Lecture 12 – Universal and diagonal lanugage, Undecidable problems, Post's correspondence problem

NTIN071 Automata and Grammars

Jakub Bulín (KTIML MFF UK)

Spring 2025

** Adapted from the Czech-lecture slides by Marta Vomlelová with gratitude. The translation, some modifications, and all errors are mine.*

Recap of Lecture 11

- Recursively enumerable languages are exactly those generated by (Type 0) grammars
  - TM to G: simulate moves on a reversed non-terminal copy of ω, generate sufficient space, cleanup if accepting state
  - G to TM: generate all strings, check if any of them represents a valid derivation of ω (sentential forms separated by $\#$)
- Context-sensitive languages:
  - context-sensitive grammars are equivalent to monotone grammars
  - Linear Bounded Automaton (LBA): nondeterministic TM with tape limited to the length of input
  - constructions: monotone grammar to LBA, LBA to monotone grammar
- Intro to computability: an overview
- decision problem $\leftrightsquigarrow$ the language of all 'YES' instances
- machine-readable encoding of TMs

The Diagonal language

Let $\mathrm{decode}(w)$ be the TM M such that $\mathrm{code}(M) = w$. (Recall: if w is not a valid code, then $\mathrm{decode}(w)$ is a fixed one-state TM with no instructions.) Then:

$$L_D = \{w \mid w \notin L(\mathrm{decode}(w))\}$$

Theorem

L_D is not recursively enumerable.

Proof idea: there cannot exist a TM recognizing L_D: running it on its own code would lead to Barber's paradox

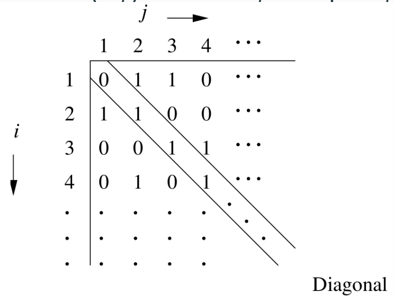
> *"The program accepts all programs that don't accept themselves. Does the program accept itself?"*

Proof that $L_D = \{w \mid w \notin L(\mathrm{decode}(w))\}$ is not RE

Proof: Assume for contradiction that $L_D = L(M)$ for some M. Let $w = \mathrm{code}(M)$. Then $L_D = \{w \mid w \notin L(M)\}$. Is $w \in L(M)$?

$$w \in L(M) \;\Leftrightarrow\; w \in L_D \;\Leftrightarrow\; w \notin L(M) \qquad \square$$

Why 'diagonal'? A variant of Cantor's diagonal argument. Order all TMs by $M_i = \mathrm{decode}(w_i)$. Does M_i accept w_i?



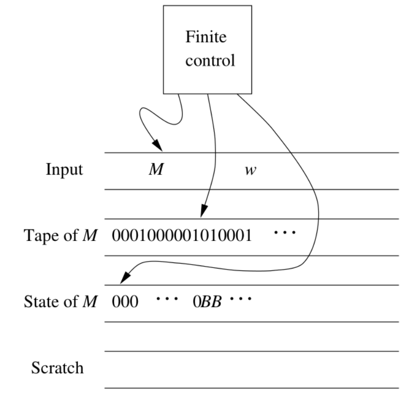
A TM for L_D would be one of the rows but differs from each row in the diagonal element. (Same as the proof that $\mathbb{R}$ is uncountable.)

The Universal Turing Machine

The Universal Turing Machine U can simulate any TM (given by its code) on any input. More precisely, U accepts exactly inputs of the form $\langle \text{code}(M), w \rangle$ where $w \in L(M)$.

The construction: four tapes

1. input tape (w and the encoded transitions of M)
2. simulated tape of M, symbols encoded as 0^i, separated by 1s
3. state of M, again represented by 0^i
4. scratch tape



The operation of U

Initialize:

- Check if the input code is valid, if not, halt without accepting
- Initialize Tape 2 with w in its encoded form: 10 for 0 in w, 100 for 1
  Blanks are left blank and replaced with 1000 only 'on demand'
  Move 2nd head to the first simulated cell.
- Place 0 (the start state of M) on Tape 3.

