_m L_P$

- Assume that $P(\emptyset) = 0$
- Since P is a nontrivial property.

  $\exists$ TM S s.t. $P(L(S)) = 1$
- Find $f(\langle M, w \rangle) = \langle M' \rangle$

M' = "On input x:

1. Run M on w
2. If M rejects w REJECT x
3. If M accepts w
   - Run S on x
   - If S accepts x ACCEPT x
   - If S rejects x REJECT x

If M accepts w

- $L(M') = L(S)$

If M rejects w

$L(M') = \emptyset$

If M loops on w

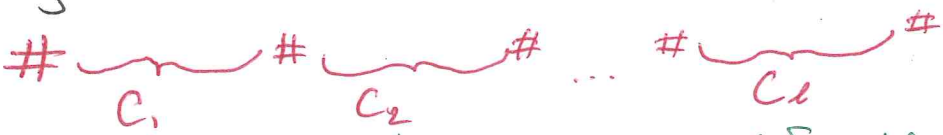
$L(M') = \emptyset$

Computation Histories

Let M be a TM and w an input string.

An accepting computation history for M on w is a sequence of configurations, $C_1, C_2, \ldots C_\ell$ where C_1 is the start configuration of M on w C_ℓ is an accepting configuration of M, and each C_i legally follows from C_{i-1} according to the rules of M.

A rejecting computation history for M on w is defined similarly except that C_ℓ is a rejecting configuration.



Computation histories are finite sequences. If M does not halt on w then no accepting history exists.

Deterministic Machines have at most one computation history for M on w

ALL_{CFG} is undecidable

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

Proof

- Assume ALL_{CFG} is decidable and use this assumption to show that A_{TM} is decidable.
- For TM M and input w we construct GFG G that ~~accepts~~ generates all strings iff M does not accept w

  If M does accept w, G will not generate some particular string.
- We design G to ~~accept~~ generate all strings that are not accepting computation histories ~~for~~ for M on w.
- A string may fail to be an accepting Computation History for several reasons

  $\#C_1\#C_2\#\ldots\#C_\ell\#$

  where C_i is the configuration for the i^{th} step of M's computation on w

- G generates all strings that:
  1. Do not start with C_1
  2. Do not end with an accepting configuration
  3. Some C_i does not properly yield C_{i+1} under the rules of M
- If M does not accept w then no accepting computation history exists. So G will generate all strings.
- Construct PDA D that can be converted into a grammar later
- D starts by non-deterministically branching to decide which of the three to check
- Checking if some C_i does not yield C_{i+1} is the hard part
  - nondeterministically choose some i
  - Push C_i onto the stack until the # is reached
  - Pop the stack to compare with C_{i+1}
  - They should match except ~~where~~ around the ~~the~~ tape head ~~was~~

- Any difference between C_i and C_{i+1} should be ~~ad~~dicated by the transition function for M.

- D accepts if it discovers a mismatched or an improper value.

- The problem is C_i is in the reverse order when popped from the stack.

- To solve this we write every other configuration in reverse order.