Chapter 5
Reducibility

Overview

- previously, we established Turing machines as a model of general-purpose computing
- we presented several problems that were solvable on a TM and looked at one problem that was not solvable, A_{TM}
- in this chapter, we will look at other unsolvable problems
- we can prove problems are unsolvable using reducibility

Overview

- reduction: method of converting one problem into another such that the solution to the second problem can be used to solve the first
- we use reduction in everyday life
  - finding your way around in a new city is made easy by finding a map
    - reduce the problem of finding your way around to the problem of obtaining a map

Overview

- reducibility always involves two problems, A and B
  - if A reduces to B, we use a solution to B to solve A
  - e.g., A is the problem of finding your way around a city and B is the problem of finding a map
  - reducibility says nothing about solving A or B alone
    - only about the solvability of A in the presence of a solution to B

Overview

- further examples of reducibility
  - problem of traveling from Boston to Paris reduces to the problem of buying a plane ticket
  - that problem reduces to the problem of earning the money for the ticket
  - that problem reduces to finding a job

Overview

- reducibility is also used for mathematical problems
  - problem of measuring the area of a rectangle reduces to the problem of measuring its length and width
  - the problem of solving a system of linear equations reduces to the problem of inverting a matrix

Overview

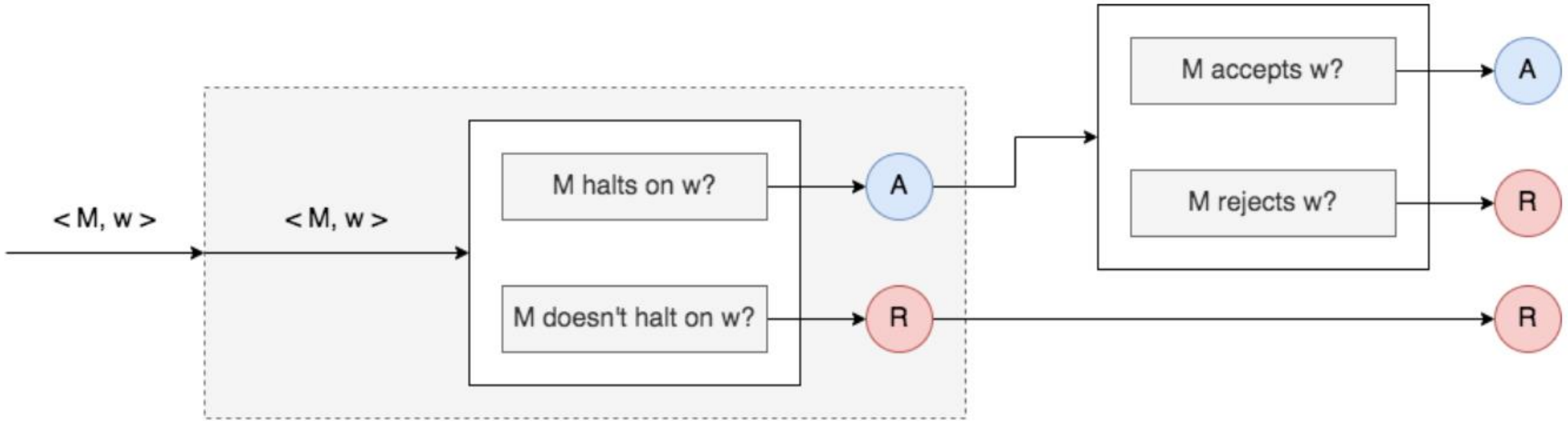
- reducibility helps in classifying problems by decidability
  - when A is reducible to B, solving A cannot be harder than solving B because a solution to B gives a solution to A
  - if A is reducible to B and B is decidable, then A is also decidable
  - if A is undecidable and reducible to B, then B is undecidable
    - we can use this to prove problems are undecidable
    - show that some other problem already known to be undecidable reduces to it

Undecidable Problems from Language Theory

- we already established the undecidability of A_{TM}, the problem of determining whether a Turing machine accepts a given input
  - through a fairly long proof
  - but now, we can use this result to show other problems are undecidable
    - e.g., the related problem $HALT_{TM}$

Undecidable Problems from Language Theory

- $HALT_{TM}$ is the problem of determining whether a TM halts by accepting or rejecting a given input
  - known as the halting problem
  - use the undecidability of A_{TM} to prove the undecidability of $HALT_{TM}$ by reducing A_{TM} to $HALT_{TM}$



http://thebeardsage.com/undecidable-language-halttm/

Undecidable Problems from Language Theory

- $HALT_{TM}$ = {<M,w> | M is a TM and M halts on input w}
- Theorem 5.1: $HALT_{TM}$ is undecidable
  - proof idea
    - use proof by contradiction
    - assume $HALT_{TM}$ is decidable to show A_{TM} is decidable, which contradicts our previous result
    - assume TM R decides $HALT_{TM}$
    - use R to construct S, a TM that decides A_{TM}
    - imagine you are S - how would you decide A_{TM}?

Undecidable Problems from Language Theory

- $HALT_{TM}$ = {<M,w> | M is a TM and M halts on input w}
- Theorem 5.1: $HALT_{TM}$ is undecidable (cont.)
  - proof idea
    - imagine you are S – how would you decide A_{TM}?
      - on input <M,w>
        - accept if M accepts
        - reject if M rejects or loops forever
      - simulate M on w
        - if it accepts or rejects, do the same
        - may not be able to tell if M is looping
        - simulation will not terminate – BAD for a decider
      - therefore, this idea doesn't work

Undecidable Problems from Language Theory

- $HALT_{TM}$ = {<M,w> | M is a TM and M halts on input w}
- Theorem 5.1: $HALT_{TM}$ is undecidable (cont.)
  - proof idea
    - instead, use TM R that decides $HALT_{TM}$
      - now test whether M halts on w
      - if R doesn't halt on w, reject because <M,w> not in A_{TM}
      - if R does halt on w, simulation can be performed as described without danger of looping
    - so, if TM R exists, we can decide A_{TM}
      - but we know A_{TM} is undecidable
    - by this contradiction, R cannot exist
    - therefore, $HALT_{TM}$ is undecidable