Simulate moves of M:

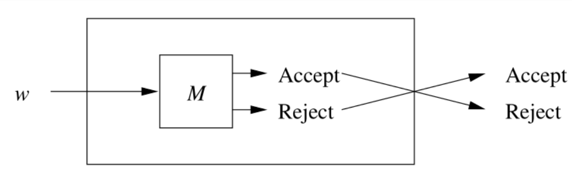
- Search Tape 1 for the appropriate transition $0^i 1 0^j 1 0^k 1 0^\ell 1 0^m$, where 0^i on Tape 3, 0^j on Tape 2.
- Change the content of Tape 3 to 0^k.
- Replace 0^j on Tape 2 by 0^ℓ. Scratch tape to manage spacing.
- Move head on Tape 2 to the next 1 left or right, depending on m.

Termination: If M has no transition matching simulated state & tape symbol, halt without accepting. If M enters accepting state, U accepts.

Recursive languages are closed under complement

Lemma

If L is recursive, then $\overline{L}$ is recursive as well.



Proof: Given M deciding L, construct M' deciding L'. Since M always halts, if it does not accept, the reason is missing transition.

M' has a single, new accepting state q_{ACCEPT}. For every non-accepting state of M and every tape symbol X such that $\delta(q, X)$ is undefined, redefine $\delta'(q, X) = (q_{\text{ACCEPT}}, X, L)$.

Clearly, $L(M') = \overline{L}$. Since M is guaranteed to halt, so is M'. $\square$

Post's theorem

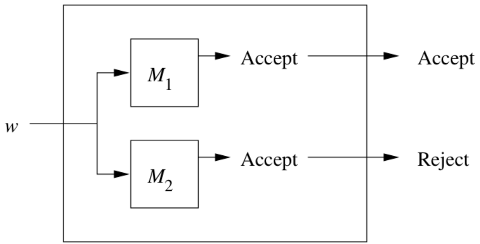
Theorem

L is recursive iff both L and $\overline{L}$ are recursively enumerable.

Proof: $\Rightarrow$ Follows from the lemma.

$\Leftarrow$ Let $L = L(M_1)$ and $\overline{L} = L(M_2)$. For an input w, simulate both M_1 and M_2 (two tapes, states with two components). L and $\overline{L}$ are complementary, one of M_1 or M_2 will halt and accept.

- If M_1 accepts, accept.
- If M_2 accepts, reject. $\square$



The Universal language and its undecidability

$L_U = \{\langle \mathrm{code}(M), w \rangle \mid w \in L(M)\}$ RE but not R

$\overline{L_D} = \{w \mid w \in L(\mathrm{decode}(w))\}$ RE but not R

$L_D = \{w \mid w \notin L(\mathrm{decode}(w))\}$ not RE

Theorem

The Universal language is recursively enumerable but not recursive. Same is true for complement of the Diagonal language.

Proof:

- L_U is RE: it is recognized by the Universal TM U
- $\overline{L_D}$ is RE: rewrite input w to $\langle w, w \rangle = w111w$, then run on U
- $\overline{L_D}$ is not recursive: if it were, by Post's theorem L_D would be RE (actually R), but we know it is not
- L_U is not recursive: if it were, $\overline{L_D}$ would be recursive (rewrite w to $\langle w, w \rangle$, run on the hypothetical M deciding L_U) $\square$

Reductions between decision problems

Definition

A reduction R is an algorithm mapping all instances of P_1 to instances of P_2 that always halts, and for every instance w of P_1 outputs an instance $R(w)$ of P_2 such that:

- w is a YES instance of P_1 iff $R(W)$ is a YES instance of P_2
- w is a NO instance of P_1 iff $R(W)$ is a NO instance of P_2

(Technically, $R = f_M$ for some TM M that always halts.)

Example The mapping $w \rightsquigarrow \langle w, w \rangle = w111w$ (from the previous proof) can clearly be done algorithmically. It is a reduction from $\overline{L_D}$ to L_U (and also from L_D to $\overline{L_U}$).

Only easy reduce to easy, hard only reduce to hard

Theorem

If there is a reduction from P_1 to P_2, then:

(i) *If P_1 is not decidable then neither is P_2.*

(ii) *If P_2 is decidable, then so is P_1.*

(iii) *If P_1 is not partially decidable then neither is P_2.*

(iv) *If P_2 is partially decidable, then so is P_1.*

(i&ii) Let P_1 be undecidable. If P_2 were decidable, we could combine the reduction from P_1 to P_2 with the algorithm deciding P_2 to construct an algorithm that decides P_1.

(iii&iv) Assume P_1 is not partially decidable, but P_2 is. Similarly as above, we could combine the reduction and the algorithm for P_2 to get an algorithm partially deciding P_1–a contradiction.□

“Does the given program halt for the given input?’

