Chapter 4
Decidability

Overview

- previously, we introduced Turing machines as a model of general-purpose computing
- we also defined an algorithm in terms of Turing machines through the Church-Turing thesis
- now, we will show that certain problems can be solved algorithmically and others cannot
- why study unsolvability?
  - useful to know that a problem is unsolvable so that it must be simplified or altered before it can be solved with an algorithm
  - will help gain perspective on computation

Decidable Languages

- we now turn to examples of languages that are decidable by algorithms
  - focus on languages concerning automata and grammars
    - often related to applications
    - example: string membership in a language related to compiling programs
    - some problems concerning automata and grammars are not decidable by algorithms

Decidable Languages

- decidable problems concerning regular languages
  - computational problems concerning finite automata
  - represent computational problems through languages
    - already have a framework and terminology
    - e.g., acceptance problem for DFAs of testing whether a DFA accepts a given string
      - A_{DFA} = {<B,w> | B is a DFA that accepts w}
    - the problem of testing whether DFA B accepts input string w is the same as testing whether <B,w> is a member of the language A_{DFA}
    - can express other computational problems similarly
    - showing a language is decidable is the same as showing the computational problem is decidable

Decidable Languages

- Theorem: A_{DFA} is a decidable language
  - proof idea
    - show a TM M that decides A_{DFA}

    M = "On input <B,w>, where B is a DFA and w is a string:
      1. Simulate B on input w
      2. If the simulation ends in an accept state, accept. If it ends in a nonaccepting state, reject."

Decidable Languages

- Theorem: A_{DFA} is a decidable language
  - proof
    - imagine writing a program to carry out the simulation
    - first, examine input <B,w>
      - we could represent DFA B by its five components: $Q, \Sigma, \delta, q_0, F$
      - when M receives input, checks whether it properly represents DFA B and string w
        - if not, M rejects
    - M carries out simulation directly
      - keeps track of B's current state and position in w by writing it on its tape
        - starts at q_0 and beginning of w
        - state and position updated through transition function δ
      - when M finishes processing last symbol of w
        - accepts if B is in accepting state
        - otherwise rejects

Decidable Languages

- similarly, we can prove for NFAs, too
  - A_{NFA} = {<B,w> | B is a NFA that accepts w}
  - the problem of testing whether NFA B accepts input string w is the same as testing whether <B,w> is a member of the language A_{NFA}

Decidable Languages

- Theorem: A_{NFA} is a decidable language
  - proof
    - show a TM N that decides A_{NFA}
    - could design N like M for DFAs, but for NFAs
    - instead, we'll convert N to a DFA first, then use M as a subroutine

N = "On input <B,w>, where B is a NFA and w is a string:

1. Convert NFA B to an equivalent DFA C
2. Run TM M on input <C,w>
3. If M accepts, accept; otherwise, reject."

Decidable Languages

- similarly, we can determine whether a regular expression generates a given string
  - A_{REX} = {<R,w> | R is a RE that generates w}

Decidable Languages

- Theorem: A_{REX} is a decidable language
  - proof
    - TM P decides A_{REX}

P = "On input <R,w>, where R is a RE and w is a string:

1. Convert regular expression R to an equivalent NFA A
2. Run TM N on input <A,w>
3. If N accepts, accept; otherwise, reject."

Decidable Languages

- for decidability purposes, it is equivalent to present the Turing machine with a DFA, NFA, or a regular expression because the machines can convert from one form of encoding to another

- next, we discuss emptiness testing for the language of a finite automaton
  - previously, we had to determine if a FA accepts a particular string
  - for emptiness testing, we have to determine if a FA accepts any strings at all
    - $E_{DFA} = \{\langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$

Decidable Languages

- Theorem: E_{DFA} is a decidable language
  - proof
    - a DFA accepts a string iff reaching an accept state from the start state along the edges
    - design a TM T that uses a marking system similar to the connected graph TM

T = "On input <*A*>, where *A* is a DFA:

1. Mark the start state of *A*.
2. Repeat until no new states get marked:
3. Mark any state that has a transition coming into it from any state that is already marked.
4. If no accept state is marked, accept; otherwise, reject."

