Reducibility

The primary method for proving that problems are computationally unsolvable

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

If A reduces to B and B is decidable, then A is decidable.

If A reduces to B and A is undecidable, then B is undecidable.

Reducibility Proofs

General Form

•••

Generally used to prove that some problem is undecidable.

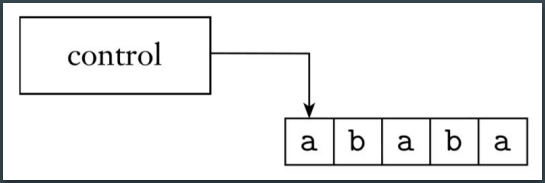
1. Assume that B is decidable.
2. Reduce A to B.
   i.e., Construct some machine that uses B to solve A.
3. If A is undecidable, then this is a contradiction, thus the assumption is wrong and B is undecidable.

Linearly Bounded Automaton

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.

Have limited power, yet are still quite powerful. LBAs are deciders for A_{DFA}, A_{CFG}, E_{DFA}, and E_{CFA}.

Also, every CFL is decidable by a LBA.



Lemma: 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: Recall that a configuration of M is like a snapshot in the middle of its computation. A configuration consists of the state of the control, position of the head, and contents of the tape. Here, M has q states. The length of its tape is n, so the head can be in one of n positions, and g^n possible strings of tape symbols appear on the tape. The product of these three quantities is the total number of different configurations of M with a tape of length n.

Computation Histories

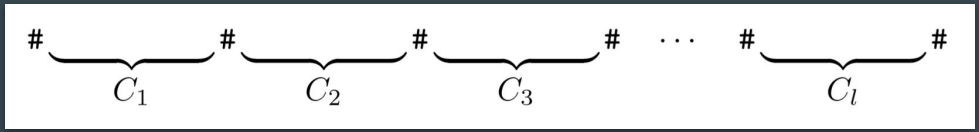
Definition:

Let M be a TM and w an input string. An accepting computation history for M on w is a sequence of configurations, $C_1, C_2, ..., C_l$, where C_1 is the start configurationof 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. (They only exist for halting computations.)

Deterministic machines have at most one history for each input.

What can we say about the bounds of the computation histories for any LBA?



Theorem - A_{LBA} is decidable. Finally!

A_{LBA} ={⟨*M*, *w*⟩ | *M* is an LBA that accepts string *w*}

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* on *w* has not halted within qng^n steps, it must be repeating a configuration according to Lemma 5.8 and therefore looping. That is why our algorithm rejects in this instance.

Fundamental difference between general TMs and LBAs:

- A_{LBA} is decidable.
- A_{TM} is undecidable.

Why?

Theorem - E_{LBA} is undecidable

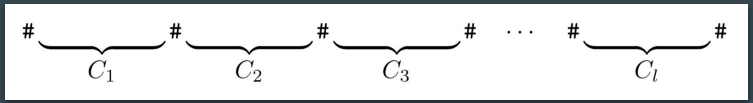
$E_{LBA} = \{\langle M \rangle \mid M \text{ is an LBA where } L(M)=\varnothing\}$

Proof by reduction from A_{TM}, using Computation Histories.

For a given TM M and input w, we will construct a LBA B. We will show that if E_{LBA} is decidable, then A_{TM} is decidable.

B is a LBA whose language comprises all accepting computation histories for M on w.

The construction of B is as follows.



When B receives an input x, B is supposed to accept if x is an accepting computation history for M on w. First, B breaks up x according to the delimiters into strings C_1, C_2, ...,C_l. Then B determines whether the C_i's satisfy the three conditions of an accepting computation history.

1. C_1 is the start configuration for M on w.
2. Each C_{i+1} legally follows from C_i.
3. C_l is an accepting configuration for M.

If all conditions are satisfied, B accepts.

By inverting the decider's answer, we obtain the answer to whether M accepts w.

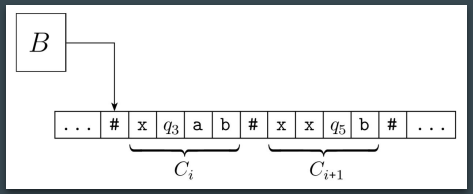
Theorem - E_{LBA} is undecidable (cont.)

Proof:

Suppose TM R decides E_{LBA}. Construct TM S to decide A_{TM} as follows.

