COMP3630 / COMP6363

week 6: **Decidability and Undecidability**

This Lecture Covers Chapter 9 of HMU: Decidability and Undecidability

slides created by: Dirk Pattinson, based on material by
Peter Hoefner and Rob van Glabbeck; with improvements by Pascal Bercher

convenor & lecturer: Pascal Bercher

The Australian National University

Semester 1, 2023

Content of this Chapter

- 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

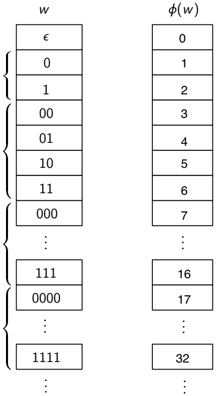
Additional Reading: Chapter 9 of HMU.

Enumeration of (Binary) Strings

- We can construct a bijective map ϕ from the set of binary strings $\{0,1\}^*$ to natural numbers $\mathbb{N}$.
  - Why might that appear surprising?
  - Because each number has a unique binary encoding, but for each we could add an arbitrary number of zeros in the front, so there seem to be more strings over $\{0,1\}$ than numbers in $\mathbb{N}$.

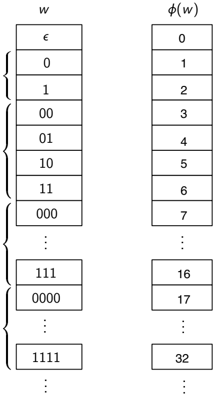
Enumeration of (Binary) Strings

- We can construct a bijective map ϕ from the set of binary strings $\{0,1\}^*$ to natural numbers $\mathbb{N}$.
  - Why might that appear surprising?
  - Because each number has a unique binary encoding, but for each we could add an arbitrary number of zeros in the front, so there seem to be more strings over $\{0,1\}$ than numbers in $\mathbb{N}$.
- Enlist all strings ordered by length, and for each length, order using lexicographic ordering.



Enumeration of (Binary) Strings

- We can construct a bijective map ϕ from the set of binary strings $\{0,1\}^*$ to natural numbers $\mathbb{N}$.
  - Why might that appear surprising?
  - Because each number has a unique binary encoding, but for each we could add an arbitrary number of zeros in the front, so there seem to be more strings over $\{0,1\}$ than numbers in $\mathbb{N}$.
- Enlist all strings ordered by length, and for each length, order using lexicographic ordering.
- The set of finite binary strings is countable/denumerable.



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).

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 $M = (Q, \Sigma = \{0, 1\}, \Gamma = \{0, 1, B, X_4, \ldots, X_\ell\}, \delta, q_1, B, F)$.
  - Rename states $\{q_1, \ldots, q_k\}$ for $k = |Q|$ 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 $\ell = |\Gamma|$.
  - Rename L as D_1 and R and D_2. (The directions.)

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 $M = (Q, \Sigma = \{0, 1\}, \Gamma = \{0, 1, B, X_4, \ldots, X_\ell\}, \delta, q_1, B, F)$.
  - Rename states $\{q_1, \ldots, q_k\}$ for $k = |Q|$ 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 $\ell = |\Gamma|$.
  - Rename L as D_1 and R and D_2. (The directions.)
- Every transition $\delta(q_i, X_j) = (q_k, X_l, D_m)$ can be represented as a tuple (i, j, k, l, m).

A Code for Turing Machines

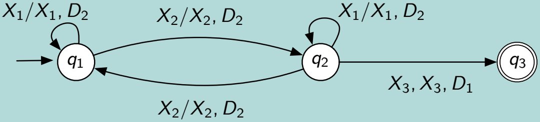
- For simplicity, let's assume that input alphabet to be binary.
- WLOG, we can assume that TMs halt at the final state. Consequently, we only need **one** final state (perhaps after collapsing all states into one).
- Consider $M = (Q, \Sigma = \{0,1\}, \Gamma = \{0,1,B,X_4,\ldots,X_\ell\}, \delta, q_1, B, F)$.
  - Rename states $\{q_1,\ldots,q_k\}$ for $k = |Q|$ 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 $\ell = |\Gamma|$.
  - Rename L as D_1 and R and D_2. (The directions.)
- 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 $0^i10^j10^k10^l10^m$.
  NB: No string representing a transition tuple contains 11.

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 $M = (Q, \Sigma = \{0,1\}, \Gamma = \{0,1,B,X_4,\ldots,X_\ell\}, \delta, q_1, B, F)$.
  - Rename states $\{q_1,\ldots,q_k\}$ for $k = |Q|$ 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 $\ell = |\Gamma|$.
  - Rename L as D_1 and R and D_2. (The directions.)
- 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 $0^i10^j10^k10^l10^m$.
  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 $\underbrace{0^{i_1}10^{j_1}10^{k_1}10^{l_1}10^{m_1}}_{\text{1st transition}}\,11\,\underbrace{0^{i_2}10^{j_2}10^{k_2}10^{l_2}10^{m_2}}_{\text{2nd transition}}\,11\,\underbrace{0^{i_3}10^{j_3}10^{k_3}10^{l_3}10^{m_3}}_{\text{3rd transition}}$
- For each TM M, define the code $\langle M\rangle$ for TM M as w_M.