Decidable Languages

- next, determine whether determining if two DFAs recognize the same language is decidable
  - $EQ_{DFA} = \{\langle A,B \rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$

Decidable Languages

- Theorem: EQ_{DFA} is a decidable language
  - proof
    - construct a new DFA C that accepts only those strings that are accepted by either A or B, but not both
      - if A and B recognize the same language, C will accept nothing

$$L(C) = \left(L(A) \cap \overline{L(B)}\right) \cup \left(\overline{L(A)} \cap L(B)\right).$$

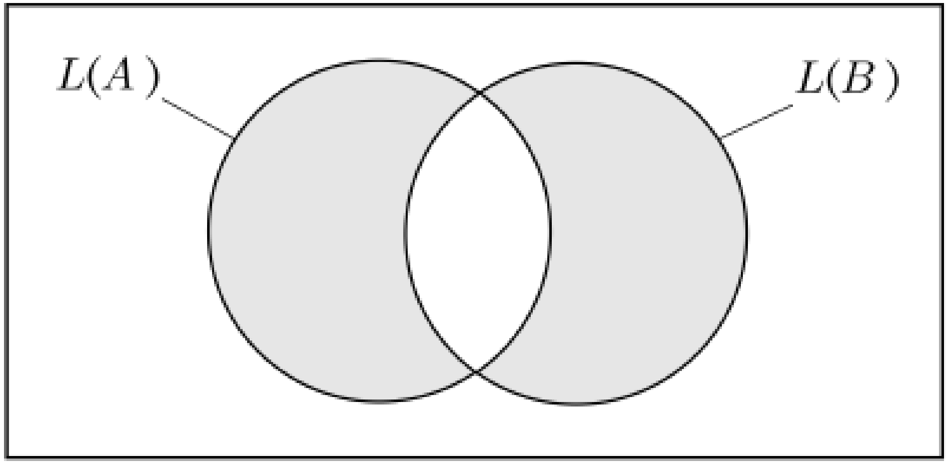
    - this is the symmetric difference of L(A) and L(B)
      - L(C) = Ø if L(A) = L(B)
    - construct C from A and B with the constructions for proving the class of regular languages closed under complementation, union, and intersection
      - these construction algorithms can be performed with Turing machines
    - use previous theorem to test whether L(C) is empty
    - if so, L(A) and L(B) must be equal

Decidable Languages

- Theorem: EQ_{DFA} is a decidable language
  - proof (cont.)

    F = "On input <A,B>, where A and B are DFAs:

    1. Construct DFA C as described.
    2. Run TM T from previous theorem on input <C>.
    3. If T accepts, accept; otherwise, reject."



Decidable Languages

- decidable problems concerning context-free languages
  - describe algorithms to determine whether a CFG generates a particular string and to determine whether the language of a CFG is empty
  - A_{CFG} = {<G,w> | G is a CFG that generates string w}

Decidable Languages

- Theorem: A_{CFG} is a decidable language
  - proof idea
    - we want to determine if G generates w
      - use G to go through all derivations to find w
        - doesn't work – infinite number of derivations
        - if G does not generate w, algorithm would compute forever
        - this is a recognizer, not a decider
    - for a decider, ensure that the algorithm tries only a finite number of derivations
      - previously, we showed if G were in Chomsky normal form, any derivation of w has 2n – 1 steps, where n = |w|
      - so, we can just check derivations with 2n – 1 steps
      - only a finite number of such derivations
      - can convert G to Chomsky normal form

Decidable Languages

- Theorem: A_{CFG} is a decidable language
  - proof (cont.)

    S = "On input <G,w>, where G is a CFG and w is a string:

    1. Convert G to an equivalent grammar in Chomsky normal form.
    2. List all derivations with 2n – 1 steps, where n = |w|; except if n= 0, then instead list all derivations with one step.
    3. If any of these derivations generates w, accept; otherwise, reject."

