More Undecidable Problems

Rice's Theorem

Post's Correspondence Problem

Some Real Problems

Properties of Languages

- Any set of languages is a *property* of languages.
- Example: The infiniteness property is the set of infinite languages.
- In what follows, we'll focus on properties of RE languages, because we can't represent other languages by TM's.

Properties of Langauges – (2)

- Thus, we shall think of a property as a problem about Turing machines.
- Let L_P be the set of binary TM codes for TM's M such that L(M) has property P.

Trivial Properties

- There are two (*trivial*) properties P for which L_P is decidable.
  1. The *always-false property*, which contains no RE languages.
  2. The *always-true property*, which contains every RE language.
- Rice's Theorem: For every other property P, L_P is undecidable.

Reductions

- A *reduction* from language L to language L' is an algorithm (TM that always halts) that takes a string w and converts it to a string x, with the property that:

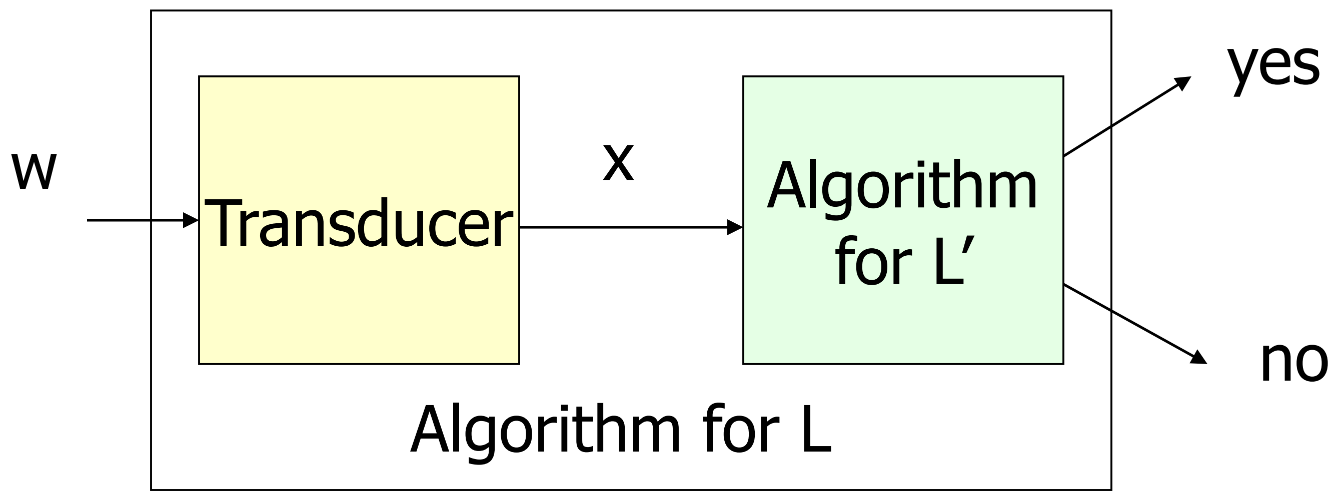
  x is in L' if and only if w is in L.

TM's as *Transducers*

- We have regarded TM's as acceptors of strings.
- But we could just as well visualize TM's as having an *output tape*, where a string is written prior to the TM halting.

Reductions – (2)

- If we reduce L to L′, and L′ is decidable, then the algorithm for L′ + the algorithm of the reduction shows that L is also decidable.



Reductions – (3)

- Normally used in the contrapositive.
- If we know L is not decidable, then L' cannot be decidable.

Reductions – Aside

- This form of reduction is not the most general.
- Example: We “reduced” L_d to L_u, but in doing so we had to complement answers.
- More in NP-completeness discussion on Karp vs. Cook reductions.

Proof of Rice's Theorem

- We shall show that for every nontrivial property P of the RE languages, L_P is undecidable.
- We show how to reduce L_u to L_P.
- Since we know L_u is undecidable, it follows that L_P is also undecidable.