Undecidable Problems from Language Theory

- $HALT_{TM}$ = {<M,w> | M is a TM and M halts on input w}
- Theorem 5.1: $HALT_{TM}$ is undecidable (cont.)
  - proof
    - assume TM R decides $HALT_{TM}$
      - construct TM S to decide A_{TM}

        S = "On input <M,w>, an encoding of a TM M and string w:

        1. Run TM R on input <M,w>.
        2. If R rejects, reject.
        3. If R accepts, simulate M on w until it halts.
        4. If M accepted, accept; if M rejected, reject."
    - if R decides $HALT_{TM}$, S decides A_{TM}, but since A_{TM} is undecidable, $HALT_{TM}$ must also be undecidable

Undecidable Problems from Language Theory

- we can use this strategy to prove other problems are undecidable
  - common to proofs of undecidability, except for the undecidability of A_{TM} itself, which was proved directly using the diagonalization method

Undecidable Problems from Language Theory

- E_{TM} = {<M> | M is a TM and L(M) = Ø}
- Theorem 5.2: E_{TM} is undecidable
  - proof idea
    - follow same strategy as before
    - assume E_{TM} is decidable to show A_{TM} is decidable, which contradicts our previous result
    - assume TM R decides E_{TM}
    - use R to construct S, a TM that decides A_{TM}
    - how will S work when it receives input <M,w>?

Undecidable Problems from Language Theory

- E_{TM} = {<M> | M is a TM and L(M) = Ø}
- Theorem 5.2: E_{TM} is undecidable
  - proof idea (cont.)
    - S can run R on input <M> to see whether it accepts
      - if so, L(M) is empty and does not accept w
      - if R rejects <M>, all we know is L(M) is not empty and M accepts some string
        - but we still don't know if it accepts w
      - therefore, this idea doesn't work

Undecidable Problems from Language Theory

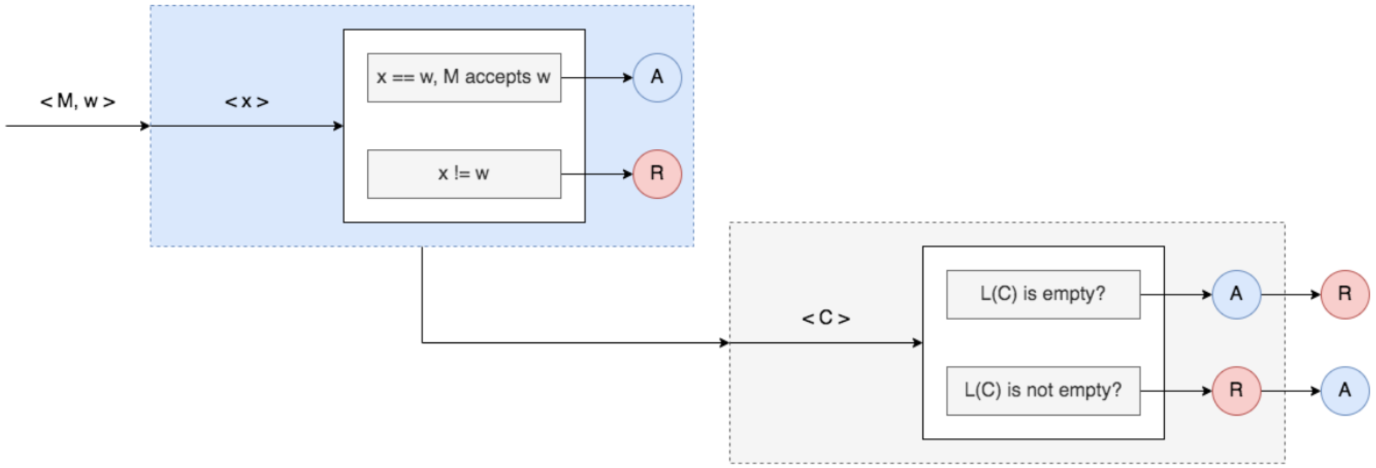
- E_{TM} = {<M> | M is a TM and L(M) = Ø}
- Theorem 5.2: E_{TM} is undecidable
  - proof idea (cont.)
    - instead of running R on <M>, run R on a modification of <M>
      - modify M so that it rejects all strings except w
      - on input w, it works as usual
      - run R to determine if the modified machine recognizes the empty language
        - the language will be nonempty iff it accepts w
      - if R accepts with the modified machine, the modified machine doesn't accept anything, so M doesn't accept w

Undecidable Problems from Language Theory

- E_{TM} = {<M> | M is a TM and L(M) = Ø}
- Theorem 5.2: E_{TM} is undecidable
  - proof
    - modified machine labeled M_1

      M_1 = "On input x:

      1. If x ≠ w, reject (compare character by character)
      2. If x = w, run M on input w and accept if M does."



http://thebeardsage.com/undecidable-language-etm/

Undecidable Problems from Language Theory

- E_{TM} = {<M> | M is a TM and L(M) = Ø}
- Theorem 5.2: E_{TM} is undecidable
  - proof
    - TM R decides E_{TM} and constructs S that decides A_{TM}

      S = "On input <M,w>, an encoding of a TM M and a
      string w:

      1. Use the description of M and w to construct the TM M_1 just as described.
      2. Run R on input $<M_1>$
      3. If R accepts, reject; if R rejects, accept."
    - S can compute a description of M_1 from a description of M and w – needs to add extra states to M to test x = w
    - if R were a decider for E_{TM}, S would be a decider for A_{TM}
      - a decider for A_{TM} cannot exist, so E_{TM} must be undecidable

Undecidable Problems from Language Theory

- another problem of interest is whether a TM can recognize a language that is recognized by a simpler computational model
  - e.g., $REGULAR_{TM}$
    - problem of determining whether a TM has an equivalent finite automaton
    - this is the same problem as determining whether a TM can recognize a regular language

Undecidable Problems from Language Theory

- $REGULAR_{TM}$ = {<M> | M is a TM and L(M) is a regular language}
- Theorem 5.3: $REGULAR_{TM}$ is undecidable
  - proof idea
    - follow same reduction from A_{TM} as before
    - assume $REGULAR_{TM}$ is decidable by TM R and use this result to show A_{TM} is decidable, which contradicts our previous result
    - how will S use R to do so?