Decidable Languages

- determining whether a CFG generates a particular string is related to compiling programming languages
  - algorithm for TM S is very inefficient and would never be used in practice
  - but is easy to describe and we don't care about efficiency

- recall that we have procedures for converting back and forth between CFGs and PDAs
  - so, everything about CFG decidability is true for PDAs
- for emptiness testing for CFGs, we can show the problem is decidable
  - $E_{CFG} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$

Decidable Languages

- Theorem: E_{CFG} is a decidable language
  - proof idea
    - could use TM S that states whether a CFG generates a particular w
    - to determine if L(G) = Ø, we could try generating all possible w's, one by one, but infinite number of w's
    - instead, test whether start variable can generate a string of terminals
      - in general, determines for each variable whether it is capable of generating a string of terminals
      - if so, algorithm places a mark on that variable
    - first, mark all of the terminal symbols in the grammar
      - scan the rules of the grammar
      - if a rule is found that permits a variable to be replaced by some string of symbols, all of which are already marked, mark this variable, too
      - continue until no more variables can be marked

Decidable Languages

- Theorem: E_{CFG} is a decidable language
  - proof

    R = "On input <G>, where G is a CFG:

    1. Mark all terminal symbols in G.
    2. Repeat until no new variables get marked:
    3. Mark any variable A where G has a rule $A \rightarrow U_1U_2\ldots U_k$ and each symbol $U_1\ldots U_k$ has already been marked.
    4. If the start variable is not marked, accept; otherwise, reject."

Decidable Languages

- next, determine whether determining if two CFGs generate the same language
  - $EQ_{CFG} = \{\langle G,H \rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}$

- for EQ_{DFA}, we use E_{DFA}
  - but, we cannot use E_{CFG} to prove EQ_{CFG} is decidable since the CFLs are not closed under complementation or intersection
  - actually, EQ_{CFG} is not decidable

Decidable Languages

- Theorem: every CFL is decidable
  - proof idea
    - goal: show CFL A is decidable
    - could convert a PDA for A directly into a TM
      - relatively easy
      - PDA may be non-deterministic, but can be converted into NTM, which can then be converted into a DTM
      - but, some branches of the PDA may go on forever
        - this would also be reflected in the equivalent TM
    - instead, use TM S that decided A_{CFG}

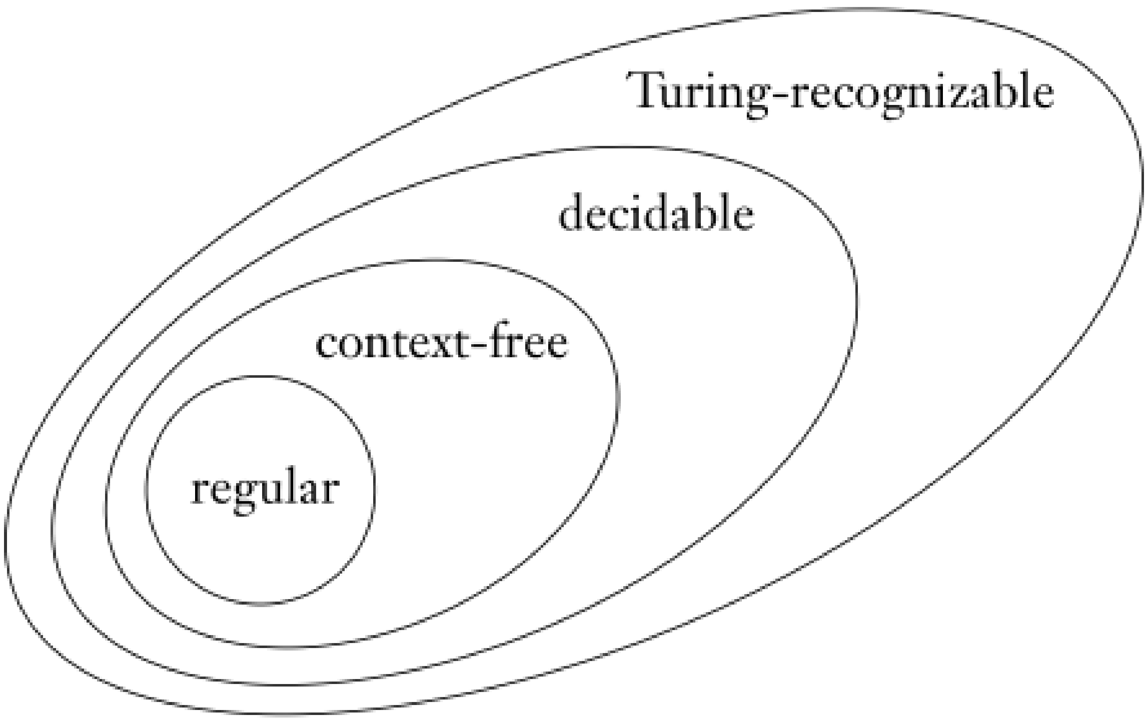
Decidable Languages

- Theorem: every CFL is decidable
  - proof
    - let G be a CFG for A and design TM M_G that decides A; build a copy of G into M_G

    M_G = "On input w:
    1. Run TM S on input $\langle G,w \rangle$.
    2. If this machine accepts, accept; otherwise, reject."

