COMP3630/6360: Theory of Computation
Semester 1, 2022
The Australian National University

Decidability

This lecture covers Chapter 9 of HMU: Decidability and Undecidability

- Preliminary Ideas
- Example of a non-RE language
- Recursive languages
- Universal Language
- Reductions of Problems
- Rice's Theorem
- Post's Correspondence Problem
- Undecidable Problems about CFGs

Additional Reading: Chapter 9 of HMU.

Preliminary Ideas

Enumeration of (Binary) Strings

- We can construct a bijective map ϕ from the set of binary strings $\{0,1\}^*$ to natural numbers $\mathbb{N}$.
- Enlist all strings ordered by length, and for each length, order using lexicographic ordering.
- The set of finite binary strings is countable/denumerable.

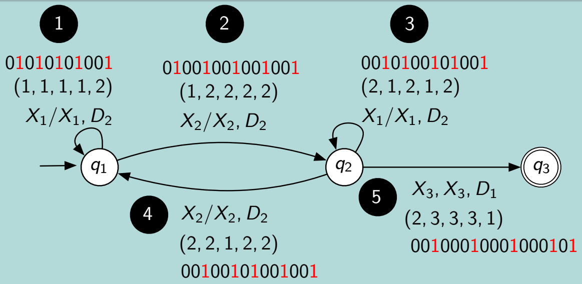
w	$\phi(w)$
ϵ	0
0	1
1	2
00	3
01	4
10	5
11	6
000	7
$\vdots$	$\vdots$
111	16
0000	17
$\vdots$	$\vdots$
1111	32
$\vdots$	$\vdots$

A Code for Turing Machines

- For simplicity, let's assume that input alphabet to be binary.
- WLOG, we can assume that TMs halt at the final state. Consequently, we only need **one** final state (perhaps after collapsing all states into one).
- Consider $M = (Q, \{0,1\}, \Gamma, \delta, q_1, B, F)$.
  - Rename states $\{q_1, \ldots, q_k\}$ for some $k \in N$ with q_1: start state and q_k: final state.
  - Rename input alphabet using $X_1 = 0$, $X_2 = 1$, and blank B as X_3.
  - Rename the rest of the tape symbols by $X_4, \ldots, X_\ell$ for some $\ell \in \mathbb{N}$.
  - Rename L as D_1 and R and D_2.
- Every transition $\delta(q_i, X_j) = (q_k, X_l, D_m)$ can be represented as a tuple (i, j, k, l, m).
- Map each transition tuple (i, j, k, l, m) to a **unique** binary string $0^i 1 0^j 1 0^k 1 0^l 1 0^m$. NB: No string representing a transition tuple contains 11.
- Order transition tuples lexicographically and concatenate all transitions using 11 to indicate end of a transition. Let the resultant string be w_M. For example, 3 transitions can be combined as $\underbrace{0^{i_1} 1 0^{j_1} 1 0^{k_1} 1 0^{l_1} 1 0^{m_1}}_{\text{1st transition}} 11 \underbrace{0^{i_2} 1 0^{j_2} 1 0^{k_2} 1 0^{l_2} 1 0^{m_2}}_{\text{2nd transition}} 11 \underbrace{0^{i_3} 1 0^{j_3} 1 0^{k_3} 1 0^{l_3} 1 0^{m_3}}_{\text{3rd transition}}$
- For each TM M, define the code $\langle M \rangle$ for TM M as w_M.

The Set of Turing Machines

An Example: A TM that accepts strings with odd # of 1s



$$< M >= 010101010^2111010^210^210^210^21110^21010^21010^2111$$
$$0^210^21010^210^21110^210^310^310^3101.$$

Remark 9.1.1

- *Each TM M corresponds to a unique natural number, i.e., $\phi(\langle M \rangle)$; each natural number corresponds to* **at most** *one TM.*
- *There are multiple numbers that represent the 'same' TM.*
- *The set of TMs/RE languages/CFLs/regular languages is countable.*

Example of a non-RE language

Diagonalization Language L_d

- Let M_i be the TM s.t. $\phi(< M_i >) = i$. (If for an i, no such TM exists, we let M_i to be the TM with 1 state, no transitions and no final state, i.e., it accepts no input).
- Construct an infinite table of 0s and 1s with a 1 at the i^{th} row and j^{th} column if M_i accepts $w_j := \phi^{-1}(j)$ (see Slide 3 for ϕ).
- Define a language $L_d = \{w_j : M_j \text{ does not accept } w_j, \text{ where } j \in \mathbb{N}\}$.