The Reduction

- Our reduction algorithm must take M and w and produce a TM M′.
- L(M′) has property P if and only if M accepts w.
- M′ has two tapes, used for:
  1. Simulates another TM M_L on the input to M′.
  2. Simulates M on w.
     - Note: neither M, M_L, nor w is input to M′.

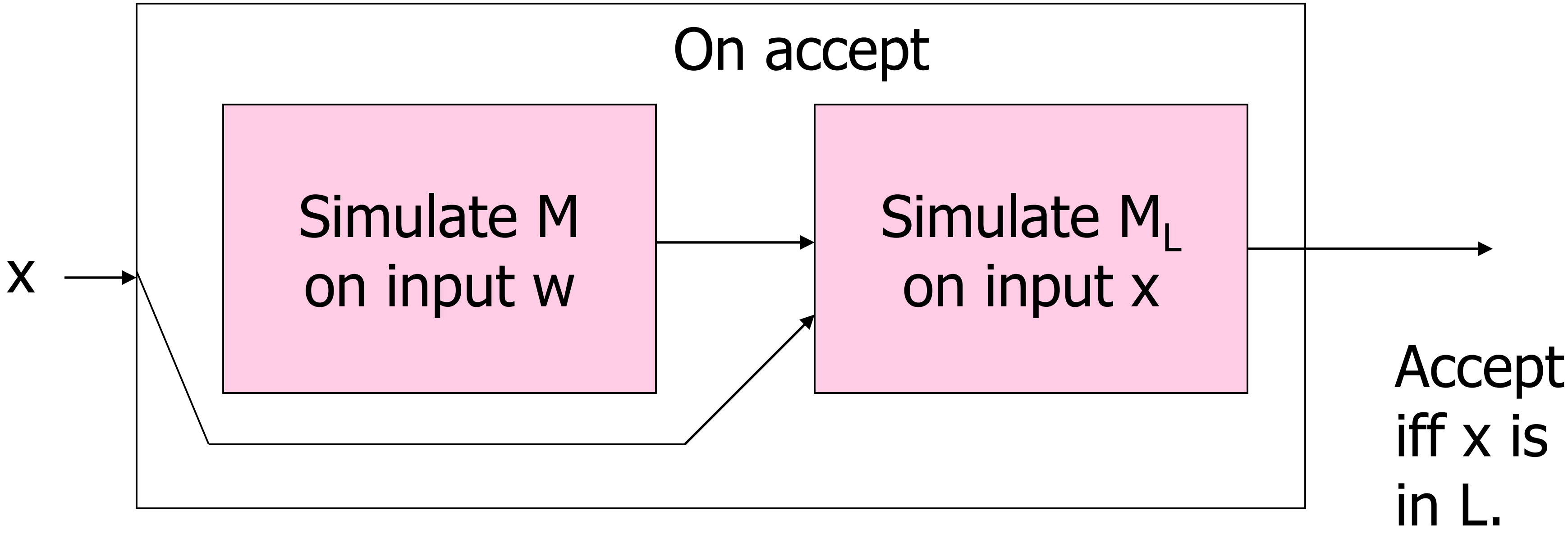
The Reduction – (2)

- Assume that $\emptyset$ does not have property P.
  - If it does, consider the complement of P, which would also be decidable if P were, because the recursive languages are closed under complementation.
- Let L be any language with property P, and let M_L be a TM that accepts L.

Design of M’

1. On the second tape, write w and then simulate M on w.
2. If M accepts w, then simulate M_L on the input x to M’, which appears initially on the first tape.
3. M’ accepts its input x if and only if M_L accepts x.

Action of M' if M Accepts w



Design of M’ – (2)

- Suppose M accepts w.
- Then M’ simulates M_L and therefore accepts x if and only if x is in L.
- That is, L(M’) = L, L(M’) has property P, and M’ is in L_P.