Decidable Languages

- we have now linked the relationships across the four main classes of languages: regular, context-free, decidable, and Turing-recognizable



Undecidability

- in this section, we will prove that a specific problem is unsolvable
  - philosophically important
- computers are extremely powerful, but some problems test their limits
  - often very ordinary problems
  - e.g., verifying that a sorting program is correct

Undecidability

- both A_{DFA} and A_{CFG} were decidable
- A_{TM} is undecidable
  - A_{TM} = {<M,w> | M is a TM and M accepts w}
  - A_{TM} is Turing-recognizable
    - recognizers are more powerful than deciders
      - requiring a TM to halt on all inputs restricts the kinds of languages it can recognize

Undecidability

- the following Turing machine recognizes A_{TM}

  U = "On input <M,w> where M is a TM and w is a string:
  1. Simulate M on input w.
  2. If M ever enters its accept state, accept; if M ever enters its reject state, reject."
  - this machine loops on <M,w> if M loops on w
    - therefore, this machine does not decide A_{TM}
    - if the algorithm had a way to determine that M was not halting on w, it could reject
      - however, an algorithm has no way to determine this
  - Turing machine U is called the Universal Turing Machine
    - capable of simulating every other Turing machine

Undecidability

- the proof of the undecidability of A_{TM} uses diagonalization
  - originally designed by Georg Cantor in 1873 to measure infinite sets
    - if we have two infinite sets, which one is larger?
      - e.g., even integers vs. all strings over (0,1)
      - can determine relative sizes by pairing elements (no counting involved)

Undecidability

- can use mapping function

DEFINITION 4.12

Assume that we have sets A and B and a function f from A to B. Say that f is ***one-to-one*** if it never maps two different elements to the same place—that is, if $f(a) \neq f(b)$ whenever $a \neq b$. Say that f is ***onto*** if it hits every element of B—that is, if for every $b \in B$ there is an $a \in A$ such that $f(a) = b$. Say that A and B are the ***same size*** if there is a one-to-one, onto function $f\colon A \longrightarrow B$. A function that is both one-to-one and onto is called a ***correspondence***. In a correspondence, every element of A maps to a unique element of B and each element of B has a unique element of A mapping to it. A correspondence is simply a way of pairing the elements of A with the elements of B.

- also called
  - injective – one-to-one
  - surjective – onto
  - bijective – one-to-one and onto

Undecidability

- example: Let N = {1, 2, 3, …} (natural numbers) and
  E = {2, 4, 6, …} (even numbers)
  - by Cantor's definition, they are the same size
    - for mapping N to E, use f(n) = 2n

n	$f(n)$
1	2
2	4
3	6
$\vdots$	$\vdots$

    - intuitively, E seems smaller since E is a proper subset of N
    - since the mapping is possible, they are the same size