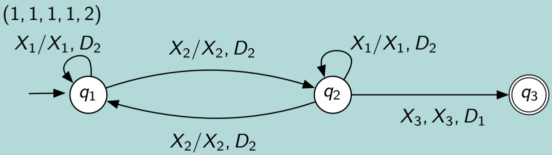
The Set of Turing Machines

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



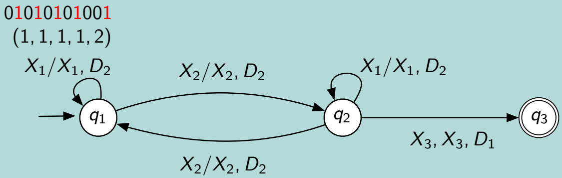
The Set of Turing Machines

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



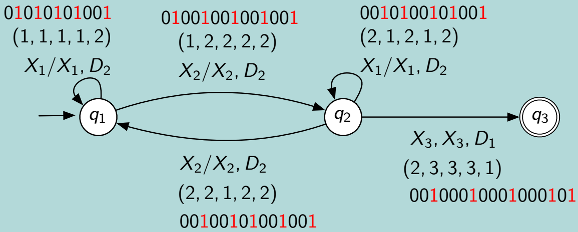
The Set of Turing Machines

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



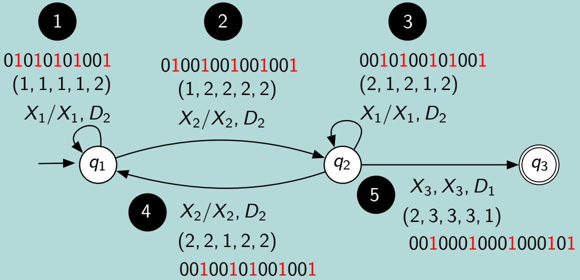
The Set of Turing Machines

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



The Set of Turing Machines

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

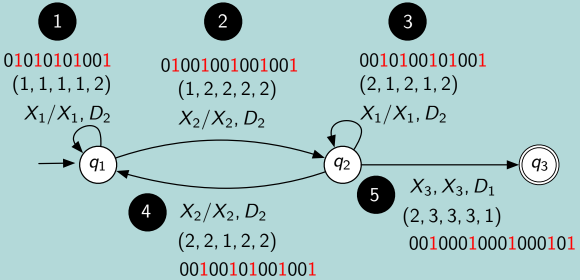


$$< M >= 010101010^2 111010^2 10^2 10^2 10^2 1110^2 1010^2 1010^2 111$$

$$0^2 10^2 1010^2 10^2 1110^2 10^3 10^3 10^3 101.$$

The Set of Turing Machines

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



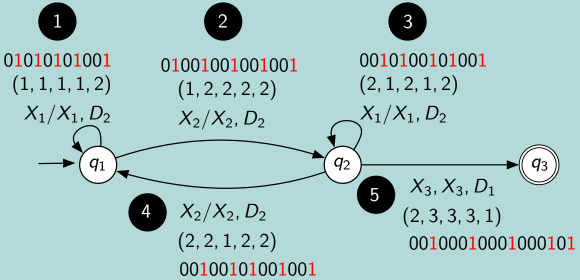
$$< M >= 010101010^2111010^210^210^210^21110^21010^21010^2111$$
$$0^210^21010^210^21110^210^310^310^3101.$$

Remark 9.1.1

- *Each TM M encoding has a unique natural number, i.e., $\phi(\langle M \rangle)$; Each TM M may have several codes $\langle M \rangle$ and thus several numbers; but each natural number corresponds to* **at most** *one TM.*

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 encoding has a unique natural number, i.e., $\phi(\langle M \rangle)$; Each TM M may have several codes $\langle M \rangle$ and thus several numbers; but each natural number corresponds to* **at most** *one TM.*
- *The set of TMs/RE languages/CFLs/regular languages is countable.*

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. Rows: M_0, M_1, ... as above and cols: All Strings according to slide 3. Cell $(i, j) = 1$ iff M_i accepts $w_j := \phi^{-1}(j)$.
- Define a language $L_d = \{w_j : M_j \text{ does not accept } w_j, \text{ where } j \in \mathbb{N}\}$.

	✔ 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	
⋮								

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

† Entries are for illustrative purposes only

L_d is not recursively enumerable language

- L_d cannot be accepted by **any** TM.

	✔			✔		✔		
	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**	
⋮								

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

† Entries are for illustrative purposes only

L_d is not recursively enumerable language

- L_d cannot be accepted by **any** TM.
- Assume it were. Then there is a TM M_j accepting L_d, i.e., $L(M_j) = L_d$.

	✔			✔		✔		
	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

L_d is not recursively enumerable language

- L_d cannot be accepted by **any** TM.
- Assume it were. Then there is a TM M_j accepting L_d, i.e., $L(M_j) = L_d$.
- But now we get a contradiction:
  - If $(j, j) = 1$, then $w_j \in L(M_j)$.
    But if $w_j \in L(M_j)$, then $w_j \notin L_d$, so cell (j, j) should be 0! ↯

	✔ 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

L_d is not recursively enumerable language

- L_d cannot be accepted by **any** TM.
- Assume it were. Then there is a TM M_j accepting L_d, i.e., $L(M_j) = L_d$.
- But now we get a contradiction:
  - If $(j,j) = 1$, then $w_j \in L(M_j)$.
    But if $w_j \in L(M_j)$, then $w_j \notin L_d$, so cell (j,j) should be 0! ↯
  - If $(j,j) = 0$, then $w_j \notin L(M_j)$.
    But if $w_j \notin L(M_j)$, then $w_j \in L_d$, so cell (j,j) should be 1! ↯

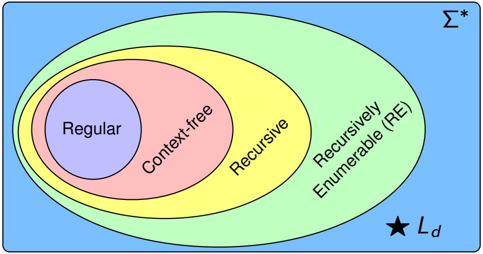
	✔			✔		✔		
	ϵ	0	1	00	01	10	11	...
	$\phi^{-1}(0)$	$\phi^{-1}(1)$	$\phi^{-1}(2)$	$\phi^{-1}(3)$	$\phi^{-1}(4)$	$\phi^{-1}(5)$	$\phi^{-1}(6)$	
M_0	**0** ✔	**0**	**0**	**0**	**0**	**0**	**0**	
M_1	**1**	**1**	**0**	**0**	**0**	**1**	**1**	
M_2	**0**	**1**	**1**	**1** Text	**0**	**0**	**1**	
M_3	**1**	**1**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

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

† Entries are for illustrative purposes only

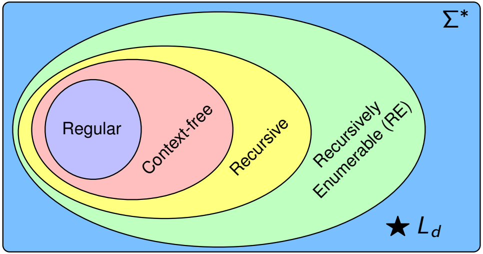
Recursive Languages

- 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).



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).



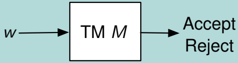
- Do not confuse deciding with accepting! TMs can accept without always terminating (namely, e.g, for languages in $RE \setminus R$, where R denotes the recursive languages).

(Some Obvious) Properties of Recursive Languages

Theorem 9.3.1

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

Proof of Theorem 9.3.1



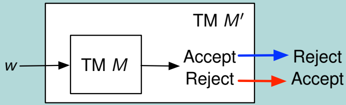
- Recursive languages are closed under complementation.

(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 with $L(M) = L$ are non-accepting states of M' with $L(M') = L^c$.
- 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.

(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

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)

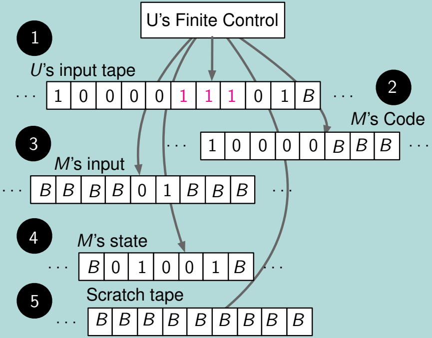
The Universal Language and Turing Machine

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.