An instance of the Halting Problem Halt: $\langle \mathrm{code}(M), w \rangle \in \{0,1\}^*$.
The answer is YES iff M halts on input w; otherwise it is NO.

Theorem

The Halting Problem is undecidable.

(Note that it is partially decidable: we can simulate using U.)

Proof: Reduce the undecidable problem $\overline{L_D}$ to Halt. Given an instance w of $\overline{L_D}$, let $M = \mathrm{decode}(w)$. Modify M to get M' such that if M halts without accepting, M' goes to an infinite loop.

Set $R(w) = \langle \mathrm{code}(M'), w \rangle$. Clearly, it can be done algorithmically.

- If $w \in \overline{L_D}$, i.e. $w \in L(M)$, then M' accepts (thus halts) on w.
- If $w \notin \overline{L_D}$, i.e. $w \notin L(M)$, then either M doesn't halt or halts without accepting. In either case M' doesn't halt on w. $\square$

Accepts no inputs?

- $L_e = \{\text{code(M)} \mid L(M) = \emptyset\}$
- $L_{ne} = \{\text{code(M)} \mid L(M) \neq \emptyset\} = \overline{L_e}$

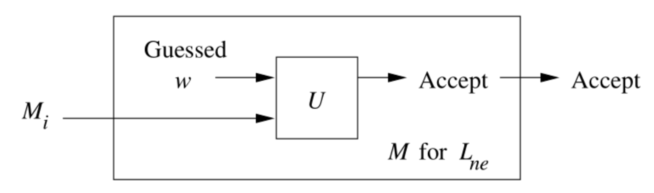
Theorem

(i) L_e *is not recursively enumerable.*

(ii) L_{ne} *is recursively enumerable but not recursive.*

Proof: As $L_{ne} = \overline{L_e}$, (i) follows from (ii) by Post's theorem.

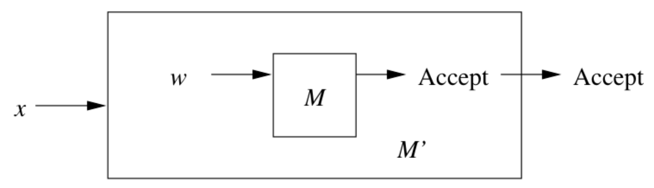
L_{ne} is RE: nondeterministically guess $w \in L(M)$, verify using U



Proof cont'd

L_{ne} is not recursive: reduction from undecidable $\overline{L_D}$

Given $w = \mathrm{code(M)}$, $R(w)$ is a TM M' that ignores its input, rewrites the input tape with w, and simulates M on w.



- If $w \in \overline{L_D}$, i.e. $w \in L(M)$, then $R(w)$ always accepts.
- If $w \notin \overline{L_D}$, i.e. $w \notin L(M)$, then $R(w)$ never accepts. $\square$

Rice's theorem

Which properties of programs are decidable?

None of them!

(Except for trivial properties true/false for all programs.)

We have all the tools, but not the time to prove Rice's theorem.

"Theoretically, static analysis of programs cannot be done automatically?"

Undecidable problems about context-free languages

Post's correspondence problem

Several problems about context-free grammars are undecidable, e.g. is $L(G_1) \cap L(G_2) = \emptyset$? We show that by reduction from a suitable undecidable problem:

Post's correspondence problem (PCP)

- **Instance:** two same-length lists of words over a finite alphabet Σ: $A = w_1, w_2, \ldots, w_k$, $B = x_1, x_2, \ldots, x_k$
- **Question:** is there a finite sequence of positive integers $i_1, \ldots, i_m$ $(m \geq 1)$ such that $w_{i_1} w_{i_2} \ldots w_{i_m} = x_{i_1} x_{i_2} \ldots x_{i_m}$?
- call $i_1, \ldots, i_m$ a solution, (w_i, x_i) a corresponding pair

Theorem

Post's correspondence problem is undecidable.