Undecidability

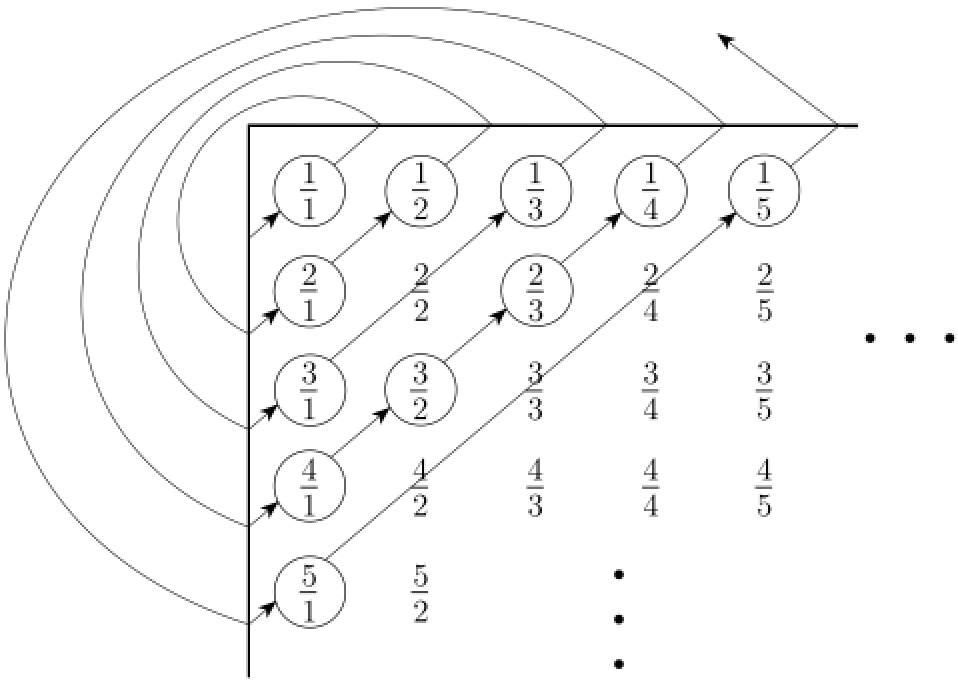
- a set *A* is countable if it is finite or has the same size as N
  - from the previous example, E is countable

Undecidability

- example: Let Q = {m/n | m,n ∈ N} (rational numbers)
  - Q seems to be much larger than N
  - by Cantor's definition, they are the same size
    - for mapping N to Q, list all elements of Q
      - pair first element with 1, second element with 2, etc.
      - each element of Q can only appear once
      - create matrix where i^{th} row contains all numbers with numerator i
      - j^{th} column has all denominators with j
      - i/j is listed in the i^{th} row, j^{th} column
      - for mapping, don't go row by row (why not?)

Undecidability

- example: Let Q = {m/n | m,n ∈ N} (rational numbers)
  - instead, go by diagonals
  - don't include any value that's already been listed



  - since we have the mapping, Q is countable

Undecidability

- after seeing these mappings, it may seem all infinite sets are the same size
  - just have to show the correspondence to N
  - but for some infinite sets, no correspondence to N exists
    - uncountable
    - the real numbers, R, are uncountable
      - R: any number that has a decimal representation
        - e.g., pi, sqrt(2)

Undecidability

- R is uncountable
  - proof: show no correspondence exists between R and N
  - use proof by contradiction
  - suppose R is countable
    - then a correspondence exists between R and N
    - use construction to help
      - choose each digit of x to make x different from one of the real numbers paired with an element from N
      - e.g., f(1) = 3.14159… f(2) = 55.55555… f(3) = …

Undecidability

- R is uncountable (cont.)
  - suppose R is countable
    - table showing one-to-one correspondence

n	$f(n)$
1	3.14159...
2	55.55555...
3	0.12345...
4	0.50000...
⋮	⋮

    - now, construct x between 0 and 1 and ensure $x \neq f(n)$ for any n
    - let first digit of x be different from first digit of f(1)
    - let second digit of x be different from f(2), etc.
    - continue through diagonal of table

Undecidability

- R is uncountable (cont.)
  - suppose R is countable
    - now, x cannot be in the table since it differs by at least one digit with every element in the table

n	$f(n)$
1	$3.\underline{1}4159\ldots$
2	$55.5\underline{5}555\ldots$
3	$0.12\underline{3}45\ldots$
4	$0.500\underline{0}0\ldots$
$\vdots$	$\vdots$

$x = 0.4641\ldots$

      - avoid 0 or 9 since 0.199... = 0.200...
    - contradiction: every real number is not in the table
      - therefore, R is uncountable

Undecidability

- since R is uncountable
  - some languages are not decidable or even Turing-recognizable
    - because there are uncountably many languages, but only countably many Turing machines
      - therefore, some languages are not recognized by any Turing machine

Undecidability

- some languages are not Turing-recognizable
  - first, show the set of all Turing machines is countable
    - set of all strings Σ* is countable for any alphabet Σ
      - with only finitely many strings of each length, write down strings of length 0, 1, 2, etc.
    - set of all Turing machines is countable because each Turing machine M has an encoding into a string <M>
      - if we omit those strings that are not legal encodings, we can make a list of Turing machines