S = "On input $\langle M, w \rangle$, where M is a TM and w is a string:

1. Construct LBA B from M and w as described in the proof idea.
2. Run R on input $\langle B \rangle$.
3. If R rejects, accept; if R accepts, reject."



If R accepts $\langle B \rangle$, then $L(B)=\varnothing$. Thus, M has no accepting computation history on w and M doesn't accept w. Consequently, S rejects $\langle M, w \rangle$.

Similarly, if R rejects $\langle B \rangle$, the language of B is nonempty. The only string that B can accept is an accepting computation history for M on w. Thus, M must accept w. Consequently, S accepts $\langle M, w \rangle$.

Because S will either reject or accept, A_{TM} is shown to be decidable for $\langle M, w \rangle$, which is a contradiction.

Therefore, E_{LBA} must be undecidable.

PCP Puzzle

Post Correspondence Problem: Undecidable problem related to string manipulation.

https://www3.nd.edu/~dchiang/teaching/puzzles/post.html
https://en.wikipedia.org/wiki/Post_correspondence_problem

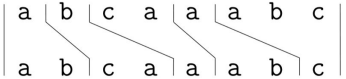
Given a domino in the form: $\left[\frac{a}{ab}\right]$

And a collection of dominoes as:

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

Is it possible to get a match string for the top and bottom letters?

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



Note: Repetitions permitted.

PCP Puzzle

Stated precisely, as a language.

An instance of the PCP is a collection of P dominoes.

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

A match is a sequence $i_1, i_2, \ldots, i_l$, where $t_{i1} t_{i2} \cdots t_{il} = b_{i1} b_{i2} \cdots b_{il}$.

The problem is to determine whether P has a match. Let

PCP = {⟨P⟩ | P is an instance of the Post Correspondence Problem with a match}.

Theorem: PCP is undecidable

Idea: Reduction from A_{TM} via accepting computation histories. Conceptually simple, but with many details.

Since reduction from A_{TM}, we know that we need to simulate some M on w.

Problem: How do we construct P so that a match is an accepting computation history for M on w?

(*i.e.*, iff M accepts w)

Answer: Create dominoes that mimic the valid transitions of one configuration to the next.

For convenience, we create a Modified PCP:

1. Assume M never moves its head off the left side of the tape. We may modify M to achieve this behavior.
2. If $w = \varepsilon$, then use $\sqcup$ as w.
3. The PCP must start with the domino $\left[\frac{t_1}{b_1}\right]$.

MPCP ={$\langle P \rangle$ | P is an instance of the Post Correspondence Problem with a match that starts with the first domino}

Also, remember that

$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$

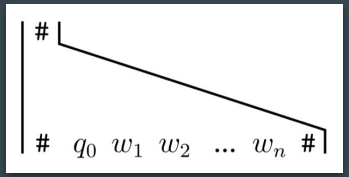
Theorem: PCP is undecidable (cont)

Construct a *P'* for the *MPCP*.
We need 7 types of dominoes.

Number 1:

Put $\left[\frac{\#}{\#q_0 w_1 w_2 \cdots w_n \#}\right]$ into P' as the first domino $\left[\frac{t_1}{b_1}\right]$.

Visualized as:



For all remaining dominoes, what does the top and the bottom each represent?

Number 2:
For every $a, b \in \Gamma$ and every $q, r \in Q$ where $q \neq q_{\text{reject}}$, if $\delta(q, a) = (r, b, R)$, put $\left[\frac{qa}{br}\right]$ into *P'*.

Number 3:
For every $a, b, c \in \Gamma$ and every $q, r \in Q$ where $q \neq q_{\text{reject}}$, if $\delta(q, a)=(r, b, L)$, put $\left[\frac{cqa}{rcb}\right]$ into *P'*.

Number 4:
For every $a \in \Gamma$, put $\left[\frac{a}{a}\right]$ into *P'*.

Number 5:

Put $\left[\frac{\#}{\#}\right]$ and $\left[\frac{\#}{\sqcup\#}\right]$ into *P'*.

Theorem: PCP is undecidable (cont)

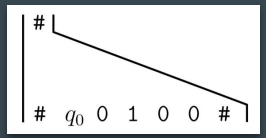
Hypothetical example of what we have so far:

Let:

- $\Gamma = \{0, 1, 2, \sqcup\}$
- $w = 0100$
- $\delta(q_0, 0) = (q_7, 2, \mathrm{R})$

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

Starting state visualized as:



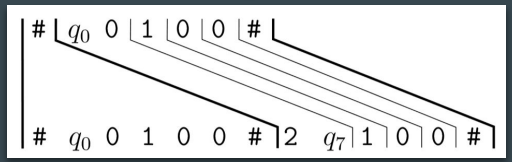
The next domino must have q_0 on the top.