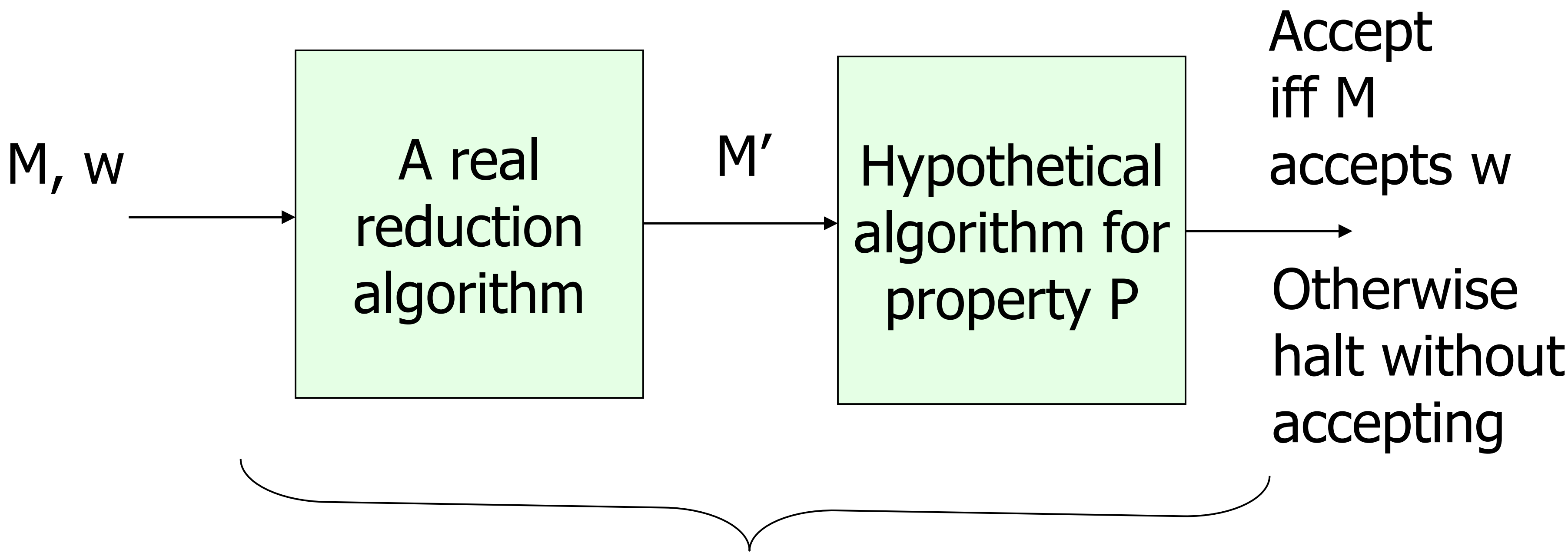
Design of M’ – (3)

- Suppose M does not accept w.
- Then M’ never starts the simulation of M_L, and never accepts its input x.
- Thus, L(M’) = ∅, and L(M’) does not have property P.
- That is, M’ is not in L_P.

Design of M' – Conclusion

- Thus, the algorithm that converts M and w to M' is a reduction of L_u to L_P.
- Thus, L_P is undecidable.

Picture of the Reduction



This would be an algorithm for L_u, which doesn't exist

Applications of Rice's Theorem

- We now have any number of undecidable questions about TM's:
  - Is L(M) a regular language?
  - Is L(M) a CFL?
  - Does L(M) include any palindromes?
  - Is L(M) empty?
  - Does L(M) contain more than 1000 strings?
  - Etc., etc.

Post's Correspondence Problem

- *Post's Correspondence Problem* (PCP) is an example of a problem that does not mention TM's in its statement, yet is undecidable.
- From PCP, we can prove many other non-TM problems undecidable.