Proof idea: Reduction from L_U by simulating computation of any TM on any input as a PCP instance. [detailes later, if there's time]

Examples of PCP instances

Solvable instance:

- solution: $2, 1, 1, 3$ (forms 101111110)
- another solution: $2, 1, 1, 3, 2, 1, 1, 3$

partial solution: $i_1, \ldots, i_r$ such that one of $w_{i_1} \ldots w_{i_r}$, $x_{i_1} \ldots x_{i_r}$ is a prefix of the other

observe: prefixes of solns are partial solns

	List A	List B
i	w_i	x_i
1	1	111
2	10111	10
3	10	0

Unsolvable instance:

- $i_1 = 1$, $A : 10$, $B : 101$ (1st letter is 1)
- if $i_2 = 1$, $A : 1010$, $B : 101101$
- if $i_2 = 2$, $A : 10011$, $B : 10111$
- so $i_2 = 3$, $A : 10101$, $B : 101011$
- same situation as after $i_1 = 1$, no way to get same length

	List A	List B
i	w_i	x_i
1	10	101
2	011	11
3	101	011

Context-free languages and (un)decidability

Let G, G' be context-free grammars, R a regular expression, Σ a finite alphabet, w a word (all given on the input).

Recall that the following are decidable:

- $L(G) \ni w$? [the CYK algorithm]
- $L(G) = \emptyset$? [check if S is a generating symbol]

Theorem

The following are undecidable:

(a) $L(G) = \Sigma^*$?

(b) $L(G) = L(R)$?

(c) $L(G) \supseteq L(R)$?

(d) $L(G) = L(G')$?

(e) $L(G) \supseteq L(G')$?

(f) $L(G) \cap L(G') \neq \emptyset$?

(g) *Is G ambiguous?*

We prove undecidability by reduction from the PCP problem.

List languages

Given a PCP instance ($A = w_1, \ldots, w_k, B = x_1, \ldots, x_k$), fix a set of new terminal symbols $S = \{s_1, \ldots, s_k\}$ to represent indices.

The list language L_A for A consists of all words of the form $w_{i_1} \ldots w_{i_m} s_{i_m} \ldots s_{i_1}$ where $m \geq 1$ and $i_j \in \{1, \ldots, k\}$.

- L_A is generated by $G_A = (\{A\}, \Sigma \cup S, \mathcal{P}_A, A)$ with rules

$$\mathcal{P}_A = \{A \to w_1 A a_1 \mid \cdots \mid w_k A a_k \mid w_1 a_1 \mid \cdots \mid w_k a_k\}$$

- L_A is recognized by a deterministic PDA (first store letters from Σ on the stack, then for each s_j pop w_j^R from the stack)
- same is true for analogously defined L_B
- the PCP instance (A, B) has a solution iff $L_A \cap L_B \neq \emptyset$,

$$\begin{aligned} z &= w_{i_1} \ldots w_{i_m} s_{i_m} \ldots s_{i_1} \\ &= x_{i_1} \ldots x_{i_m} s_{i_m} \ldots s_{i_1} \in L(A) \cap L(B) \end{aligned}$$

Proof of (g): Undecidability of ambiguiuty of G

Recall: ambiguous iff $\exists z \in L(G)$ with two different parse trees

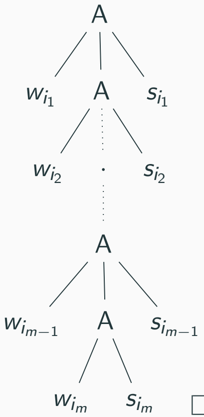
Reduction from PCP: given $(A = w_1, \ldots, w_k, B = x_1, \ldots, x_k)$ construct $G = (\{S, A, B\}, \Sigma \cup \{S\}, \mathcal{P}, S)$ with rules

$$\mathcal{P} = \{S \to A \mid B\} \cup \mathcal{P}_A \cup \mathcal{P}_B$$

G is ambiguous iff the PCP instance (A, B) has a solution: Every $z \in L_A$ has a unique parse tree from A (given by the s_{i_j}'s), same for B.

Different parse trees for $z \in L(G)$ mean:

- $z = w_{i_1} \ldots w_{i_m} s_{i_m} \ldots s_{i_1} \in L_A$, and
- $z = x_{i_1} \ldots x_{i_m} s_{i_m} \ldots s_{i_1} \in L_B$, thus
- $z \in L_A \cap L_B$



Which happens iff (A, B) has as solution. $\square$

Proofs of (f), (a)

Proof of (f): Undecidability of $L(G) \cap L(G') \neq \emptyset$

Reduction from PCP: the given PCP instance (A, B) has a solution iff $L(G_A) \cap L(G_B) \neq \emptyset$ $\square$

Proof of (a): Undecidability of $L(G) = \Sigma^*$

Recall that L_A, L_B are deterministic. Since DCFLs are closed under complement, $L = \overline{L_A} \cup \overline{L_B}$ is context-free. Let $L = L(G)$. Then:

$$(A, B) \text{ has a solution} \;\Leftrightarrow\; L_A \cap L_B \neq \emptyset \;\Leftrightarrow\; L(G) \neq (\Sigma \cup S)^* \qquad \square$$

Note: We proved undecidability of "$L(G) \neq \Sigma^*$?". Undecidability of "$L(G) = \Sigma^*$?" follows easily: they are almost complementary, except for 'invalid inputs':

- Check if input is a valid encoding of G and Σ, if not, reject.
- If it is valid, run algorithm for "$\neq$" and reverse its answer.