Undecidable Problems from Language Theory

- $REGULAR_{TM}$ = {<M> | M is a TM and L(M) is a regular language}
- Theorem 5.3: $REGULAR_{TM}$ is undecidable
  - proof idea (cont.)
    - S takes its input <M,w> and modifies M to recognize a regular language iff M accepts w
      - new TM called M_2
      - design M_2 to recognize nonregular language $\{0^n1^n \mid n \geq 0\}$ if M does not accept w
      - and to recognize the regular language Σ^* if M accepts w
      - how will S construct M_2 from M and w?
        - M_2 will automatically accept all strings 0^n1^n
        - if M accepts w, M_2 accepts all other strings

Undecidable Problems from Language Theory

- $REGULAR_{TM}$ = {<M> | M is a TM and L(M) is a regular language}
- Theorem 5.3: $REGULAR_{TM}$ is undecidable
    - proof idea (cont.)
        - note that M_2 is not constructed to actually run on some input
            - only use it to feed its description into the decider for $REGULAR_{TM}$ that we have assumed to exist
            - once this decider returns its answer, we can use it to determine if M accepts w
            - thus, we can decide A_{TM}, a contradiction

Undecidable Problems from Language Theory

- $REGULAR_{TM}$ = {<M> | M is a TM and L(M) is a regular language}
- Theorem 5.3: $REGULAR_{TM}$ is undecidable
  - proof
    - let R be a TM that decides $REGULAR_{TM}$ and construct S to decide A_{TM}

      S = "On input <M,w>, an encoding of a TM M and a string w:

      1. Construct the following TM M_2.

         M_2 = "On input x:

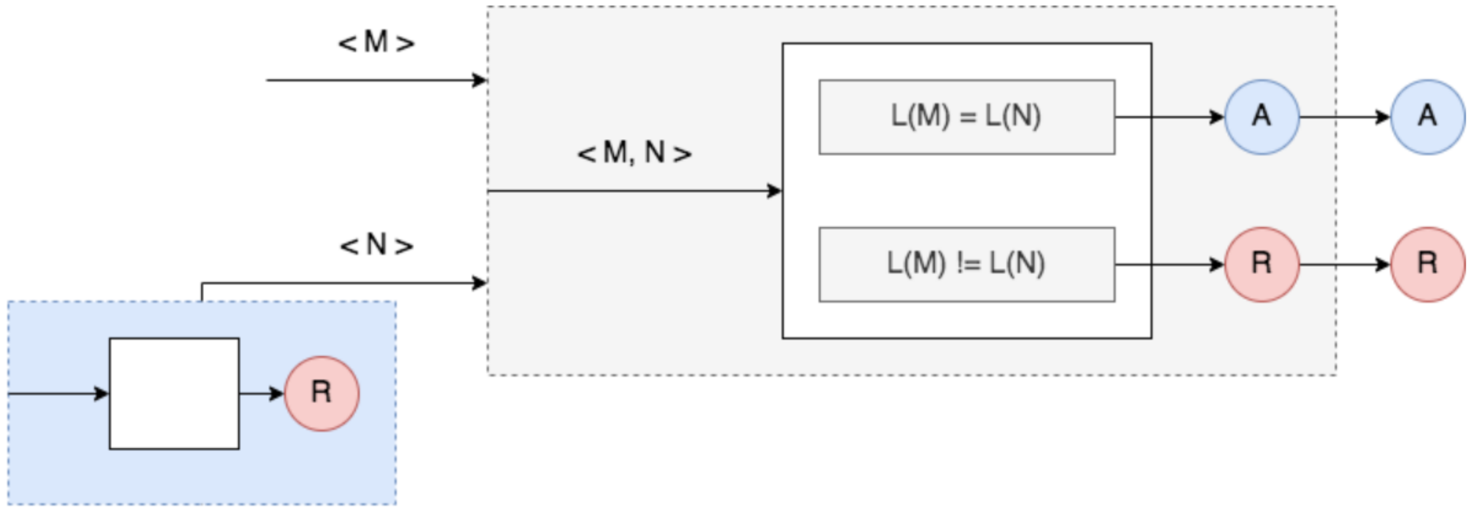
         1. If x has the form 0^n1^n, accept.
         2. If x does not have this form, run M on input w and accept if M accepts w.
      2. Run R on input <M_2>
      3. If R accepts, accept; if R rejects, reject."

Undecidable Problems from Language Theory

- similarly, the following problems can be shown to be undecidable
  - whether the language of a TM is context-free
  - whether the language of a TM is decidable
  - whether the language of a TM is finite
- Rice's Theorem: Showing any property of the languages recognized by a TM is undecidable
- so far, we've used the reduction to A_{TM} as our strategy
- can also reduce from other undecidable languages, such as E_{TM}

Undecidable Problems from Language Theory

- next, we'll prove the testing the equivalence of two TMs is an undecidable problem
  - we could use reduction to A_{TM} to prove this, but we'll use reduction to E_{TM} instead



http://thebeardsage.com/undecidable-language-eqtm/

Undecidable Problems from Language Theory

- $EQ_{TM} = \{\langle M_1, M_2\rangle \mid M_1$ and M_2 are TMs and $L(M_1) = L(M_2)\}$
- Theorem 5.4: EQ_{TM} is undecidable
  - proof idea
    - show that if EQ_{TM} were decidable, then E_{TM} would be decidable by using a reduction from E_{TM} to EQ_{TM}
    - recall that E_{TM} is the problem of determining if M is empty
    - if one of the languages happens to be Ø, our problem becomes determining if the other language is empty
    - in this way, the E_{TM} problem is a special case of the EQ_{TM} problem
      - this makes the reduction easy

Undecidable Problems from Language Theory

- $EQ_{TM} = \{\langle M_1, M_2\rangle \mid M_1$ and M_2 are TMs and $L(M_1) = L(M_2)\}$
- Theorem 5.4: EQ_{TM} is undecidable
  - proof
    - let R be a TM that decides EQ_{TM} and construct TM S to decide E_{TM}

      S = "On input ⟨M⟩, where M is a TM:

      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."