Undecidability

- some languages are not Turing-recognizable (cont.)
  - second, show the set of all languages is uncountable
    - show set of all infinite binary sequences B is uncountable
      - infinite binary sequence: unending sequence of 0s and 1s
      - show B is uncountable using diagonalization as before for R

Undecidability

- some languages are not Turing-recognizable (cont.)
  - second, show the set of all languages is uncountable
    - let L be all languages over alphabet Σ
      - show L is uncountable using correspondence with B
      - let $\Sigma^* = \{s_1, s_2, s_3, \ldots\}$
      - each language A has a unique sequence in B
        - i^{th} bit is 1 if $s_i \in A$ and 0 otherwise
        - termed characteristic sequence of A

Undecidability

- some languages are not Turing-recognizable (cont.)
  - second, show the set of all languages is uncountable
    - e.g., if A were language of all strings beginning with 0 over alphabet {0,1}, its characteristic sequence would be

$$
\begin{array}{rcccccccccccl}
\Sigma^* = \{ & \varepsilon, & 0, & 1, & 00, & 01, & 10, & 11, & 000, & 001, & \cdots & \} ; \\
A = \{ & & 0, & & 00, & 01, & & & 000, & 001, & \cdots & \} ; \\
\chi_A = & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & \cdots & .
\end{array}
$$

    - the function f where f(A) = characteristic sequence of A is one-to-one and onto, and hence a correspondence
    - therefore, as B is uncountable, L is uncountable
  - set of all L cannot correspond to all TM
    - therefore, some languages are not recognized by any Turing machine

Undecidability

- now, we are ready to prove that A_{TM} is undecidable where A_{TM} = {<M,w> | M is a TM and M accepts w}
  - proof
    - assume A_{TM} is decidable and obtain a contradiction
    - suppose H is a decider for A_{TM}
      - on input <M,w>, H halts and accepts if M accepts w
      - H halts and rejects if M fails to accept w
      - therefore, H is a TM where

$$H(\langle M, w\rangle) = \begin{cases} \textit{accept} & \text{if } M \text{ accepts } w \\ \textit{reject} & \text{if } M \text{ does not accept } w. \end{cases}$$

Undecidability

- now, we are ready to prove that A_{TM} is undecidable where $A_{TM} = \{\langle M,w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$
  - proof (cont.)
    - construct new TM D with H as a subroutine
      - call H to determine what M does when the input to M is its own description
      - once D has determined this information, it does the opposite
        - rejects if M accepts
        - accepts if M does not accept
      - similar to running a program on with itself as input
        - e.g., an interpreter written in Python may be used on the interpreter

Undecidability

- now, we are ready to prove that A_{TM} is undecidable where A_{TM} = {<M,w> | M is a TM and M accepts w}
  - proof (cont.)
    - Turing machine D

      D = "On input <M>, where M is a TM:

      1. Run H on input <M, <M>>.
      2. Output the opposite of what H outputs. That is, if H accepts, reject; and if H rejects, accept."

Undecidability

- now, we are ready to prove that A_{TM} is undecidable where A_{TM} = {<M,w> | M is a TM and M accepts w}
  - proof (cont.)
    - in summary

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

    - what happens when we run D on <D>?

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

    - whatever D does, it is forced to do the opposite, which is a contradiction
    - therefore, neither TM D nor TM H can exist

Undecidability

- now, we are ready to prove that A_{TM} is undecidable where A_{TM} = {<M,w> | M is a TM and M accepts w}
  - proof (cont.)
    - steps of proof in summary
      - assume TM H decides
      - use H to build TM D that takes input <M>, where D accepts its input exactly when M does not accept its input <M>
      - run D on itself
    - machines take the following actions
      - H accepts <M,w> exactly when M accepts w
      - D rejects <M> exactly when M accepts <M>
      - D rejects <D> exactly when D accepts <D>