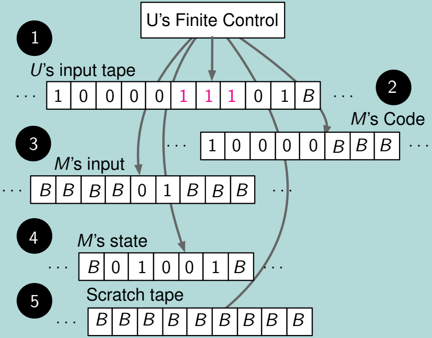
The Universal Language and Turing Machine

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 $0 \mapsto 01$, $1 \mapsto 001$)



The Universal Language and Turing Machine

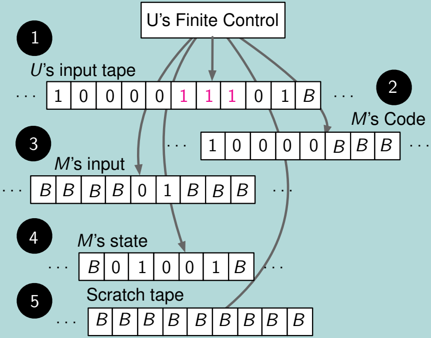
Universal Language L_u

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



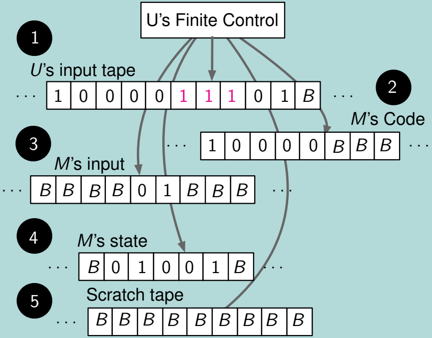
The Universal Language and Turing Machine

Universal Language L_u

- $L_u := \{\langle M \rangle 111 w :$ TM M 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.

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.

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' accept $w \in L_u^c$ and reject $w \in L_u$.

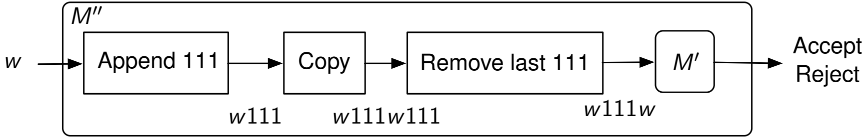
Where does L_u Lie in the Hierarchy of Languages?

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' accept $w \in L_u^c$ and reject $w \in L_u$.
- Construct a TM M'' such that it first takes its input w and appends it with $111w$. It then moves to the beginning of the first w and simulates M'.



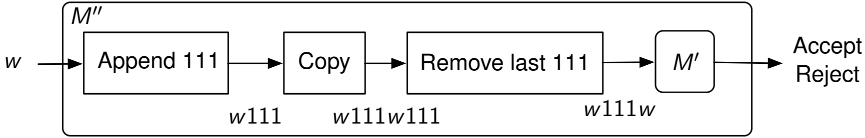
Where does L_u Lie in the Hierarchy of Languages?

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' accept $w \in L_u^c$ and reject $w \in L_u$.
- Construct a TM M'' such that it first takes its input w and appends it with $111w$. It then moves to the beginning of the first w and simulates M'.
- M'' accepts $w \iff w111w \in L_u^c$



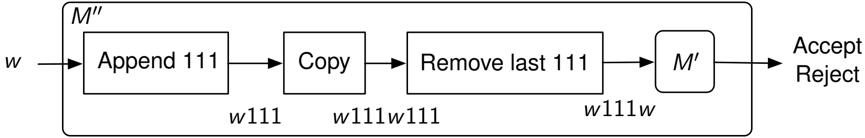
Where does L_u Lie in the Hierarchy of Languages?

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' accept $w \in L_u^c$ and reject $w \in L_u$.
- Construct a TM M'' such that it first takes its input w and appends it with $111w$. It then moves to the beginning of the first w and simulates M'.
- M'' accepts $w \iff w111w \in L_u^c \iff w111w \notin L_u$



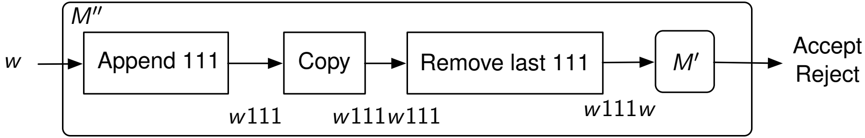
Where does L_u Lie in the Hierarchy of Languages?

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' accept $w \in L_u^c$ and reject $w \in L_u$.
- Construct a TM M'' such that it first takes its input w and appends it with $111w$. It then moves to the beginning of the first w and simulates M'.
- M'' accepts $w \iff w111w \in L_u^c \iff w111w \notin L_u \iff w \in L_d$.



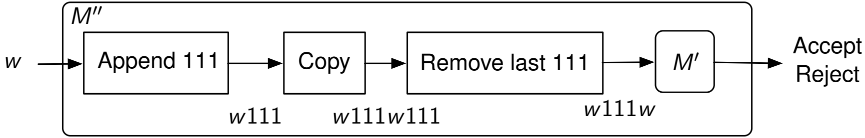
Where does L_u Lie in the Hierarchy of Languages?

Theorem 9.4.1

L_u is recursively enumerable, but is not recursive.

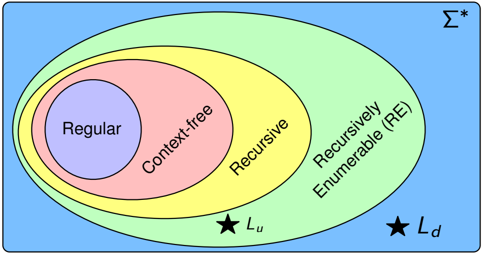
Proof of Theorem 9.4.1

- L_u is recursively enumerable because TM U accepts it.
- Suppose it were recursive. Then, L_u^c is also recursive.
- Let TM M' accept $w \in L_u^c$ and reject $w \in L_u$.
- Construct a TM M'' such that it first takes its input w and 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

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



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

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.
  - We did already make use of reductions in this lecture multiple times!
  - E.g., reduce the problem of deciding L^c to the problem of deciding L: Here the new problem was only a minimal modification, by flipping results (see slide 9).
- 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.

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.
  - We did already make use of reductions in this lecture multiple times!
  - E.g., reduce the problem of deciding L^c to the problem of deciding L: Here the new problem was only a minimal modification, by flipping results (see slide 9).
- 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.
- Alternative formulation: There is a function $f : \Sigma^* \to \Sigma^*$, s.t., $\sigma \in P \leftrightarrow f(\sigma) \in Q$, and f can be computed by a terminating TM.

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.
  - We did already make use of reductions in this lecture multiple times!
  - E.g., reduce the problem of deciding L^c to the problem of deciding L: Here the new problem was only a minimal modification, by flipping results (see slide 9).
- 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.
- Alternative formulation: There is a function $f : \Sigma^* \to \Sigma^*$, s.t., $\sigma \in P \leftrightarrow f(\sigma) \in Q$, and f can be computed by a terminating TM.

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-RE $\Rightarrow$ Q is non-RE.