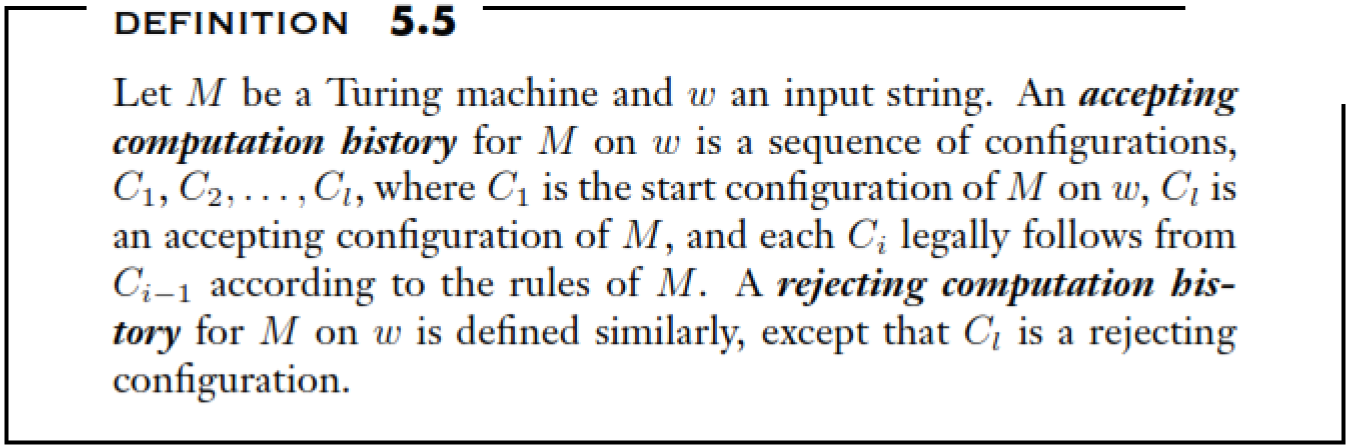
    - if R decides EQ_{TM}, S decides E_{TM}
      - but E_{TM} is undecidable, so EQ_{TM} must also be undecidable

Undecidable Problems from Language Theory

- restrictions via computation histories
  - the computation history method is an important technique for proving A_{TM} is reducible to certain languages
  - useful when the problem to be shown to be undecidable involves testing for the existence of something
    - e.g., Hilbert's tenth problem for testing for integral roots of a polynomial

Undecidable Problems from Language Theory

- restrictions via computation histories
  - the computation history for a TM on an input is just the sequence of configurations that the machine goes through as it processes the input
    - a complete record of the computation of the machine



DEFINITION 5.5

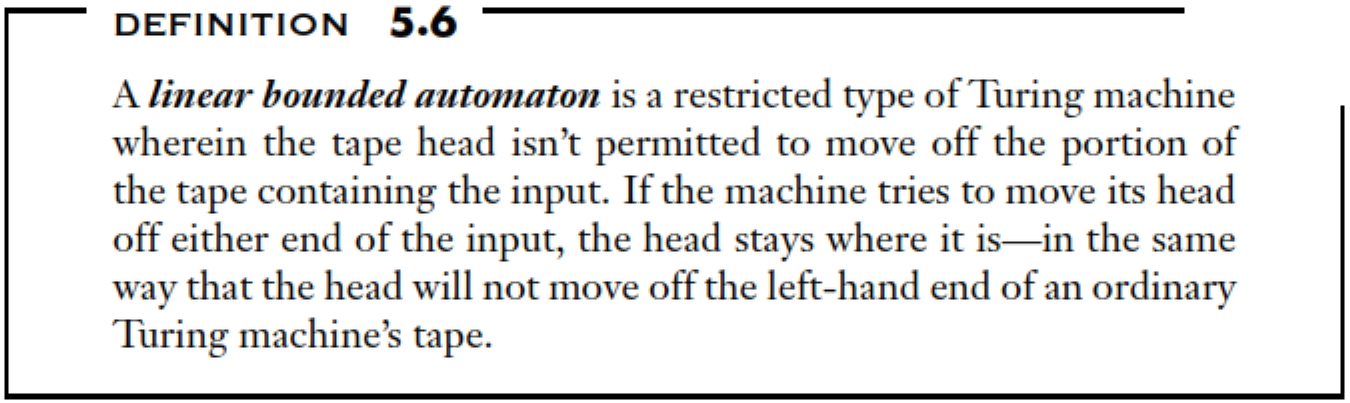
Let M be a Turing machine and w an input string. An ***accepting computation history*** for M on w is a sequence of configurations, $C_1, C_2, \ldots, C_l$, where C_1 is the start configuration of M on w, C_l is an accepting configuration of M, and each C_i legally follows from C_{i-1} according to the rules of M. A ***rejecting computation history*** for M on w is defined similarly, except that C_l is a rejecting configuration.

Undecidable Problems from Language Theory

- restrictions via computation histories
  - computation histories are finite sequences
  - if M doesn't halt on w, no accepting or rejecting computation history exists for M on w
  - deterministic machines have at most one computation history for any given input
  - nondeterministic machines may have many computation histories for a single input
  - we'll focus on deterministic machines

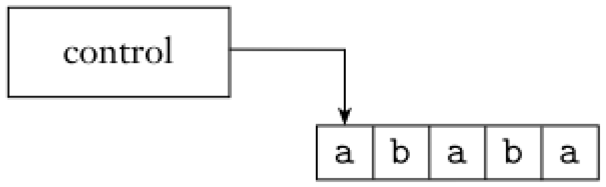
Undecidable Problems from Language Theory

- our first undecidability proof using computational history is called a linear bounded automaton



DEFINITION 5.6

A ***linear bounded automaton*** is a restricted type of Turing machine wherein the tape head isn't permitted to move off the portion of the tape containing the input. If the machine tries to move its head off either end of the input, the head stays where it is—in the same way that the head will not move off the left-hand end of an ordinary Turing machine's tape.