PCP Instances

- An instance of PCP is a list of pairs of nonempty strings over some alphabet Σ.
  - Say $(w_1, x_1), (w_2, x_2), \ldots, (w_n, x_n)$.
- The answer to this instance of PCP is “yes” if and only if there exists a nonempty sequence of indices $i_1,\ldots,i_k$, such that $w_{i1}\ldots w_{in} = x_{i1}\ldots x_{in}$.

Example: PCP

- Let the alphabet be {0, 1}.
- Let the PCP instance consist of the two pairs (0, 01) and (100, 001).
- We claim there is no solution.
- You can’t start with (100, 001), because the first characters don’t match.

Example: PCP – (2)

Recall: pairs are (0, 01) and (100, 001)

0 100 100
01 001 001

But we can never make the first string as long as the second.

Must start with first pair

Can add the second pair for a match

As many times as we like

Example: PCP – (3)

- Suppose we add a third pair, so the instance becomes: 1 = (0, 01); 2 = (100, 001); 3 = (110, 10).
- Now 1,3 is a solution; both strings are 0110.
- In fact, any sequence of indexes in **12*****3** is a solution.

Proving PCP is Undecidable

- We'll introduce the *modified* PCP (MPCP) problem.
  - Same as PCP, but the solution must start with the first pair in the list.
- We reduce L_u to MPCP.
- But first, we'll reduce MPCP to PCP.

Example: MPCP

- The list of pairs (0, 01), (100, 001), (110, 10), as an instance of MPCP, has a solution as we saw.
- However, if we reorder the pairs, say (110, 10), (0, 01), (100, 001) there is no solution.
  - No string 110… can ever equal a string 10… .

Representing PCP or MPCP Instances

- Since the alphabet can be arbitrarily large, we need to code symbols.
- Say the i-th symbol will be coded by "a" followed by i in binary.
- Commas and parentheses can represent themselves.

Representing Instances – (2)

- Thus, we have a finite alphabet in which all instances of PCP or MPCP can be represented.
- Let L_{PCP} and L_{MPCP} be the languages of coded instances of PCP or MPCP, respectively, that have a solution.

Reducing L_{MPCP} to L_{PCP}

- Take an instance of L_{MPCP} and do the following, using new symbols * and $.
  1. For the first string of each pair, add * after every character.
  2. For the second string of each pair, add * before every character.
  3. Add pair ($, *$).
  4. Make another copy of the first pair, with *'s and an extra * prepended to the first string.

Example: L_{MPCP} to L_{PCP}

MPCP instance, in order:	PCP instance:
(110, 10)	(1*1*0*, *1*0)
(0, 01)	(0*, *0*1)
(100, 001)	(1*0*0*, *0*0*1)
	($, *$) ← *Ender*
	(*1*1*0*, *1*0)

Special pair version of first MPCP choice – only possible start for a PCP solution.

L_{MPCP} to L_{PCP} – (2)

- If the MPCP instance has a solution string w, then padding with stars fore and aft, followed by a $ is a solution string for the PCP instance.
  - Use same sequence of indexes, but the special pair to start.
  - Add ender pair as the last index.

L_{MPCP} to L_{PCP} – (3)

- Conversely, the indexes of a PCP solution give us a MPCP solution.
  1. First index must be special pair – replace by first pair.
  2. Remove ender.

Reducing L_u to L_{MPCP}

- We use MPCP to simulate the sequence of ID's that M executes with input w.
- Suppose $q_0w \vdash I_1 \vdash I_2 \vdash$ … is the sequence of ID's of M with input w.
- Then any solution to the MPCP instance we can construct will begin with this sequence of ID's, separated by #'s.

Reducing L_u to L_{MPCP} – (2)

- But until M reaches an accepting state, the string formed by concatenating the second components of the chosen pairs will always be a full ID ahead of the string from the first pairs.
- If M accepts, we can even out the difference and solve the MPCP instance.