Problem Reduction

Proof of Theorem 9.6.1

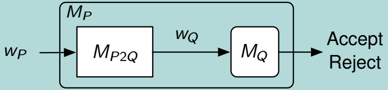
(a) P **is undecidable** $\Rightarrow$ Q **is undecidable**.
Suppose P is undecidable and Q is decidable. Let TM M_Q decide Q.

Problem Reduction

Proof of Theorem 9.6.1

(a) P **is undecidable** $\Rightarrow$ Q **is undecidable**.
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_{P2Q} that transforms P to Q, and then operates as M_Q.

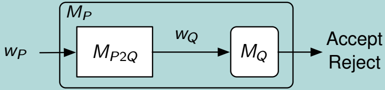


Problem Reduction

Proof of Theorem 9.6.1

(a) P **is undecidable** $\Rightarrow$ Q **is undecidable**.
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_{P2Q} that transforms P to Q, and then operates as M_Q.



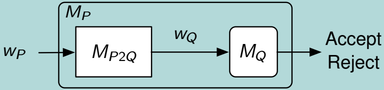
- This is a TM that decides all instances of P, a contradiction.

Problem Reduction

Proof of Theorem 9.6.1

(a) P **is undecidable** $\Rightarrow$ Q **is undecidable**.
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_{P2Q} that transforms P to Q, and then operates as M_Q.



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

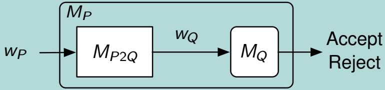
(b) P **is non-RE** $\Rightarrow$ Q **is non-RE**.
Suppose P is non-RE and Q is RE. Then there must be a TM M_Q that accepts inputs when they correspond to instances of Q whose answer is yes.

Problem Reduction

Proof of Theorem 9.6.1

(a) P **is undecidable** $\Rightarrow$ Q **is undecidable**.
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_{P2Q} that transforms P to Q, and then operates as M_Q.



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

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

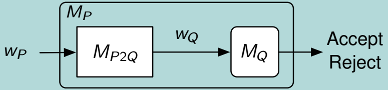
- Consider the TM M_P that first operates as TM M_{P2Q}, and then operates as M_Q.

Problem Reduction

Proof of Theorem 9.6.1

(a) P **is undecidable** $\Rightarrow$ Q **is undecidable**.
Suppose P is undecidable and Q is decidable. Let TM M_Q decide Q.

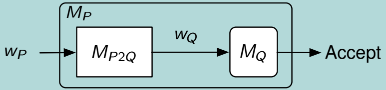
- Consider the TM M_P that first operates as TM M_{P2Q} that transforms P to Q, and then operates as M_Q.



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

(b) P **is non-RE** $\Rightarrow$ Q **is non-RE**.
Suppose P is non-RE and Q is RE. 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_{P2Q}, and then operates as M_Q.
- Note that M_P might not halt, since M_Q might not.

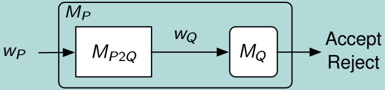


Problem Reduction

Proof of Theorem 9.6.1

(a) P **is undecidable** $\Rightarrow$ Q **is undecidable**.
Suppose P is undecidable and Q is decidable. Let TM M_Q decide Q.

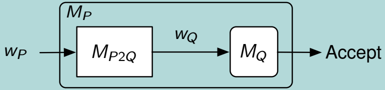
- Consider the TM M_P that first operates as TM M_{P2Q} that transforms P to Q, and then operates as M_Q.



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

(b) P **is non-RE** $\Rightarrow$ Q **is non-RE**.
Suppose P is non-RE and Q is RE. 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_{P2Q}, 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.

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: $L_{ne} \neq L_e^c$, because some strings don't encode TMs.)

Theorem 9.7.1

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: $L_{ne} \neq L_e^c$, because some strings don't encode TMs.)

Theorem 9.7.1

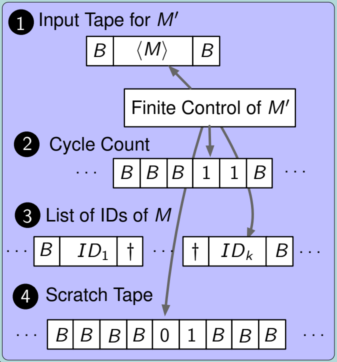
L_{ne} is RE.

Note that this theorem doesn't say whether it's recursive or not!

L_{ne} is RE.

Proof of Theorem 9.7.1 (using "dovetailing")

- 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	$101\cdots0$	$ID(1,k)$ † $ID(2,k-1)$ † $ID(3,k-2)$ † $\cdots$ † $ID(k,1)$

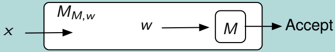
L_{ne} is not recursive

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 $M_{M,w}$ that ignores its input and runs M on w: $M_{M,w}$ erases its input tape, pastes w, and runs it as/on M.



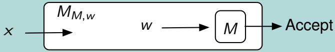
L_{ne} is not recursive

Theorem 9.7.2

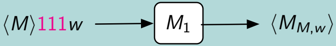
L_{ne} is not recursive.

Proof of Theorem 9.7.2

- For every TM M and string w, there is a TM $M_{M,w}$ that ignores its input and runs M on w: $M_{M,w}$ erases its input tape, pastes w, and runs it as/on M.



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



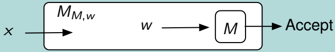
L_{ne} is not recursive

Theorem 9.7.2

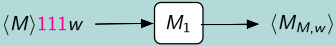
L_{ne} is not recursive.

Proof of Theorem 9.7.2

- For every TM M and string w, there is a TM $M_{M,w}$ that ignores its input and runs M on w: $M_{M,w}$ erases its input tape, pastes w, and runs it as/on M.



- **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!)



- M accepts $w \iff M_{M,w}$ accepts **all** inputs

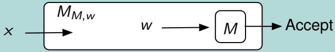
L_{ne} is not recursive

Theorem 9.7.2

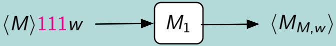
L_{ne} is not recursive.

Proof of Theorem 9.7.2

- For every TM M and string w, there is a TM $M_{M,w}$ that ignores its input and runs M on w: $M_{M,w}$ erases its input tape, pastes w, and runs it as/on M.



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



- M accepts w $\iff$ $M_{M,w}$ accepts **all** inputs $\iff$ $\langle M_{M,w} \rangle \in L_{ne}$

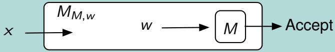
L_{ne} is not recursive

Theorem 9.7.2

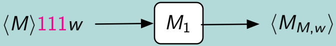
L_{ne} is not recursive.

Proof of Theorem 9.7.2

- For every TM M and string w, there is a TM $M_{M,w}$ that ignores its input and runs M on w: $M_{M,w}$ erases its input tape, pastes w, and runs it as/on M.



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



- M accepts w $\iff$ $M_{M,w}$ accepts **all** inputs $\iff$ $\langle M_{M,w}\rangle \in L_{ne}$
- Suppose L_{ne} is recursive. Then there is a TM M_2 that accepts iff input $\langle M\rangle \in L_{ne}$.