Undecidable Problems from Language Theory

- a linear bounded automaton is a TM with a limited amount of memory
  - can only solve problems requiring memory that can fit within the tape used for input
  - using a tape alphabet larger than the input alphabet allows available memory to be increased up by a constant factor
  - for input length n, the amount of available memory is linear in n

Undecidable Problems from Language Theory

- despite these limitations, linear bounded automata (LBAs) are quite powerful
  - deciders for A_{DFA}, A_{CFG}, E_{DFA}, and E_{CFG}, are all LBAs
  - every CFL can be decided by an LBA
    - trying to find a decidable language that can't be decided by an LBA takes work
  - A_{LBA} is the problem of determining whether an LBA accepts its input
    - even though same as the undecidable problem A_{TM} where the TM is restricted to an LBA, we show that A_{LBA} is decidable

Undecidable Problems from Language Theory

- A_{LBA} = {<M,w> | M is an LBA that accepts string w}
- the following lemma will be useful
- Lemma 5.8: Let M be an LBA with q states and g symbols in the tape alphabet. There are exactly qng^n distinct configurations of M for a tape of length n.
  - proof
    - a configuration for M is a snapshot of computation
    - a configuration consists of the state of control, position of the head, and contents of the tape
    - M has q states
    - tape length n with M on one of those n positions
    - g^n possible strings of tape symbols on the tape
    - multiplying these gives total number of configurations

Undecidable Problems from Language Theory

- A_{LBA} = {<M,w> | M is an LBA that accepts string w}
- Theorem 5.9: A_{LBA} is decidable.
  - proof idea
    - simulate M on w
      - if M halts and accepts or rejects, we accept or reject accordingly
      - difficulty is when M loops on w
        - we need to be able to detect looping so that we can halt and reject

Undecidable Problems from Language Theory

- A_{LBA} = {<M,w> | M is an LBA that accepts string w}
- Theorem 5.9: A_{LBA} is decidable.
  - proof idea (cont.)
    - to detect looping
      - as M computes on w, it goes from configuration to configuration
      - if M ever repeats a configuration again and again, we have detected a loop
      - because M is an LBA, the amount of tape available is limited
        - Lemma 5.8 states M can only be in a limited number of configurations
        - therefore, limited amount of time before M can enter some configuration previously entered
        - so, simulate M for number of steps given in lemma
        - if M has not halted by then, it must be looping

Undecidable Problems from Language Theory

- A_{LBA} = {<M,w> | M is an LBA that accepts string w}
- Theorem 5.9: A_{LBA} is decidable.
  - proof

    L = "On input <M,w>, where M is an LBA and w is a string:

    1. Simulate M on w for qng^n steps or until it halts.
    2. If M has halted, accept if it has accepted and reject if it has rejected. If it has not halted, reject."

  - if M has not halted within qng^n steps, it must be repeating a configuration and therefore looping
    - the algorithm therefore rejects in this situation

Undecidable Problems from Language Theory

- this result shows that LBAs and TMs differ in one essential way
  - for LBAs, the acceptance problem is decidable, but for TMs, it isn't
- certain other problems involving LBAs are undecidable
  - emptiness problem E_{LBA}

A Simple Undecidable Problem

- undecidability is not confined to problems concerning automata
- one example is the Post Correspondence problem (PCP)
  - string manipulation problem

A Simple Undecidable Problem

- Post Correspondence problem (PCP)
  - puzzle problem with a collection of dominos
  - each domino contains two strings: one on each side
    - example of individual domino

$$\left[\frac{\mathtt{a}}{\mathtt{ab}}\right]$$

    - collection of dominos

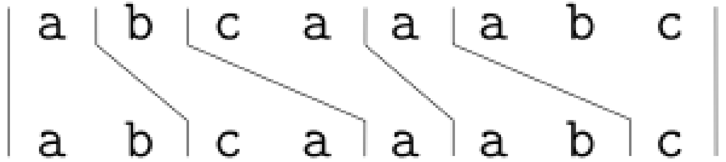
$$\left\{\left[\frac{\mathtt{b}}{\mathtt{ca}}\right], \left[\frac{\mathtt{a}}{\mathtt{ab}}\right], \left[\frac{\mathtt{ca}}{\mathtt{a}}\right], \left[\frac{\mathtt{abc}}{\mathtt{c}}\right]\right\}$$

A Simple Undecidable Problem

- Post Correspondence problem (PCP)
  - task: make a list of these dominos (repetitions OK) so that the string along the top is the same as the symbols on the bottom
    - termed a match

$$\left[\frac{\mathtt{a}}{\mathtt{ab}}\right]\left[\frac{\mathtt{b}}{\mathtt{ca}}\right]\left[\frac{\mathtt{ca}}{\mathtt{a}}\right]\left[\frac{\mathtt{a}}{\mathtt{ab}}\right]\left[\frac{\mathtt{abc}}{\mathtt{c}}\right]$$

      - top string: abcaaabc
      - bottom string: same
  - another way to show the match (deform dominos to line up match)



A Simple Undecidable Problem

- Post Correspondence problem (PCP)
  - for some collections of dominos, finding a match may not be possible

$$\left\{ \left[\frac{\mathrm{abc}}{\mathrm{ab}}\right], \left[\frac{\mathrm{ca}}{\mathrm{a}}\right], \left[\frac{\mathrm{acc}}{\mathrm{ba}}\right] \right\}$$

    - this collection cannot contain a match because every top string is longer than its corresponding bottom string

A Simple Undecidable Problem

- Post Correspondence problem (PCP)
  - the problem is to determine whether a collection of dominos has a match
    - this problem is unsolvable by algorithms

A Simple Undecidable Problem

- Post Correspondence problem (PCP)
  - first, we'll state the problem precisely and express it as a language
    - an instance is a collection P of dominos

$$P = \left\{ \left[\frac{t_1}{b_1}\right], \left[\frac{t_2}{b_2}\right], \ldots, \left[\frac{t_k}{b_k}\right] \right\}$$

    - a match is a sequence

      $i_1, i_2, \ldots, i_l$ where $t_{i1}, t_{i2}, \ldots, t_{il} = b_{i1}, b_{i2}, \ldots, b_{il}$
    - the problem is to determine whether P has a match

PCP = {<P> | P is an instance of the PCP with a match}

A Simple Undecidable Problem

- Post Correspondence problem (PCP)