Undecidability

- now, we are ready to prove that A_{TM} is undecidable where A_{TM} = {<M,w> | M is a TM and M accepts w}
  - proof (cont.)
    - diagonalization comes into play in tables of behavior for TMs H and D
      - list all TMs down the rows, M_1, M_2, …
      - descriptions across the columns <M_1>, <M_2>, …
      - entries state whether machine in given row accepts input in given column
        - accept if accepts
        - blank if rejects or loops

Undecidability

- now, we are ready to prove that A_{TM} is undecidable where A_{TM} = {<M,w> | M is a TM and M accepts w}
  - proof (cont.)
    - e.g., with sample entries

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$
M_1	*accept*		*accept*		
M_2	*accept*	*accept*	*accept*	*accept*	
M_3					$\cdots$
M_4	*accept*	*accept*			
$\vdots$			$\vdots$		

    - results of running H on same inputs as above

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$
M_1	*accept*	*reject*	*accept*	*reject*	
M_2	*accept*	*accept*	*accept*	*accept*	$\cdots$
M_3	*reject*	*reject*	*reject*	*reject*	
M_4	*accept*	*accept*	*reject*	*reject*	
$\vdots$			$\vdots$		

Undecidability

- now, we are ready to prove that A_{TM} is undecidable
  where A_{TM} = {<M,w> | M is a TM and M accepts w}
  - proof (cont.)
    - now add D to table
      - both H and D are TMs
      - D computes the opposite of the diagonal entries
      - ? shows where contradiction occurs

	$\langle M_1\rangle$	$\langle M_2\rangle$	$\langle M_3\rangle$	$\langle M_4\rangle$	$\cdots$	$\langle D\rangle$	$\cdots$
M_1	*accept*	*reject*	*accept*	*reject*		*accept*	
M_2	*accept*	*accept*	*accept*	*accept*	$\cdots$	*accept*	$\cdots$
M_3	*reject*	*reject*	*reject*	*reject*		*reject*	
M_4	*accept*	*accept*	*reject*	*reject*		*accept*	
$\vdots$			$\vdots$		$\ddots$		
D	*reject*	*reject*	*accept*	*accept*		?	
$\vdots$			$\vdots$				$\ddots$

Undecidability

- we just showed that A_{TM} is undecidable
- is there a language that is not even Turing-recognizable?
  - can't use A_{TM} because we showed A_{TM} is Turing-recognizable
- if both a language and its complement are Turing-recognizable, the language is decidable
  - so, if any language or its complement is not Turing-recognizable, it is undecidable
    - recall that the complement of a language is language consisting of al strings that are not in the language
    - a language is co-Turing-recognizable if it is the complement of a Turing-recognizable language

Undecidability

- theorem: a language is decidable iff it is Turing-recognizable and co-Turing-recognizable
  - thus, a language is decidable exactly when both it and its complement are Turing-recognizable
  - proof
    - prove two directions
    - first: if *A* is decidable, both *A* and its complement are Turing-recognizable
      - any decidable language is Turing-recognizable
      - the complement of a decidable language is also decidable

Undecidability

- theorem: a language is decidable iff it is Turing-recognizable and co-Turing-recognizable
  - proof (cont.)
    - second: if both *A* and its complement are Turing-recognizable, let M_1 be the recognizer for *A* and M_2 be the recognizer for the complement of *A*
    - the following TM is a decider for *A*

      M = "On input *w*:

      1. Run both M_1 and M_2 on input *w* in parallel.
      2. If M_1 accepts, accept; if M_2 accepts, reject."
    - running the machines in parallel means *M* has two tapes: one for simulating M_1 and one for M_2
    - *M* takes turns simulating M_1 and M_2 until one accepts

Undecidability

- theorem: a language is decidable iff it is Turing-recognizable and co-Turing-recognizable
  - proof (cont.)
    - every string w is either in *A* or its complement
    - therefore, either M_1 or M_2 must accept w
    - M halts when M_1 or M_2 accepts
      - therefore, it is a decider
    - M accepts all strings in *A* and rejects all string not in *A*
      - therefore M is a decider for *A*
      - thus, *A* is decidable

Undecidability

- corollary: the complement of A_{TM} is not Turing-recognizable
  - we know A_{TM} is Turing-recognizable
  - if the complement of A_{TM} were Turing-recognizable, A_{TM} would be decidable
  - since A_{TM} is not decidable, the complement of A_{TM} must not be Turing-recognizable