The rest of the proof is easy

Proof of (b), (c): Undecidability of $L(G) = L(R)$, $L(G) \supseteq L(R)$

Reduction from $L(G) = \Sigma^*$, create a regular expression R such that $L(R) = \Sigma^*$

Proof of (d), (e): Undecidability of $L(G) = L(G')$, $L(G) \supseteq L(G')$

Reduction from $L(G) = \Sigma^*$, create a grammar G' that generates all words in Σ^* $\square$

Note: "$L(G) \subseteq L(R)$?" is decidable:

$$L(G) \subseteq L(R) \;\Leftrightarrow\; L(G) \cap \overline{L(R)} = \emptyset$$

and $L(G) \cap \overline{L(R)}$ is context-free (intersection with regular).

Undecidability of Post's correspondence problem

Modified Post's correspondence problem (MPCP)

A solution $(i_1, i_2, \ldots, i_m)$ to a PCP instance $(A = w_1, \ldots, w_k, B = x_1, \ldots, x_k)$ is initial if $i_1 = 1$, i.e.,

$$\mathbf{w_1} w_{i_1} w_{i_2} \ldots w_{i_m} = \mathbf{x_1} x_{i_1} x_{i_2} \ldots x_{i_m}$$

Modified PCP: Does a given PCP instance have an initial solution?

Example: This PCP instance has no initial solution:

	A	B
i	w_i	x_i
1	1	111
2	10111	10
3	10	0

Why? A:1, B:111 $\leadsto$ A:11, B:111111 $\leadsto$ A:111, B:111111111

The lengths wont't match. Other choices lead to mismatch.

Reducing MPCP to PCP

From (A, B) construct (C, D) such that (A, B) has an initial solution iff (C, D) has a solution.

The construction: Add new symbols $*, \$ \notin \Sigma$. Define y_i from w_i by adding $*$ after each letter, and z_i from x_i adding $*$ before each letter $(1 \leq i \leq k)$. Define $y_0 = *y_1, z_0 = z_1, y_{k+1} = \$, z_{k+1} = *\$$.

Example:

	List A	List B	List C	List D
i	w_i	x_i	y_i	z_i
0			*1*	*1*1*1
1	1	111	1*	*1*1*1
2	10111	10	1*0*1*1*1*	*1*0
3	10	0	1*0*	*0
4			$	*$

Clearly, $(i_1, i_2, \ldots, i_m)$ is an initial solution to (A, B) iff $(0, i_1, i_2, \ldots, i_m, k+1)$ is a solution to (C, D). □

Undecidability of MPCP

Reduction from L_U: Given a TM M and input w, construct (A, B). Assume M never writes B and never moves left of inital position.

List A	List B	
$\#$	$\#q_0w\#$	
X	X	for all tape symbols $X \in \Gamma$
$\#$	$\#$	
qX	Yp	for $\delta(q, X) = (p, Y, R)$
ZqX	pZY	for $\delta(q, X) = (p, Y, L)$, $Z \in \Gamma$ tape symbol
$q\#$	$Yp\#$	for $\delta(q, B) = (p, Y, R)$
$Zq\#$	$pZY\#$	for $\delta(q, B) = (p, Y, L)$, $Z \in \Gamma$ tape symbol
XqY	q	$q \in F$ accepting state
Xq	q	$q \in F$
qY	q	$q \in F$
$q\#\#$	$q\#$	$q \in F$

Example: $M, w \rightsquigarrow (A, B)$

$M = (\{q_1, q_2, q_3\}, \{0, 1\}, \{0, 1, B\}, \delta, q_1, B, \{q_3\})$, input word $w = 01$

	0	1	B
$\rightarrow q_1$	$(q_2, 1, R)$	$(q_2, 0, L)$	$(q_2, 1, L)$
q_2	$(q_3, 0, L)$	$(q_1, 0, R)$	$(q_2, 0, R)$
$*q_3$	–	–	–