	✓			✓		✓		
	ϵ	0	1	00	01	10	11	...
	$\phi^{-1}(0)$	$\phi^{-1}(1)$	$\phi^{-1}(2)$	$\phi^{-1}(3)$	$\phi^{-1}(4)$	$\phi^{-1}(5)$	$\phi^{-1}(6)$	
M_0	**0** ✓	**0**	**0**	**0**	**0**	**0**	**0**	
M_1	**1**	**1**	**0**	**0**	**0**	**1**	**1**	
M_2	**0**	**1**	**1**	**1** Text	**0**	**0**	**1**	
M_3	**1**	**1**	**1**	**0** ✓	**0**	**1**	**1**	
M_4	**1**	**0**	**0**	**1**	**1**	**0**	**0**	
M_5	**1**	**1**	**0**	**0**	**0**	**0** ✓	**1**	
$\vdots$								

$L_d = \{\epsilon, 00, 10, \dots\}$

† Entries are for illustrative purposes only

L_d is not recursively enumerable language

- L_d cannot be accepted by **any** TM.
- For each $i \in \mathbb{N}$, the string w_i is exclusively in either L_d or $L(M_i)$.
- Hence $L_d \neq L(M_i)$ for any $i \in \mathbb{N}$.

	✔ ϵ $\phi^{-1}(0)$	0 $\phi^{-1}(1)$	1 $\phi^{-1}(2)$	✔ 00 $\phi^{-1}(3)$	01 $\phi^{-1}(4)$	✔ 10 $\phi^{-1}(5)$	11 $\phi^{-1}(6)$	...
M_0	0 ✔	0	0	0	0	0	0	
M_1	1	1	0	0	0	1	1	
M_2	0	1	1	1 Text	0	0	1	
M_3	1	1	1	0 ✔	0	1	1	
M_4	1	0	0	1	1	0	0	
M_5	1	1	0	0	0	0 ✔	1	
$\vdots$								

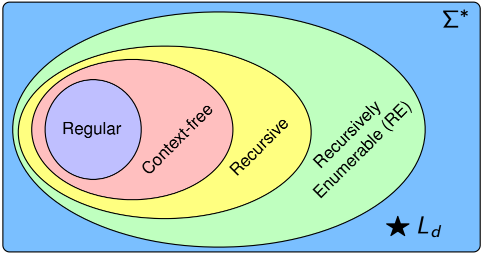
$L_d = \{\epsilon, 00, 10, \ldots\}$

† Entries are for illustrative purposes only

Recursive languages

Recursive Languages

- A language L is **recursive** if it is accepted by a TM M that halts on **all** inputs
  - In such a case, the TM M is said to **decide** L.
  - Every recursive language is recursively enumerable (by definition).



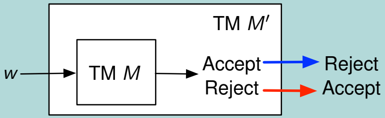
- A (decision) problem that is equivalent to: "is a given w in a given recursive language L?" is said to be **decidable** (for the TM that accepts/rejects L is effectively the machine description of an algorithm for solving the problem).

(Some Obvious) Properties of Recursive Languages

Theorem 9.3.1

If L is **recursive**, *so is L^c.*

Proof of Theorem 9.3.1



- Accepting states of M are non-accepting states of M'.
- Add a new and only final state q_f in M' such that

$$\delta_M(q, X) \text{ undefined and } q \notin F$$
$$\Downarrow$$
$$\delta_{M'}(q, X) = (q_f, X, R).$$

- Recursive languages are closed under complementation.

(Some Obvious) Properties of Recursive Languages

Theorem 9.3.2

If L and L^c are both recursively enumerable, then L (and L^c) are recursive.

Proof of Theorem 9.3.2

- Let $L = L(M)$ and $L^c = L(M')$. Run M and M' in parallel using a 2-tape TM.
- Both TMs cannot halt in final states, and both TMs cannot halt in non-final states.
- Continue running both TMs until either halts in a final state.
- Accept (or reject) if M (or M') halts in a final state, respectively.

Alternate Definition of Recursive Languages

L is recursive if both L and L^c are recursively enumerable.