Number 2 provides the domino $\left[\frac{q_0 0}{2 q_7}\right]$ because of the transition $\delta(q_0, 0) = (q_7, 2, \mathrm{R})$.

Number 4 provides the dominoes:

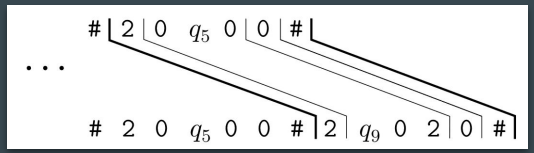
$$\left[\frac{0}{0}\right], \left[\frac{1}{1}\right], \left[\frac{2}{2}\right], \text{ and } \left[\frac{\sqcup}{\sqcup}\right]$$

This creates the following series of dominoes:



Theorem: PCP is undecidable (cont)

Suppose the following appeared:



Because of the presence of the domino $\left[\frac{0q_50}{q_902}\right]$.

Question: What was the transition (δ) that produced the domino?

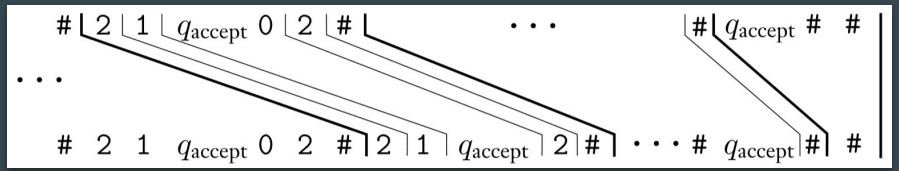
Answer: $\delta(q_5, \texttt{0}) = (q_9, \texttt{0}, \text{L})$

Number 6:
For every $a \in \Gamma$, put $\left[\frac{a\ q_{\text{accept}}}{q_{\text{accept}}}\right]$

and $\left[\frac{q_{\text{accept}}\ a}{q_{\text{accept}}}\right]$ into P'.

Number 7:
Put $\left[\frac{q_{\text{accept}}\#\#}{\#}\right]$ into P'.

Dominoes from 6 and 7 allow us to complete the computation and match the top and bottom.



Theorem: PCP is undecidable (cont)

That concludes the construction of *P'*.

A few observations/questions:

- What is the main difference between the *PCP* and *MPCP*?
- *P'* is an instance of the *MPCP*. What would be the significance of saying that it is an instance of the *PCP* instead?
- How can we modify the *P'* to avoid this problem, and represent the *PCP* directly?

We need technical trickery and slight of hand!

Let $u = u_1 u_2 \cdots u_n$ be any string of length n.
Define $\star u$, $u\star$, and $\star u\star$ to be the three strings:

$$\star u = *u_1 * u_2 * u_3 * \cdots * u_n$$
$$u\star = u_1 * u_2 * u_3 * \cdots * u_n *$$
$$\star u\star = *u_1 * u_2 * u_3 * \cdots * u_n *.$$

To convert *P'* to *P*, do the following. If *P'* is:

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

P is :

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

Theorem: PCP is undecidable.

What is the proof?

Does this match your intuition of the problem?

Assume TM R decides the PCP.

Construct TM S to decide A_{TM}:

"S = On input $\langle M, w \rangle$, an encoding of a TM M and a string w:

1. Construct P (using previous description) that has a match iff M accepts w.
2. Run TM R on input $\langle P \rangle$.
3. If R accepts, accept. If R rejects, reject.

Because A_{TM} is undecidable, contradiction is shown. Therefore, PCP is undecidable.

———

Formalism for Reducibility

Clarifies the previously-seen Reducibility approaches.
(Is a.k.a. many-one reducibility)

Mapping Reducibility is the use of a computable function to convert instances of problem A to instances of problem B.

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

Example: arithmetic operator +

Input: $\langle m, n \rangle$

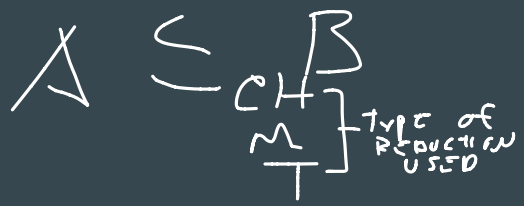
Output: sum of m and n

Example: TM that never moves left off tape.