Reducing L_u to L_{MPCP} – (3)

- Key assumption: M has a semi-infinite tape; it never moves left from its initial head position.
- Alphabet of MPCP instance: state and tape symbols of M (assumed disjoint) plus special symbol # (assumed not a state or tape symbol).

Reducing L_u to L_{MPCP} – (4)

- First MPCP pair: (#, #q_0w#).
  - We start out with the second string having the initial ID and a full ID ahead of the first.
- (#, #).
  - We can add ID-enders to both strings.
- (X, X) for all tape symbols X of M.
  - We can copy a tape symbol from one ID to the next.

Example: Copying Symbols

- Suppose we have chosen MPCP pairs to simulate some number of steps of M, and the partial strings from these pairs look like:

. . . #AB

. . . #ABqCD#A B

Reducing L_u to L_{MPCP} – (5)

- For every state q of M and tape symbol X, there are pairs:
  1. (qX, Yp) if δ(q, X) = (p, Y, R).
  2. (ZqX, pZY) if δ(q, X) = (p, Y, L) [any Z].
- Also, if X is the blank, # can substitute.
  1. (q#, Yp#) if δ(q, B) = (p, Y, R).
  2. (Zq#, pZY#) if δ(q, X) = (p, Y, L) [any Z].

Example: Copying Symbols – (2)

- Continuing the previous example, if δ(q, C) = (p, E, R), then:

. . . #ABqCD#

. . . #ABqCD#ABEpD#

- If M moves left, we should not have copied B if we wanted a solution.

Reducing L_u to L_{MPCP} – (6)

- If M reaches final state f, then f "eats" the neighboring tape symbols, one or two at a time, to enable M to reach an "ID" that is essentially empty.
- The MPCP instance has pairs (XfY, f), (fY, f), and (Xf, f) for all tape symbols X and Y.
- To even up the strings and solve: (f##, #).

Example: Cleaning Up After Acceptance

… #ABfCDE#AfD E # f E #f##

… #ABfCDE#AfDE # f E # f##

CFG's from PCP

- We are going to prove that the *ambiguity problem* (is a given CFG ambiguous?) is undecidable.
- As with PCP instances, CFG instances must be coded to have a finite alphabet.
- Let *a* followed by a binary integer i represent the i-th terminal.

CFG's from PCP – (2)

- Let A followed by a binary integer i represent the i-th variable.
- Let A1 be the start symbol.
- Symbols ->, comma, and ϵ represent themselves.
- Example: S -> 0S1 | A, A -> c is represented by A1->a1A1a10,A1->A10,A10->a11

CFG's from PCP – (3)

- Consider a PCP instance with k pairs.
  - i-th pair is (w_i, x_i).
- Assume *index symbols* $a_1, \ldots, a_k$ are not in the alphabet of the PCP instance.
- The *list language* for $w_1, \ldots, w_k$ has a CFG with productions $A \to w_i A a_i$ and $A \to w_i a_i$ for all $i = 1, 2, \ldots, k$.

List Languages

- Similarly, from the second components of each pair, we can construct a list language with productions B -> $x_i B a_i$ and B -> $x_i a_i$ for all i = 1, 2,…, k.
- These languages each consist of the concatenation of strings from the first or second components of pairs, followed by the reverse of their indexes.

Example: List Languages

- Consider PCP instance (a,ab), (baa,aab), (bba,ba).
- Use 1, 2, 3 as the index symbols for these pairs in order.

A -> aA1 | baaA2 | bbaA3 | a1 | baa2 | bba3

B -> abB1 | aabB2 | baB3 | ab1 | aab2 | ba3

Reduction of PCP to the Ambiguity Problem

- Given a PCP instance, construct grammars for the two list languages, with variables A and B.
- Add productions S -> A | B.
- The resulting grammar is ambiguous if and only if there is a solution to the PCP instance.