  PCP = {<P> | P is an instance of the PCP with a match}
- Theorem 5.15: PCP is undecidable
  - proof idea (cont.)
    - how do we construct P so that a match is an accepting computation history for M on w?
      - choose dominos in P so that making a match forces a simulation of M to occur
      - each domino links a position or positions in one configuration with the corresponding one in the next configuration

A Simple Undecidable Problem

- Post Correspondence problem (PCP)

  PCP = {<P> | P is an instance of the PCP with a match}
- Theorem 5.15: PCP is undecidable
  - proof idea (cont.)
    - three small technical points
      1. assume M on w never attempts to move its head off the left-hand end of the tape
         - must alter M to prevent this behavior
      2. if w = ε, use _ (␣) in place of w in the construction
      3. modify PCP to require that match starts with the first domino (eliminate this requirement later)

$$\left[\frac{t_1}{b_1}\right]$$

A Simple Undecidable Problem

- Modified Post Correspondence problem (MPCP)

  MPCP = {<P> | P is an instance of the PCP with a match that starts with the first domino}
- Theorem 5.15: PCP is undecidable
  - proof
    - let TM R decide PCP and construct S deciding A_{TM}

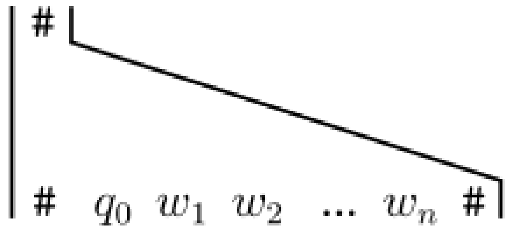
      $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$
    - S constructs an instance of the PCP P that has a match iff M accepts w
    - S first constructs P' of the MPCP
      - described in 7 parts, each of which covers a particular aspect of simulating M on w

A Simple Undecidable Problem

- Post Correspondence problem (PCP)

  PCP = {<P> | P is an instance of the PCP with a match}
- Theorem 5.15: PCP is undecidable
  - proof (cont.)
    - 7 parts of simulation (through example)
      1. put $\left[\frac{\#}{\#q_0w_1w_2\cdots w_n\#}\right]$ into P′ as the first domino $\left[\frac{t_1}{b_1}\right]$

         - the match must begin with this domino
         - the bottom string begins with the first configuration in the accepting history for M on w



A Simple Undecidable Problem

- Post Correspondence problem (PCP)

  PCP = {<P> | P is an instance of the PCP with a match}
- Theorem 5.15: PCP is undecidable
  - proof (cont.)
    - 7 parts of simulation (through example) (cont.)
      1. partial match achieved so far
         - to get a match, need to extend top string to match bottom string
           - use additional dominos
           - cause M's next configuration to appear at the extension of the bottom by forcing a single-step simulation of M

A Simple Undecidable Problem

- Post Correspondence problem (PCP)

  PCP = {<P> | P is an instance of the PCP with a match}
- Theorem 5.15: PCP is undecidable
  - proof (cont.)
    - 7 parts of simulation (through example) (cont.)
      - next three steps perform simulation
        - part 2 handles head motions to the right
        - part 3 handles head motions to the left
        - part 4 handles tape cells not adjacent to the head

A Simple Undecidable Problem

- Post Correspondence problem (PCP)

  PCP = {<P> | P is an instance of the PCP with a match}
- Theorem 5.15: PCP is undecidable
  - proof (cont.)
    - 7 parts of simulation (through example) (cont.)

      2. for every a,b ∈ Γ and q,r ∈ Q where $q \neq q_{reject}$
         - if δ(q,a) = (r,b,R), put $\left[\frac{qa}{br}\right]$ into P′

A Simple Undecidable Problem

- Post Correspondence problem (PCP)

  PCP = {<P> | P is an instance of the PCP with a match}
- Theorem 5.15: PCP is undecidable
  - proof (cont.)
    - 7 parts of simulation (through example) (cont.)
      3. for every a,b,c ∈ Γ and q,r ∈ Q where $q \neq q_{reject}$
         - if δ(q,a) = (r,b,L), put $\left[\frac{cqa}{rcb}\right]$ into P′

A Simple Undecidable Problem

- Post Correspondence problem (PCP)

  PCP = {<P> | P is an instance of the PCP with a match}
- Theorem 5.15: PCP is undecidable
  - proof (cont.)
    - 7 parts of simulation (through example)

      4. for every a ∈ Γ
         - put $\left[\frac{a}{a}\right]$ into P′

A Simple Undecidable Problem

- Post Correspondence problem (PCP)

  PCP = {<P> | P is an instance of the PCP with a match}
- Theorem 5.15: PCP is undecidable
  - proof (cont.)
    - 7 parts of simulation (through example)
      - example of process so far
      - let $\Gamma = \{0, 1, 2, \_\}$
      - w is string 0100
      - start state is q_0
        - upon reading 0, M enters state q_7, writes a 2 on the tape and moves its head to the right

          $\delta(q_0,0) = (q_7,2,R)$

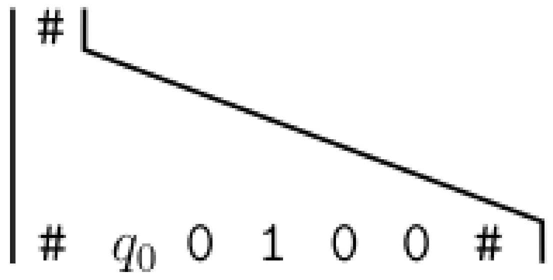
A Simple Undecidable Problem

- Post Correspondence problem (PCP)

  PCP = {<P> | P is an instance of the PCP with a match}
- Theorem 5.15: PCP is undecidable
  - proof (cont.)
    - 7 parts of simulation (through example)
      - part 1 places the following domino in P'

$$\left[\frac{\#}{\#q_0 0100\#}\right] = \left[\frac{t_1}{b_1}\right]$$

      - and the match begins



A Simple Undecidable Problem

- Post Correspondence problem (PCP)

  PCP = {<P> | P is an instance of the PCP with a match}
- Theorem 5.15: PCP is undecidable
  - proof (cont.)
    - 7 parts of simulation (through example)
      - part 2 places the following domino in P'

$$\left[\frac{q_0 0}{2 q_7}\right]$$

      - since $\delta(q_0,0) = (q_7,2,R)$

A Simple Undecidable Problem

- Post Correspondence problem (PCP)