The Universal Language and Turing Machine

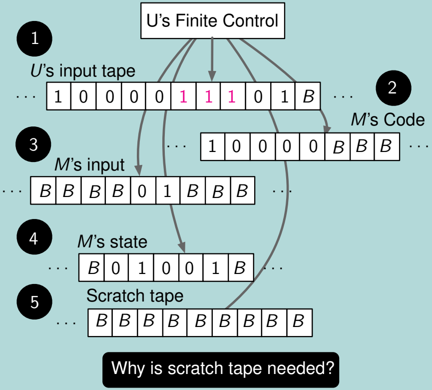
The Universal Language and Turing Machine

Universal Language L_u

- $L_u := \{\langle M \rangle 111 w : \text{ TM } M \text{ and } w \in L(M)\}$. [See Slide 3]

Universal TM U (modelled as 5-tape TM)

1. U copies $\langle M \rangle$ to tape 2 and verifies it for valid structure.
2. Copies w onto tape 3 (maps $0 \mapsto 01$, $1 \mapsto 001$)
3. Initiates 4th tape with 0^1 (M starts in q_1)
4. To simulate a move of M, U reads tapes 3 and 4 to identify M's state and input as 0^i and 0^j; if state is accepting, M (and hence U) accepts its inputs and halts. Else, U scans tape 2 for 110^i10^j1 or $BB0^i10^j1$.
   - If found, using the transition, tapes 4 and 3 are updated, and tape 3's head moves to right or left.
   - If not, M halts, and so does U.



Where does L_u Lie in the Hierarchy of Languages?

Theorem 9.4.1

L_u is recursively enumerable, but is not recursive.

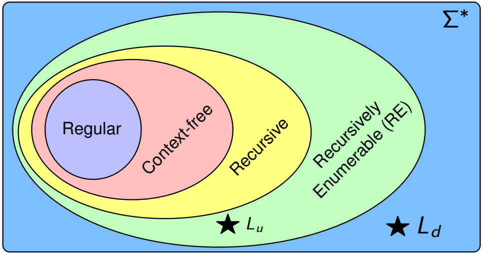
Proof of Theorem 9.4.1

- L_u is recursively enumerable because TM U accepts it.
- Suppose it were recursive. Then, L_u^c is also recursive.
- Let TM M' accepts $w \in L_u^c$ and reject $w \in L_u$.
- Construct a TM M'' such that it first takes its input w appends it with $111w$. It then moves to the beginning of the first w and simulates M'.
- M'' accepts $w \iff w111w \in L_u^c \iff w111w \notin L_u \iff w \in L_d$.
- Then, $L(M'')$ **is** the diagonal language L_d, which is impossible!

Recap

Recap

- There exists a bijection $\phi : \Sigma^* \to \mathbb{N}$.
- There exists an injective (1-1 map) $< \cdot >$: Set of TMs $\to \Sigma^*$.
- RE languages are countable.



- The diagonalization Language L_d is not recursively enumerable.
- Recursive languages are closed under complementation
- The universal language $L_u = \{\langle M \rangle 111w : M \text{ accepts } w\}$ is RE, but not recursive.

Reductions of Problems

What is a Reduction?

- A decision problem P is said to reduce to decision problem Q if **every** instance of P can be transformed to **some** instance of Q and a yes (or no) answer to that instance of Q yields a yes (or no) answer to original instance of P, respectively.
- Here, **transform** implies the existence of a Turing machine that takes an instance of P written on a tape and **always halts** with an instance of Q written on it.
- Note that for deciding **all** instances of P, it is not necessary for all instances of Q to be (re)solved.

Theorem 9.6.1

If a problem P reduces to a problem Q then:

(a) P is undecidable $\Rightarrow$ Q is undecidable

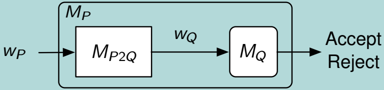
(b) P is non-R.E. $\Rightarrow$ Q is non-R.E.

Problem Reduction

Proof of Theorem 9.6.1

(a) Suppose P is undecidable and Q is decidable. Let TM M_Q decide Q.

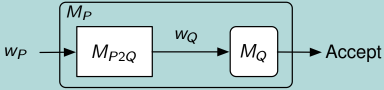
- Consider the TM M_P that first operates as TM M_{P-2-Q} that transforms P to Q, and then operates as M_Q.



- This is a TM that decides all instances of P, a contradiction.

(b) Suppose P is non-R.E. and Q is R.E. Then there must be a TM M_Q that accepts inputs when they correspond to instances of Q whose answer is yes.

- Consider the TM M_P that first operates as TM M_{P-2-Q}, and then operates as M_Q.
- Note that M_P might not halt, since M_Q might not.