List A	List B	source
$q_1 0$	$1 q_2$	$\delta(q_1, 0) = (q_2, 1, R)$
$0 q_1 1$	$q_2 00$	$\delta(q_1, 1) = (q_2, 0, L)$
$1 q_1 1$	$q_2 10$	$\delta(q_1, 1) = (q_2, 0, L)$
$0 q_1 \#$	$q_2 01 \#$	$\delta(q_1, B) = (q_2, 1, L)$
$1 q_1 \#$	$q_2 11 \#$	$\delta(q_1, B) = (q_2, 1, L)$
$0 q_2 0$	$q_3 00$	$\delta(q_2, 0) = (q_3, 0, L)$
$1 q_2 0$	$q_3 10$	$\delta(q_2, 0) = (q_3, 0, L)$
$q_2 1$	$0 q_1$	$\delta(q_2, 1) = (q_1, 0, R)$
$q_2 \#$	$0 q_2 \#$	$\delta(q_2, B) = (q_2, 0, R)$

$\vdots$

List A	List B
$\#$	$\# q_1 01 \#$
0	0
1	1
$\#$	$\#$
$0 q_3 0$	q_3
$0 q_3 1$	q_3
$1 q_3 0$	q_3
$1 q_3 1$	q_3
$0 q_3$	q_3
$1 q_3$	q_3
$q_3 0$	q_3
$q_3 1$	q_3
$q_3 \#\#$	$\#$

MPCP simulating the TM

List A	List B	source
$q_1 0$	$1 q_2$	$\delta(q_1, 0) = (q_2, 1, R)$
$0 q_1 1$	$q_2 00$	$\delta(q_1, 1) = (q_2, 0, L)$
$1 q_1 1$	$q_2 10$	$\delta(q_1, 1) = (q_2, 0, L)$
$0 q_1 \#$	$q_2 01\#$	$\delta(q_1, B) = (q_2, 1, L)$
$1 q_1 \#$	$q_2 11\#$	$\delta(q_1, B) = (q_2, 1, L)$
$0 q_2 0$	$q_3 00$	$\delta(q_2, 0) = (q_3, 0, L)$
$1 q_2 0$	$q_3 10$	$\delta(q_2, 0) = (q_3, 0, L)$
$q_2 1$	$0 q_1$	$\delta(q_2, 1) = (q_1, 0, R)$
$q_2 \#$	$0 q_2 \#$	$\delta(q_2, B) = (q_2, 0, R)$

- Accepting path of M:
  $q_1 01 \vdash 1 q_2 1 \vdash 10 q_1 \vdash 1 q_2 01 \vdash q_3 101$

List A	List B
$\#$	$\# q_1 01 \#$
0	0
1	1
$\#$	$\#$
$0 q_3 0$	q_3
$0 q_3 1$	q_3
$1 q_3 0$	q_3
$1 q_3 1$	q_3
$0 q_3$	q_3
$1 q_3$	q_3
$q_3 0$	q_3
$q_3 1$	q_3
$q_3 \#\#$	$\#$

A : $\# q_1 01 \# 1 q_2 1 \# 10 q_1 \# 1 q_2 01 \# q_3 101 \# q_3 01 \# q_3 1 \# q_3 \#\#$

B : $\# q_1 01 \# 1 q_2 1 \# 10 q_1 \# 1 q_2 01 \# q_3 101 \# q_3 01 \# q_3 1 \# q_3 \#\#$

Undecidability of PCP

Theorem

Post's correspondence problem is undecidable.

Proof: We have already shown that MPCP reduces to PCP. To show that L_U reduces to MPCP, we must verify that:

M accepts w $\Leftrightarrow$ the constructed (A, B) has an initial solution

$\Rightarrow$ If $w \in L(M)$, start with the initial pair and simulate the computation of M on input w.

$\Leftarrow$ Given an initial solution to (A, B), there is a corresponding computation of M on w:

- MPCP must start with the initial pair
- while $q \notin F$, rules for cleaning cannot be used
- the partial solution is of the form A:x,B:xy, i.e., B is longer
- thus we must end in an accepting state $\square$

Summary of Lecture 12

- the Diagonal language L_D is not recursively enumerable
- the Universal language L_U, the Universal TM: simulate any M on any w
- recursive languages are closed under complement
- Post's theorem: L recursive iff both $L, \overline{L}$ are RE
- L_U, $\overline{L_D}$ are recursively enumerable but not recursive
- reductions between decision problems
- the Halting problem is undecidable
- (Rice's thm: nontriv. properties of programs are undecidable)
- Undecidable problems about context-free grammars
- Source of undecidability: Post's correspondence problem