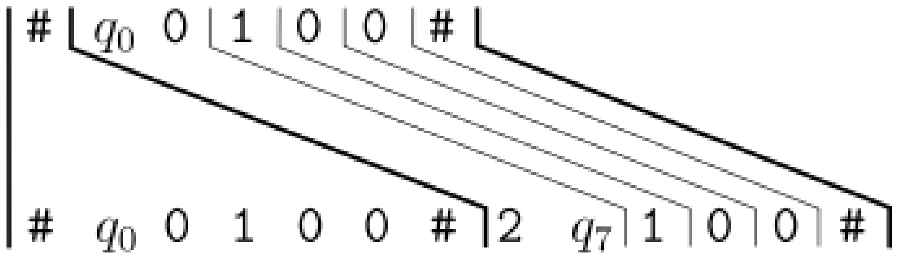
  PCP = {<P> | P is an instance of the PCP with a match}
- Theorem 5.15: PCP is undecidable
  - proof (cont.)
    - 7 parts of simulation (through example)
      - part 4 places the following dominos in P'

        $$\left[\frac{0}{0}\right], \left[\frac{1}{1}\right], \left[\frac{2}{2}\right], \text{ and } \left[\frac{\sqcup}{\sqcup}\right]$$
      - since 0, 1, 2, and _ are members of Γ

A Simple Undecidable Problem

- Post Correspondence problem (PCP)

  PCP = {<P> | P is an instance of the PCP with a match}
- Theorem 5.15: PCP is undecidable
  - proof (cont.)
    - 7 parts of simulation (through example)
      - with part 5, the match is extended to



      - thus, the dominos of parts 2, 3, and 4 let us extend the match by adding the second configuration after the first, then adding the third, the fourth, etc.
        - need to add a domino for copying the # symbol

A Simple Undecidable Problem

- Post Correspondence problem (PCP)

  PCP = {<P> | P is an instance of the PCP with a match}

- Theorem 5.15: PCP is undecidable
  - proof (cont.)
    - 7 parts of simulation (through example)

      5. put the following into P'

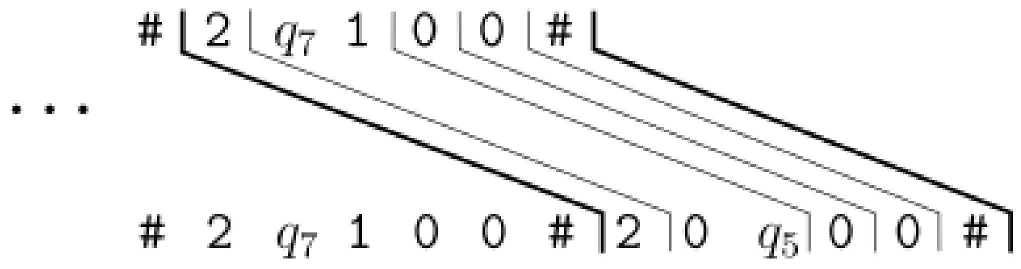
$$\left[\frac{\#}{\#}\right] \qquad \left[\frac{\#}{\sqcup\#}\right]$$

- the first allows us to copy the # symbol that marks the separation of the configurations
- the second allows us to add a blank symbol at the end to simulate the infinite blanks to the right that are otherwise suppressed

A Simple Undecidable Problem

- Post Correspondence problem (PCP)

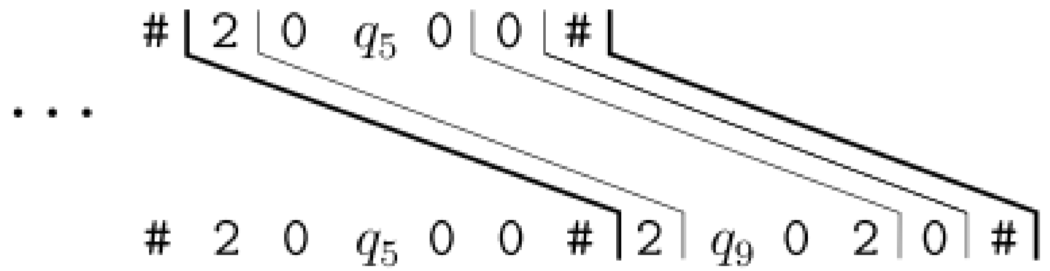
  PCP = {<P> | P is an instance of the PCP with a match}
- Theorem 5.15: PCP is undecidable
  - proof (cont.)
    - 7 parts of simulation (through example)
      - in the example, let's say we have $\delta(q_7,1) = (q_5,0,R)$
      - put the following in P': $\left[\frac{q_7 1}{0 q_5}\right]$
      - partial match extends to



A Simple Undecidable Problem

- Post Correspondence problem (PCP)

  PCP = {<P> | P is an instance of the PCP with a match}
- Theorem 5.15: PCP is undecidable
  - proof (cont.)
    - 7 parts of simulation (through example)
      - now suppose we have δ(q_5,0) = (q_9,2,L)
      - put the following in P′: $\left[\frac{0q_50}{q_902}\right]$, $\left[\frac{1q_50}{q_912}\right]$, $\left[\frac{2q_50}{q_922}\right]$, and $\left[\frac{\sqcup q_50}{q_9\sqcup 2}\right]$
      - first one has 0 L of head; partial match extends to



A Simple Undecidable Problem

- Post Correspondence problem (PCP)

  PCP = {<P> | P is an instance of the PCP with a match}
- Theorem 5.15: PCP is undecidable
  - proof (cont.)
    - 7 parts of simulation (through example)
      - as we construct a match, we are forced to simulate M on input w
      - process continues until M reaches a halting state
      - if accept, the top part of the partial match must catch up with the bottom to complete the match
        - need to add more dominos

A Simple Undecidable Problem

- Post Correspondence problem (PCP)

  PCP = {<P> | P is an instance of the PCP with a match}
- Theorem 5.15: PCP is undecidable
  - proof (cont.)
    - 7 parts of simulation (through example)