- This is a TM that accepts all instances of P whose answer is a yes, a contradiction.

Rice's Theorem

Some More Abstract Languages

Language of TMs Accepting Empty and Non-empty Languages

- $L_e = \{\langle M \rangle : L(M) = \emptyset\}$.
- $L_{ne} = \{\langle M \rangle : L(M) \neq \emptyset\}$. (Note: $L_{ne} \neq L_e^c$).

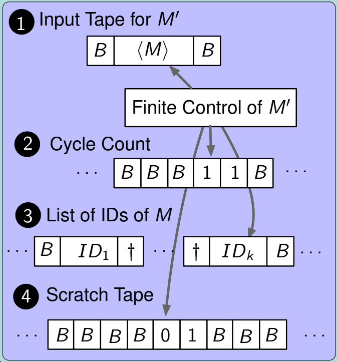
Theorem 9.7.1

L_{ne} is R.E.

L_{ne} is R.E.

Proof of Theorem 9.7.1

- In cycle k, M' runs one move of M for each ID, and adds the initial ID of M when $\phi^{-1}(k)$ is on the tape.
- ID(i,j) = the ID after $j-1$ moves when M reads $\phi^{-1}(j)$ on its tape.
- If any ID contains an accepting state, M' halts as M would have on that input.



Cycle	Tape 1	Tape 2
1	1	$ID(1,1)$
2	10	$ID(1,2)$ † $ID(2,1)$
3	11	$ID(1,3)$ † $ID(2,2)$ † $ID(3,1)$
$\vdots$	$\vdots$	$\vdots$
k	$101\cdots0$	$ID(1,k)$ † $ID(2,k-1)$ † $ID(3,k-2)$ † $\cdots$ † $ID(k,1)$

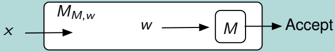
L_{ne} is not recursive

Theorem 9.7.2

L_{ne} is not recursive.

Proof of Theorem 9.7.2

- For every TM M and string w, there is a TM M_w that ignores its input and runs M on w: M_w erases its input tape, and paste w and runs as M.



- **Mind-bending step:** There is a TM M_1 that takes $\langle M\rangle 111w$ and outputs $\langle M_w\rangle$. Note: M_1 **always** halts (even if M does not halt when input is w!)

$$\langle M\rangle 111w \longrightarrow \boxed{M_1} \longrightarrow \langle M_{M,w}\rangle$$

- M accepts $w \iff M_w$ accepts **all** inputs $\iff \langle M_w\rangle \in L_{ne}$
- Suppose L_{ne} is recursive. Then there is a TM M_2 that accepts iff input $\langle M\rangle \in L_{ne}$.
- Let TM M_3 read $\langle M\rangle 111w$ and operate as M_1 and then when M_1 halts, operate as M_2. Then, M_3 accepts/rejects $\langle M\rangle 111w$ iff M accepts/rejects w.
- L_u is then recursive, which is a contradiction.

Rice's Theorem

Given: alphabet Σ and let $RE = \{L \subseteq \Sigma^* \mid L \text{ recursively enumerable}\}$.

- Recursively enumerable (RE) languages L corresponds to TM M if $L = L(M)$
- A **property** of RE languages is subset $P \subseteq RE$ of the set of RE languages over Σ.
- A property P is **trivial** if $P = \emptyset$ or $P = RE$ (and non-trivial otherwise).
- a property $\mathcal{P} \subseteq RE$ is decidable if $L_{\mathcal{P}} = \{\langle M \rangle \mid L(M) \in \mathcal{P}\}$ is decidable.
  - identify TM M with RE language $L(M)$
  - identify M with its code $\langle M \rangle$.

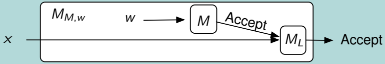
Theorem 9.7.3

Every **non-trivial** *property $\mathcal{P}$ of RE languages is undecidable, i.e., L_P is not recursive.*

Rice's Theorem

Proof of Theorem 9.7.3

- WLOG, we can assume that $\emptyset \notin \mathcal{P}$. Else consider $\mathcal{P}^c$.
- Since $\mathcal{P}$ is non-trivial, there is a language $L \in \mathcal{P}$ and a TM M_L that accepts L
- Let $M_{M,w}$ be a TM that runs M on w and if M accepts w, then reads its input and operates as M_L.