Input: $\langle M \rangle$

Output: $\langle M' \rangle$ where $L(M) = L(M')$

Mapping Reducibility Formalism



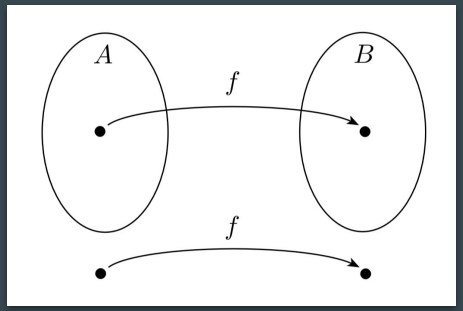
Language A is mapping reducible to language B, written $A \leq_m B$, if there is a computable function

$$f : \Sigma^* \to \Sigma^*,$$

where for every w,

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

The function f is called the reduction from A to B.



Sad Reduction Face

A mapping reduction of A to B provides a way to convert questions about membership testing in A to membership testing in B.

To test whether $w \in A$, we use the reduction f to map w to $f(w)$ and test whether $f(w) \in B$.

Question: Why is this called a reduction?

Note: Mapping Reducibility may seem like a repeat of previous lectures (and, granted, it is very similar), but there are a few important subtleties which we will address throughout the lecture.

Theorem: Decidability and Undecidability

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

PROOF:
We let M be the decider for B and f be the reduction from A to B. We describe a decider N for A as follows.

N = "On input w:

1. Compute $f(w)$.
2. Run M on input $f(w)$ and output whatever M outputs."

Clearly, if $w \in A$, then $f(w) \in B$ because f is a reduction from A to B. Thus, M accepts $f(w)$ whenever $w \in A$. Therefore, N works as desired.

Corollary:
"If $A \leq_m B$ and A is undecidable, then B is undecidable."

Also:
"If $A \leq_m B$ and B is recognizable, then A is recognizable."

Also:
"If $A \leq_m B$ and A is unrecognizable, then B is unrecognizable."

BUT!

What if B is undecidable? What does that prove about A?
What if A is decidable?

Theorem - $\mathrm{HALT}_{\mathrm{TM}}$ is undecidable

Original Method:

$\mathrm{HALT}_{\mathrm{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$

Let's assume that TM R decides $\mathrm{HALT}_{\mathrm{TM}}$. Construct TM S to decide A_{TM} as follows.

S = "On input $\langle M, w\rangle$, an encoding of a TM M and a string w:

1. Run TM R on input $\langle M, w\rangle$.
2. If R rejects, reject.
3. If R accepts, simulate M on w until it halts.
4. If M has accepted, accept; if M has rejected, reject."

Clearly, if R decides $\mathrm{HALT}_{\mathrm{TM}}$, then S decides A_{TM}. This is the contradiction. Because A_{TM} is undecidable, $\mathrm{HALT}_{\mathrm{TM}}$ also *must* be undecidable.

Mapping Reduction:

$\langle M, w\rangle \in A_{\mathrm{TM}}$ if and only if $\langle M', w'\rangle \in \mathrm{HALT}_{\mathrm{TM}}$.
The following machine F computes a reduction f.

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

1. Construct the following machine M'.
   M' = "On input x:
   a. Run M on x.
   b. If M accepts, accept.
   c. If M rejects, enter a loop."
2. Output $\langle M', w\rangle$."

More Theorems Re-examined: PCP

PCP has two reductions:

$ATM \leq_m MPCP$

$MPCP \leq_m PCP$

Is Mapping Reduction transitive?

PROOF:

Suppose $A \leq_m B$ and $B \leq_m C$. Then there are computable functions f and g such that $x \in A \Leftrightarrow f(x) \in B$ and $y \in B \Leftrightarrow g(y) \in C$.

Consider the composition function $h(x) = g(f(x))$.

We can build a TM that computes h as follows:

> First, simulate a TM for f (such a TM exists because we assumed that f is computable) on input x and call the output y.
>
> Then simulate a TM for g on y. The output is $h(x) = g(f(x))$.

Therefore, h is a computable function. Moreover, $x \in A \Leftrightarrow h(x) \in C$.

Hence $A \leq_m C$ via the reduction function h.