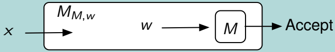
L_{ne} is not recursive

Theorem 9.7.2

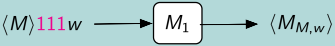
L_{ne} is not recursive.

Proof of Theorem 9.7.2

- For every TM M and string w, there is a TM $M_{M,w}$ that ignores its input and runs M on w: $M_{M,w}$ erases its input tape, pastes w, and runs it as/on M.



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



- M accepts w $\iff$ $M_{M,w}$ accepts **all** inputs $\iff$ $\langle M_{M,w} \rangle \in L_{ne}$
- 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 $\langle M \rangle 111 w$ and operate as M_1 and then when M_1 halts, operate as M_2. Then, M_3 accepts/rejects $\langle M \rangle 111 w$ iff M accepts/rejects w.

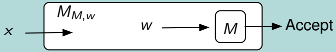
L_{ne} is not recursive

Theorem 9.7.2

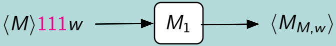
L_{ne} is not recursive.

Proof of Theorem 9.7.2

- For every TM M and string w, there is a TM $M_{M,w}$ that ignores its input and runs M on w: $M_{M,w}$ erases its input tape, pastes w, and runs it as/on M.



- **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!)



- M accepts $w \iff M_{M,w}$ accepts **all** inputs $\iff \langle M_{M,w} \rangle \in L_{ne}$
- 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 $\mathcal{P} \subseteq RE$ of the set of RE languages over Σ. Why do we call sets of languages a property? Think of examples:
  - $\mathcal{P}_1 = \{L \subseteq \Sigma^* : |L| < \infty\}$ (the property is being finite)
  - $\mathcal{P}_2 = \{L \subseteq \Sigma^* : \text{there is a DFA D, s.t. } L = L(D)\}$ (the property is being regular)

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 $\mathcal{P} \subseteq RE$ of the set of RE languages over Σ. Why do we call sets of languages a property? Think of examples:
  - $\mathcal{P}_1 = \{L \subseteq \Sigma^* : |L| < \infty\}$ (the property is being finite)
  - $\mathcal{P}_2 = \{L \subseteq \Sigma^* : \text{there is a DFA D, s.t. } L = L(D)\}$ (the property is being regular)
- A property $\mathcal{P}$ is **trivial** if $\mathcal{P} = \emptyset$ or $\mathcal{P} = RE$ (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.

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 $\mathcal{P} \subseteq RE$ of the set of RE languages over Σ. Why do we call sets of languages a property? Think of examples:
  - $\mathcal{P}_1 = \{L \subseteq \Sigma^* : |L| < \infty\}$ (the property is being finite)
  - $\mathcal{P}_2 = \{L \subseteq \Sigma^* : \text{there is a DFA D, s.t. } L = L(D)\}$ (the property is being regular)
- A property $\mathcal{P}$ is **trivial** if $\mathcal{P} = \emptyset$ or $\mathcal{P} = RE$ (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.

Theorem 9.7.3

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

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 $\mathcal{P} \subseteq RE$ of the set of RE languages over Σ.
  Why do we call sets of languages a property? Think of examples:
  - $\mathcal{P}_1 = \{L \subseteq \Sigma^* : |L| < \infty\}$ (the property is being finite)
  - $\mathcal{P}_2 = \{L \subseteq \Sigma^* : \text{there is a DFA D, s.t. } L = L(D)\}$ (the property is being regular)
- A property $\mathcal{P}$ is **trivial** if $\mathcal{P} = \emptyset$ or $\mathcal{P} = RE$ (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.

Theorem 9.7.3

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

- So Rice's theorem says something about some (many!) subsets
  $S \subseteq \{\langle M \rangle : \text{M is a TM}\}$ (So we want to know something about TMs!)

Rice's Theorem (Example 1)

How about the "property" that a TM has 10 states? (Should be decidable!)

- Let $L_{10} = \{\langle M \rangle : \text{M has 10 states}\}$. But we have to be able to write it as: $L_{10} = \{\langle M \rangle : L(M) \in \mathcal{P}\}$ where $\mathcal{P} \subseteq RE$ and not trivial.

Rice's Theorem (Example 1)

How about the "property" that a TM has 10 states? (Should be decidable!)

- Let $L_{10} = \{\langle M \rangle : \text{M has 10 states}\}$. But we have to be able to write it as: $L_{10} = \{\langle M \rangle : L(M) \in \mathcal{P}\}$ where $\mathcal{P} \subseteq RE$ and not trivial.
- So how about $\mathcal{P}_{10} = \{L \subseteq \Sigma^* : \text{there is a TM M, s.t. } L = L(M) \text{ and } M \text{ has 10 states}\}$?

Rice's Theorem (Example 1)

How about the "property" that a TM has 10 states? (Should be decidable!)

- Let $L_{10} = \{\langle M \rangle : \text{M has 10 states}\}$. But we have to be able to write it as: $L_{10} = \{\langle M \rangle : L(M) \in \mathcal{P}\}$ where $\mathcal{P} \subseteq RE$ and not trivial.
- So how about $\mathcal{P}_{10} = \{L \subseteq \Sigma^* : \text{there is a TM M, s.t. } L = L(M) \text{ and } M \text{ has 10 states}\}$?
- This doesn't work since we can take some M_9 with 9 states (and thus $\langle M_9 \rangle \notin L_{10}$) and add a dummy state, so we have 10 in the resulting TM M_{10}. Now we have:
  - $\langle M_9 \rangle \notin L_{10}$, and $\langle M_{10} \rangle \in L_{10}$, but
  - $L(M_9) = L(M_{10})$, so $L(M_9) \in \mathcal{P}_{10}$ and $L(M_{10}) \in \mathcal{P}_{10}$.
  - Recall $L_{\mathcal{P}} = \{\langle M \rangle \mid L(M) \in \mathcal{P}\}$, so $\langle M_9 \rangle \in L_{\mathcal{P}_{10}}$. ↯

  → So it doesn't work! It's not a property of languages! (So Rice's theorem doesn't apply.)

Rice's Theorem (Example 2)

How about the property that the language contains String "01"?

- Let $\mathcal{P}_{01} = \{L \subseteq \Sigma : 01 \in L\}$, which is non-trivial:

Rice's Theorem (Example 2)

How about the property that the language contains String "01"?

- Let $\mathcal{P}_{01} = \{L \subseteq \Sigma : 01 \in L\}$, which is non-trivial:
  - $\mathcal{P}_{01} \neq \emptyset$ (e.g., $L_1 = \{01\} \in \mathcal{P}_{01}$)

Rice's Theorem (Example 2)

How about the property that the language contains String "01"?