- **Mind-bending step:** There is a TM M_1 that takes $\langle M \rangle 111w$ and outputs $\langle M_{M,w} \rangle$. Note: M_1 **always** halts (even if M does not halt when input is w!)

$$\langle M \rangle 111w \longrightarrow \boxed{M_1} \longrightarrow \langle M_{M,w} \rangle$$

- M accepts $w \iff L(M_{M,w}) = L \in \mathcal{P}$
- If $\mathcal{P}$ were decidable, then there is a ML M_2 such that M_2 accepts $\langle M \rangle$ iff $L(M) \in \mathcal{P}$.
- Then, we can devise a TM M_3 such that it reads $\langle M \rangle 111w$ operates first as M_1 and then when M_1 has halted, it operates as M_2.
- M_3 accepts/rejects $\langle M \rangle 111w \iff L(M_{M,w}) \in / \notin \mathcal{P} \iff M$ accepts/rejects w.
- Then, L_u is recursive, a contradiction

Post's Correspondence Problem

PCP: Definition

- Suppose we are given two ordered lists of strings over Σ, say $A = (u_1, \ldots, u_k)$ and $B = (v_1, \ldots, v_k)$.
- We say (u_i, v_i) to be a **corresponding pair**
- PCP Problem: Is there a sequence of integers $i_1, \ldots, i_m$ such that $u_{i_1} \cdots u_{i_m} = v_{i_1} \cdots v_{i_m}$?
  - m can be greater than k, the list length.
  - We can reuse pairs as many times as we like.

A PCP example

A	110	0011	0110
B	110110	00	110

- A solution cannot start with $i_1 = 3$.
- A solution can start with $i_1 = 1$, but then $i_2 = 1$, and $i_3 = 1 \ldots$. Consequently, i_1 cannot equal 1.
- A solution does exist: $(i_1, i_2, i_3) = (2, 3, 1)$.
- $(i_1, i_2, i_3, i_4, i_5, i_6) = (2, 3, 1, 2, 3, 1)$ is also solution.

Modified PCP (MPCP): Definition

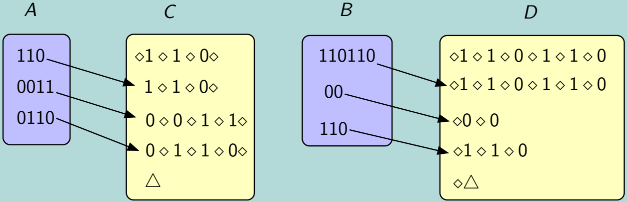
- Suppose we are given two ordered lists of strings over Σ, say $A = (u_1, \ldots, u_k)$ and $B = (v_1, \ldots, v_k)$.
- MPCP Problem: Is there a sequence of integers $i_1, \ldots, i_m$ such that $u_1 u_{i_1} \cdots u_{i_m} = v_1 v_{i_1} \cdots v_{i_m}$
- The previous example does not have a solution when viewed as an MPCP problem.
- So MPCP is indeed a different problem to PCP, but...

Theorem 9.8.1

MPCP reduces to PCP

Outline of Proof of Theorem 9.8.1

- Given lists $A = (u_1, \ldots, u_k)$ and $B = (v_1, \ldots, v_k)$ for MPCP, suppose that symbols $\diamond, \triangle$ are not in the strings.
- Construct lists $C = (w_1, \ldots, w_{k+2})$ and $D = (x_1, \ldots, x_{k+2})$ for PCP as follows.
  - For $i = 1, \ldots, k$, If $u_k = s_1 \ldots s_\ell$, then $w_{k+1} = s_1 \diamond s_2 \diamond \cdots \diamond s_\ell \diamond$. [$\diamond$ succeeds symbols]
  - For $i = 1, \ldots, k$, If $v_k = s_1 \ldots s_\ell$, then $x_{k+1} = \diamond s_1 \diamond s_2 \diamond \cdots \diamond s_\ell$. [$\diamond$ precedes symbols]
  - $w_1 = \diamond w_2$ and $x_1 = x_2$. [Ensures any solution to PCP also starts with $i_1 = 1$]
  - $w_{k+2} = \triangle$ and $x_{k+2} = \diamond\triangle$. [Balances the extra $\diamond$]



$$w_{i_1} \cdots w_{i_m} = x_{i_1} \cdots x_{i_m} \quad \text{(PCP)}$$

$$\Updownarrow$$

$$u_1 u_{i_2 - 1} \cdots u_{i_{m-1} - 1} = v_1 v_{i_2 - 1} \cdots v_{i_{m-1} - 1} \quad \text{(MPCP)}$$

PCP is undecidable

Theorem 9.8.2

PCP is undecidable.