      6. for every a ∈ Γ, put the following into P′

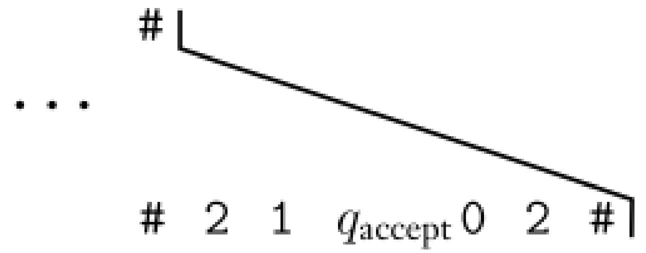
      $$\left[\frac{a\, q_{\text{accept}}}{q_{\text{accept}}}\right] \text{ and } \left[\frac{q_{\text{accept}}\, a}{q_{\text{accept}}}\right]$$

         - adds pseudo-steps of TM after it has halted
         - the head eats symbols until none are left

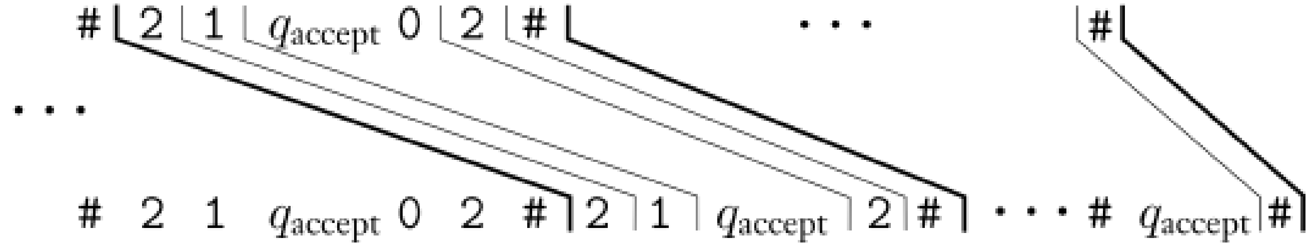
A Simple Undecidable Problem

- Post Correspondence problem (PCP)

  PCP = {<P> | P is an instance of the PCP with a match}
- Theorem 5.15: PCP is undecidable
  - proof (cont.)
    - 7 parts of simulation (through example)
      - the partial match up to when TM halts is



      - with the added dominos, the match continues


# 2 1 q_{accept} 0 2 # ... #
...
# 2 1 q_{accept} 0 2 # 2 1 q_{accept} 2 # ... # q_{accept} #

A Simple Undecidable Problem

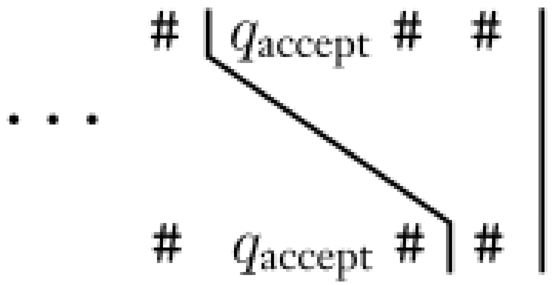
- Post Correspondence problem (PCP)

  PCP = {<P> | P is an instance of the PCP with a match}
- Theorem 5.15: PCP is undecidable
  - proof (cont.)
    - 7 parts of simulation (through example)

      7. finally, we add the domino

      $$\left[\frac{q_{\text{accept}}\#\#}{\#}\right]$$

         - and complete the match



A Simple Undecidable Problem

- Modified Post Correspondence problem (MPCP)

  MPCP = {<P> | P is an instance of the PCP with a match
  that starts with the first domino}
- Theorem 5.15: PCP is undecidable
  - proof (cont.)
    - thus, construction of P′ is complete
    - P′ is really just an instance of PCP instead of MPCP
    - to convert P′ to P
      - take the requirement that the match starts with the first domino and build it directly into the problem itself so that it becomes enforced automatically

A Simple Undecidable Problem

- Post Correspondence problem (PCP)

  PCP = {<P> | P is an instance of the PCP with a match}
- Theorem 5.15: PCP is undecidable
  - proof (cont.)
    - let u = $u_1u_2\ldots u_n$ be any string of length n
    - define *u, u*, and *u* to be

$$\begin{aligned} \star u &= * u_1 * u_2 * u_3 * \quad \cdots \quad * u_n \\ u\star &= u_1 * u_2 * u_3 * \quad \cdots \quad * u_n * \\ \star u \star &= * u_1 * u_2 * u_3 * \quad \cdots \quad * u_n * \end{aligned}$$

      - *u adds the * before every character in u
      - u* adds one after each character in u
      - and *u* adds one both before and after every character in u

A Simple Undecidable Problem

- Post Correspondence problem (PCP)

  PCP = {<P> | P is an instance of the PCP with a match}
- Theorem 5.15: PCP is undecidable
  - proof (cont.)
    - to convert P' to P, if P' is the collection

$$\left\{ \left[\frac{t_1}{b_1}\right], \left[\frac{t_2}{b_2}\right], \left[\frac{t_3}{b_3}\right], \ldots, \left[\frac{t_k}{b_k}\right] \right\}$$

    - we let P be the collection

$$\left\{ \left[\frac{\star t_1}{\star b_1 \star}\right], \left[\frac{\star t_1}{b_1 \star}\right], \left[\frac{\star t_2}{b_2 \star}\right], \left[\frac{\star t_3}{b_3 \star}\right], \ldots, \left[\frac{\star t_k}{b_k \star}\right], \left[\frac{\ast\diamond}{\diamond}\right] \right\}$$

    - since P is an instance of PCP, the only domino that could possibly start a match is the first one $\left[\frac{\star t_1}{\star b_1 \star}\right]$

A Simple Undecidable Problem

- Post Correspondence problem (PCP)

  PCP = {<P> | P is an instance of the PCP with a match}
- Theorem 5.15: PCP is undecidable
  - proof (cont.)
    - since P is an instance of PCP, the only domino that could possibly start a match is the first one $\left[\frac{\star t_1}{\star b_1 \star}\right]$
      - only one where both top and bottom start with same symbol, *
      - the *s don't affect matches because they interleave with the original symbols, which now occur in the even positions
      - domino $\left[\frac{*\diamond}{\diamond}\right]$ allows top to add extra * at end of match