- Let $\mathcal{P}_{01} = \{L \subseteq \Sigma : 01 \in L\}$, which is non-trivial:
  - $\mathcal{P}_{01} \neq \emptyset$ (e.g., $L_1 = \{01\} \in \mathcal{P}_{01}$)
  - $\mathcal{P}_{01} \neq RE$ (e.g., $L_{ne} \notin \mathcal{P}_{01}$ because $01 \notin L_{ne}$ because 01 is not the code of a TM, but L_{ne} is in RE; recall: $L_{ne} = \{\langle M \rangle : L(M) \neq \emptyset\}$)

Rice's Theorem (Example 2)

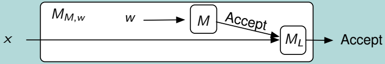
How about the property that the language contains String "01"?

- Let $\mathcal{P}_{01} = \{L \subseteq \Sigma : 01 \in L\}$, which is non-trivial:
  - $\mathcal{P}_{01} \neq \emptyset$ (e.g., $L_1 = \{01\} \in \mathcal{P}_{01}$)
  - $\mathcal{P}_{01} \neq RE$ (e.g., $L_{ne} \notin \mathcal{P}_{01}$ because $01 \notin L_{ne}$ because 01 is not the code of a TM, but L_{ne} is in RE; recall: $L_{ne} = \{\langle M \rangle : L(M) \neq \emptyset\}$)
- Thus, $L_{\mathcal{P}_{01}} = \{\langle M \rangle : L(M) \in \mathcal{P}_{01}\}$ is undecidable. In other words: We can't decide whether a given TM accepts a language that contains a 01.

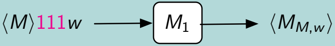
Rice's Theorem (Proof)

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 $\langle M \rangle 111 w$ and outputs $\langle M_{M,w} \rangle$. Note: M_1 **always** halts (even if M does not halt when input is w!)



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

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 the list length k.
  - We can reuse pairs as many times as we like.

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 the list length k.
  - 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 a solution.

Modified PCP (MPCP): Definition

- Suppose we are again 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

$$u_1 u_{i_1} \cdots u_{i_m}$$
$$= v_1 v_{i_1} \cdots v_{i_m}$$

Modified PCP (MPCP): Definition

- Suppose we are again 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

MPCP: Thoughts/Ideas before constructing a Proof

- So we want to prove that MPCP reduces to PCP.
- More specifically we need to:
  - Turn every MPCP problem into a PCP problem (with preserving solutions).
  - I.e., **how can we enforce PCP to always select the first element first?**

Thus, the problem we need to solve is:

- To make sure that that the first string gets selected first, but
- without making additional solutions available or cutting some out!

MPCP: Thoughts/Ideas before constructing a Proof

- So we want to prove that MPCP reduces to PCP.
- More specifically we need to:
  - Turn every MPCP problem into a PCP problem (with preserving solutions).
  - I.e., **how can we enforce PCP to always select the first element first?**

Thus, the problem we need to solve is:

- To make sure that that the first string gets selected first, but
- without making additional solutions available or cutting some out!

Initial thoughts:

- We add a new start symbol to u_1 and v_1 so that they match.
- But that still doesn't enforce that we start with them! ...

Outline of Proof of Theorem 9.8.1

- Given MPCP's lists $A = (u_1, \ldots, u_k)$ and $B = (v_1, \ldots, v_k)$. We now transform this into a PCP problem! Suppose that symbols $\diamond, \triangle$ are not in the strings of A and B.

Outline of Proof of Theorem 9.8.1

- Given MPCP's lists $A = (u_1, \ldots, u_k)$ and $B = (v_1, \ldots, v_k)$. We now transform this into a PCP problem! Suppose that symbols $\diamond, \triangle$ are not in the strings of A and B.
- Construct lists $C = (w_0, \ldots, w_{k+1})$ and $D = (x_0, \ldots, x_{k+1})$ for PCP as follows.
  - For $i = 1, \ldots, k$,
    - if $u_i = s_1 \ldots s_\ell$, then $w_i = \ s_1 \diamond s_2 \diamond \cdots \diamond s_\ell \diamond$ [$\diamond$ succeeds symbols]
    - if $v_i = s_1 \ldots s_\ell$, then $x_i = \diamond s_1 \diamond s_2 \diamond \cdots \diamond s_\ell$ [$\diamond$ precedes symbols]

Outline of Proof of Theorem 9.8.1

- Given MPCP's lists $A = (u_1, \ldots, u_k)$ and $B = (v_1, \ldots, v_k)$. We now transform this into a PCP problem! Suppose that symbols $\diamond, \triangle$ are not in the strings of A and B.
- Construct lists $C = (w_0, \ldots, w_{k+1})$ and $D = (x_0, \ldots, x_{k+1})$ for PCP as follows.
  - For $i = 1, \ldots, k$,
    - if $u_i = s_1 \ldots s_\ell$, then $w_i = \; s_1 \diamond s_2 \diamond \cdots \diamond s_\ell \diamond$ [$\diamond$ succeeds symbols]
    - if $v_i = s_1 \ldots s_\ell$, then $x_i = \diamond s_1 \diamond s_2 \diamond \cdots \diamond s_\ell$ [$\diamond$ precedes symbols]
  - $w_0 = \diamond w_1$ and $x_0 = x_1$. [Ensures any solution to PCP also starts with $i_1 = 1$]

Outline of Proof of Theorem 9.8.1

- Given MPCP's lists $A = (u_1, \ldots, u_k)$ and $B = (v_1, \ldots, v_k)$. We now transform this into a PCP problem! Suppose that symbols $\diamond, \triangle$ are not in the strings of A and B.
- Construct lists $C = (w_0, \ldots, w_{k+1})$ and $D = (x_0, \ldots, x_{k+1})$ for PCP as follows.
  - For $i = 1, \ldots, k$,
    - if $u_i = s_1 \ldots s_\ell$, then $w_i = \; s_1 \diamond s_2 \diamond \cdots \diamond s_\ell \diamond$ [$\diamond$ succeeds symbols]
    - if $v_i = s_1 \ldots s_\ell$, then $x_i = \diamond s_1 \diamond s_2 \diamond \cdots \diamond s_\ell$ [$\diamond$ precedes symbols]
  - $w_0 = \diamond w_1$ and $x_0 = x_1$. [Ensures any solution to PCP also starts with $i_1 = 1$]
  - $w_{k+1} = \triangle$ and $x_{k+1} = \diamond\triangle$. [Balances the extra $\diamond$]

Outline of Proof of Theorem 9.8.1

- Given MPCP's lists $A = (u_1, \ldots, u_k)$ and $B = (v_1, \ldots, v_k)$. We now transform this into a PCP problem! Suppose that symbols $\diamond, \triangle$ are not in the strings of A and B.
- Construct lists $C = (w_0, \ldots, w_{k+1})$ and $D = (x_0, \ldots, x_{k+1})$ for PCP as follows.
  - For $i = 1, \ldots, k$,
    - if $u_i = s_1 \ldots s_\ell$, then $w_i = s_1 \diamond s_2 \diamond \cdots \diamond s_\ell \diamond$ [$\diamond$ succeeds symbols]
    - if $v_i = s_1 \ldots s_\ell$, then $x_i = \diamond s_1 \diamond s_2 \diamond \cdots \diamond s_\ell$ [$\diamond$ precedes symbols]
  - $w_0 = \diamond w_1$ and $x_0 = x_1$. [Ensures any solution to PCP also starts with $i_1 = 1$]
  - $w_{k+1} = \triangle$ and $x_{k+1} = \diamond\triangle$. [Balances the extra $\diamond$]