Outline of Proof of Theorem 9.8.2 (for one-sided TM)

- The proof proceeds by constructing a MPCP for each TM M and input w

Rule A: Construct two lists A and B whose first entries are $\diamond$ and $\diamond q_0 w \diamond$

Rule I: Add corresponding pairs (X, X) (all $X \in \Gamma$) and $(\diamond, \diamond)$

Rule B: Suppose q is not a final state. Then, append to the list the following entries

List A	List B	
qX	Yp	*if* $\delta(q, X) = (p, Y, R)$
ZqX	pZY	*if* $\delta(q, X) = (p, Y, L)$
$q\diamond$	$Yp\diamond$	*if* $\delta(q, B) = (p, Y, R)$
$Zq\diamond$	$pZY\diamond$	*if* $\delta(q, B) = (p, Y, L)$

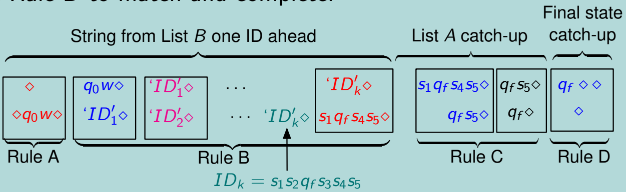
Rule C: For $q \in F$, let (XqY, q), (Xq, q) and (qY, Y) be corresponding pairs for $X, Y \in \Gamma$

Rule D: For $q \in F$ $(q \diamond \diamond, \diamond)$ is a corresponding pair.

PCP is undecidable

Outline of Proof of Theorem 9.8.2

- Suppose there is a solution to the MPCP problem. The solution starts with the first corresponding pair, and the string constructed from List B is already a ID of TM M ahead of the string from List A.
- As we select strings from List A (corresponding to Rule B) to match the last ID, the string from List B adds to its string another valid ID.
- The sequence of IDs constructed are valid sequences of IDs for M starting from q_0w.
- Suppose the last ID constructed in the string constructed from List B corresponds to a final state, then we can gobble up one neighboring symbol at a time using Rule C.
- Once we are done gobbling up all tape symbols, the string from List B is still one final state symbol ahead of List A's string.
- We then use Rule D to match and complete.



PCP is undecidable

Outline of Proof of Theorem 9.8.2

- M accepts $w \iff$ a solution to the MPCP exists.
- If MPCP were decidable, then L_u would be recursive, which it isn't.
- Hence, MPCP is undecidable. [Theorem 9.6.1]
- Since MPCP is undecidable, PCP is also undecidable. [Theorem 9.6.1]

Ambiguity in CFGs

- We'll now revisit CFGs and prove that ambiguity in CFGs is undecidable.

Theorem 9.9.1

The problem if a grammar is ambiguous is undecidable

Outline of Proof of Theorem 9.8.2

- We'll reduce every instance of PCP problem to a CFG.
- Given an PCP problem $A = (w_1, \cdots, w_k)$ and $B = (x_1, \ldots, x_k)$, pick symbols $a_1, \ldots, a_k$ that don't appear in any string in list A or B.
- Now define a grammar G with production rules

$$\begin{aligned} S &\longrightarrow A|B \\ A &\longrightarrow w_1 A a_1 | \cdots | w_k A a_k | w_1 a_1 | \cdots | w_k a_k \\ B &\longrightarrow x_1 B a_1 | \cdots | x_k B a_k | x_1 a_1 | \cdots | x_k a_k \end{aligned}$$

- If there are two leftmost derivations of a string in $L(G)$, one must use $S \longrightarrow A$ and other $S \longrightarrow B$
- Every solution to the PCP leads to 2 leftmost derivations of some string in $L(G)$ and vice versa.
- Since PCP is undecidable, the ambiguity of CFGs must be undecidable [Thm 9.6.1]

Some More Undecidable Problems Concerning CFGs

- Given CFGs G_1 and G_2, is $L(G_1) \cap L(G_2) = \emptyset$?
- Given CFGs G_1 and G_2, is $L(G_1) \subseteq L(G_2)$?
- Given CFGs G_1 and G_2, is $L(G_1) = L(G_2)$?
- Given CFG G and regular language L, is $L(G) = L$?
- Given CFG G and regular language L, is $L \subseteq L(G)$?
- Given CFG G, is $L(G) = \Sigma^*$?