Example: Reduction to Ambiguity

A -> aA1 | baaA2 | bbaA3 | a1 | baa2 | bba3

B -> abB1 | aabB2 | baB3 | ab1 | aab2 | ba3

S -> A | B

- There is a solution 1, 3.
- Note abba31 has leftmost derivations:

S => A => aA1 => abba31

S => B => abB1 => abba31

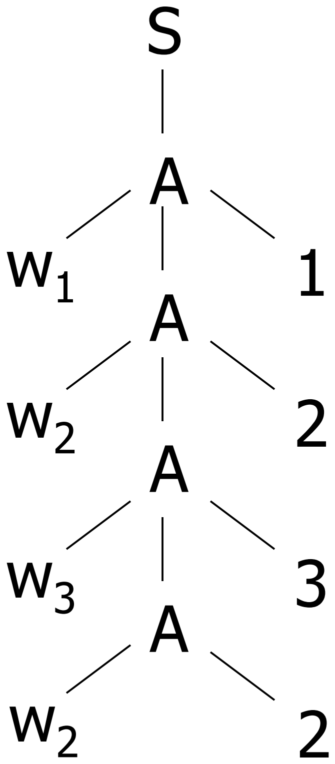
Proof the Reduction Works

- In one direction, if $a_1, \dots, a_k$ is a solution, then $w_1 \dots w_k a_k \dots a_1$ equals $x_1 \dots x_k a_k \dots a_1$ and has two derivations, one starting S -> A, the other starting S -> B.
- Conversely, there can only be two leftmost derivations of the same terminal string if they begin with different first productions. Why? Next slide.

Proof – Continued

- If the two derivations begin with the same first step, say S -> A, then the sequence of index symbols uniquely determines which productions are used.
  - Each except the last would be the one with A in the middle and that index symbol at the end.
  - The last is the same, but no A in the middle.

Example: S =>A=>*...2321



More "Real" Undecidable Problems

- To show things like CFL-equivalence to be undecidable, it helps to know that the complement of a list language is also a CFL.
- We'll construct a deterministic PDA for the complement langauge.

DPDA for the Complement of a List Language

- Start with a bottom-of-stack marker.
- While PCP symbols arrive at the input, push them onto the stack.
- After the first index symbol arrives, start checking the stack for the reverse of the corresponding string.

Complement DPDA – (2)

- The DPDA accepts after every input, with one exception.
- If the input has consisted so far of only PCP symbols and then index symbols, and the bottom-of-stack marker is exposed after reading an index symbol, do not accept.

Using the Complements

- For a given PCP instance, let L_A and L_B be the list languages for the first and second components of pairs.
- Let L_A^c and L_B^c be their complements.
- All these languages are CFL's.

Using the Complements

- Consider $L_A^c \cup L_B^c$.
- Also a CFL.
- = Σ^* if and only if the PCP instance has no solution.
- Why? a solution $a_1, \ldots, a_n$ implies $w_1 \ldots w_n a_n \ldots a_1$ is not in L_A^c, and the equal $x_1 \ldots x_n a_n \ldots a_1$ is not in L_B^c.
- Conversely, anything missing is a solution.

Undecidability of "= Σ*"

- We have reduced PCP to the problem is a given CFL equal to all strings over its terminal alphabet?

Undecidablility of "CFL is Regular"

- Also undecidable: is a CFL a regular language?
- Same reduction from PCP.
- Proof: One direction: If $L_A^c \cup L_B^c = \Sigma^*$, then it surely is regular.

"= Regular" – (2)

- Conversely, we can show that if L $= L_A^c \cup L_B^c$ is not Σ^*, then it can't be regular.
- Proof: Suppose wx is a solution to PCP, where x is the indices.
- Define homomorphism h(0) = w and h(1) = x.

"= Regular" – (3)

- $h(0^n1^n)$ is not in L, because the repetition of any solution is also a solution.
- However, h(y) is in L for any other y in {0,1}*.
- If L were regular, so would be $h^{-1}(L)$, and so would be its complement = $\{0^n1^n \mid n \geq 1\}$.