A	B
110	110110
0011	00
0110	110

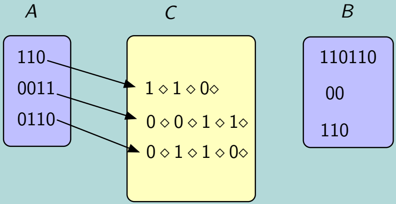
$$u_1 u_{i_1} \ldots u_{i_n} = v_1 v_{i_1} \ldots v_{i_n} \iff$$

$$\diamond w_1 w_{i_1} \cdots \diamond w_{i_n} = x_1 x_{i_1} \ldots x_{i_n} \diamond \iff$$

$$w_0 w_{i_1} \ldots w_{i_n} \triangle = x_0 x_{i_1} \ldots x_{i_n} \diamond \triangle$$

Outline of Proof of Theorem 9.8.1

- Given MPCP's lists $A = (u_1, \ldots, u_k)$ and $B = (v_1, \ldots, v_k)$. We now transform this into a PCP problem! Suppose that symbols $\diamond, \triangle$ are not in the strings of A and B.
- Construct lists $C = (w_0, \ldots, w_{k+1})$ and $D = (x_0, \ldots, x_{k+1})$ for PCP as follows.
  - For $i = 1, \ldots, k$,
    - if $u_i = s_1 \ldots s_\ell$, then $w_i = s_1 \diamond s_2 \diamond \cdots \diamond s_\ell \diamond$ [$\diamond$ succeeds symbols]
    - if $v_i = s_1 \ldots s_\ell$, then $x_i = \diamond s_1 \diamond s_2 \diamond \cdots \diamond s_\ell$ [$\diamond$ precedes symbols]
  - $w_0 = \diamond w_1$ and $x_0 = x_1$. [Ensures any solution to PCP also starts with $i_1 = 1$]
  - $w_{k+1} = \triangle$ and $x_{k+1} = \diamond\triangle$. [Balances the extra $\diamond$]



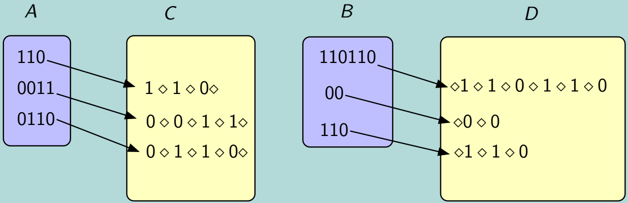
$$u_1 u_{i_1} \ldots u_{i_n} = v_1 v_{i_1} \ldots v_{i_n} \quad \Longleftrightarrow$$

$$\diamond w_1 w_{i_1} \cdots \diamond w_{i_n} = x_1 x_{i_1} \ldots x_{i_n} \diamond \quad \Longleftrightarrow$$

$$w_0 w_{i_1} \ldots w_{i_n} \triangle = x_0 x_{i_1} \ldots x_{i_n} \diamond \triangle$$

Outline of Proof of Theorem 9.8.1

- Given MPCP's lists $A = (u_1, \ldots, u_k)$ and $B = (v_1, \ldots, v_k)$. We now transform this into a PCP problem! Suppose that symbols $\diamond, \triangle$ are not in the strings of A and B.
- Construct lists $C = (w_0, \ldots, w_{k+1})$ and $D = (x_0, \ldots, x_{k+1})$ for PCP as follows.
  - For $i = 1, \ldots, k$,
    - if $u_i = s_1 \ldots s_\ell$, then $w_i = s_1 \diamond s_2 \diamond \cdots \diamond s_\ell \diamond$ [$\diamond$ succeeds symbols]
    - if $v_i = s_1 \ldots s_\ell$, then $x_i = \diamond s_1 \diamond s_2 \diamond \cdots \diamond s_\ell$ [$\diamond$ precedes symbols]
  - $w_0 = \diamond w_1$ and $x_0 = x_1$. [Ensures any solution to PCP also starts with $i_1 = 1$]
  - $w_{k+1} = \triangle$ and $x_{k+1} = \diamond\triangle$. [Balances the extra $\diamond$]



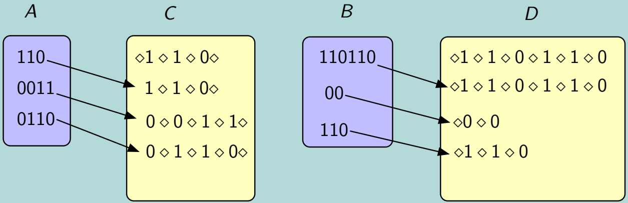
$$u_1 u_{i_1} \ldots u_{i_n} = v_1 v_{i_1} \ldots v_{i_n} \iff$$

$$\diamond w_1 w_{i_1} \cdots \diamond w_{i_n} = x_1 x_{i_1} \ldots x_{i_n} \diamond \iff$$

$$w_0 w_{i_1} \ldots w_{i_n} \triangle = x_0 x_{i_1} \ldots x_{i_n} \diamond \triangle$$

Outline of Proof of Theorem 9.8.1

- Given MPCP's lists $A = (u_1, \dots, u_k)$ and $B = (v_1, \dots, v_k)$. We now transform this into a PCP problem! Suppose that symbols $\diamond, \triangle$ are not in the strings of A and B.
- Construct lists $C = (w_0, \dots, w_{k+1})$ and $D = (x_0, \dots, x_{k+1})$ for PCP as follows.
  - For $i = 1, \dots, k$,
    - if $u_i = s_1 \dots s_\ell$, then $w_i = s_1 \diamond s_2 \diamond \cdots \diamond s_\ell \diamond$ [$\diamond$ succeeds symbols]
    - if $v_i = s_1 \dots s_\ell$, then $x_i = \diamond s_1 \diamond s_2 \diamond \cdots \diamond s_\ell$ [$\diamond$ precedes symbols]
  - $w_0 = \diamond w_1$ and $x_0 = x_1$. [Ensures any solution to PCP also starts with $i_1 = 1$]
  - $w_{k+1} = \triangle$ and $x_{k+1} = \diamond\triangle$. [Balances the extra $\diamond$]



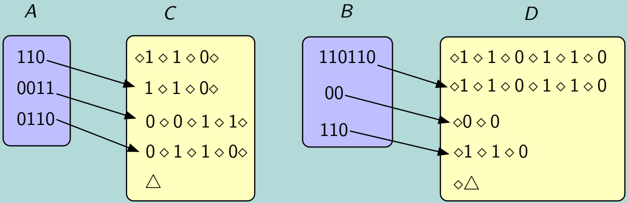
$$u_1 u_{i_1} \dots u_{i_n} = v_1 v_{i_1} \dots v_{i_n} \iff$$

$$\diamond w_1 w_{i_1} \cdots \diamond w_{i_n} = x_1 x_{i_1} \dots x_{i_n} \diamond \iff$$

$$w_0 w_{i_1} \dots w_{i_n} \triangle = x_0 x_{i_1} \dots x_{i_n} \diamond \triangle$$

Outline of Proof of Theorem 9.8.1

- Given MPCP's lists $A = (u_1, \ldots, u_k)$ and $B = (v_1, \ldots, v_k)$. We now transform this into a PCP problem! Suppose that symbols $\diamond, \triangle$ are not in the strings of A and B.
- Construct lists $C = (w_0, \ldots, w_{k+1})$ and $D = (x_0, \ldots, x_{k+1})$ for PCP as follows.
  - For $i = 1, \ldots, k$,
    - if $u_i = s_1 \ldots s_\ell$, then $w_i = s_1 \diamond s_2 \diamond \cdots \diamond s_\ell \diamond$ [$\diamond$ succeeds symbols]
    - if $v_i = s_1 \ldots s_\ell$, then $x_i = \diamond s_1 \diamond s_2 \diamond \cdots \diamond s_\ell$ [$\diamond$ precedes symbols]
  - $w_0 = \diamond w_1$ and $x_0 = x_1$. [Ensures any solution to PCP also starts with $i_1 = 1$]
  - $w_{k+1} = \triangle$ and $x_{k+1} = \diamond\triangle$. [Balances the extra $\diamond$]



$$u_1 u_{i_1} \ldots u_{i_n} = v_1 v_{i_1} \ldots v_{i_n} \quad \Longleftrightarrow$$

$$\diamond w_1 w_{i_1} \cdots \diamond w_{i_n} = x_1 x_{i_1} \ldots x_{i_n} \diamond \quad \Longleftrightarrow$$

$$w_0 w_{i_1} \ldots w_{i_n} \triangle = x_0 x_{i_1} \ldots x_{i_n} \diamond \triangle$$

PCP is undecidable

Theorem 9.8.2

PCP is undecidable.

Outline of Proof of Theorem 9.8.2 (Overview)

We reduce L_u to MPCP (and did already MPCP to PCP).

PCP is undecidable

Theorem 9.8.2

PCP is undecidable.

Outline of Proof of Theorem 9.8.2 (Overview)

We reduce L_u to MPCP (and did already MPCP to PCP). We will show:

- M accepts $w \iff$ a solution to the MPCP exists.

PCP is undecidable

Theorem 9.8.2

PCP is undecidable.

Outline of Proof of Theorem 9.8.2 (Overview)

We reduce L_u to MPCP (and did already MPCP to PCP). We will show:

- M accepts $w \iff$ a solution to the MPCP exists.
- If MPCP were decidable, then L_u would be too (i.e., recursive), which it isn't.
- Hence, MPCP is undecidable. [following Theorem 9.6.1]

PCP is undecidable

Theorem 9.8.2

PCP is undecidable.

Outline of Proof of Theorem 9.8.2 (Overview)

We reduce L_u to MPCP (and did already MPCP to PCP). We will show:

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

PCP is undecidable

Theorem 9.8.2

PCP is undecidable.

Outline of Proof of Theorem 9.8.2 (Overview)

We reduce L_u to MPCP (and did already MPCP to PCP). We will show:

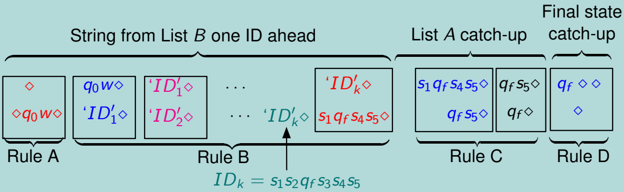
- M accepts $w \iff$ a solution to the MPCP exists.
- If MPCP were decidable, then L_u would be too (i.e., recursive), which it isn't.
- Hence, MPCP is undecidable. [following Theorem 9.6.1]
- Since MPCP is undecidable, PCP is also undecidable. [following Theorem 9.6.1]

So the hard work is to solve/model $\langle M \rangle 111w \in L_u$ via MPCP!

PCP is undecidable (More detailed proof at the end)

Outline of Proof of Theorem 9.8.2 (Overview)

Abstract overview of existing pairs in the constructed MPCP:



The rules $A \ldots, D$ are in the appendix.

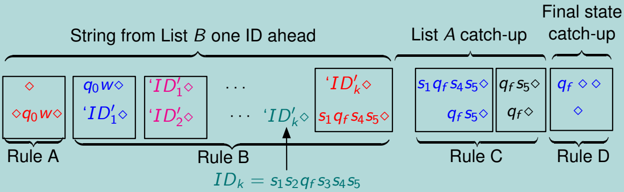
The overall idea is as follows:

- We have two lines of strings (which should match in the end).

PCP is undecidable (More detailed proof at the end)

Outline of Proof of Theorem 9.8.2 (Overview)

Abstract overview of existing pairs in the constructed MPCP:



The rules $A \ldots, D$ are in the appendix.

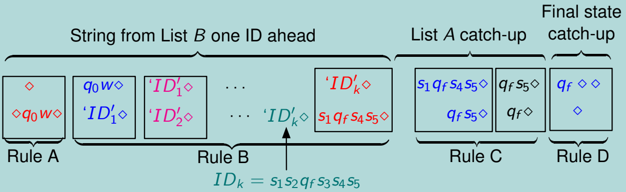
The overall idea is as follows:

- We have two lines of strings (which should match in the end).
- The first pair we construct is "empty" in the first line/entry and contains the TM's start configuration in the second. (Rule A)

PCP is undecidable (More detailed proof at the end)

Outline of Proof of Theorem 9.8.2 (Overview)

Abstract overview of existing pairs in the constructed MPCP:



The rules $A \ldots, D$ are in the appendix.

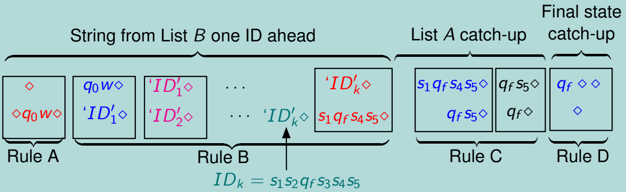
The overall idea is as follows:

- We have two lines of strings (which should match in the end).
- The first pair we construct is "empty" in the first line/entry and contains the TM's start configuration in the second. (Rule A)
- We construct a pair for every valid TM transition! (Rule B) In such a pair, the first line/entry is the old configuaration and the second the new.

PCP is undecidable (More detailed proof at the end)

Outline of Proof of Theorem 9.8.2 (Overview)

Abstract overview of existing pairs in the constructed MPCP:



The rules $A \ldots, D$ are in the appendix.

The overall idea is as follows:

- We have two lines of strings (which should match in the end).
- The first pair we construct is "empty" in the first line/entry and contains the TM's start configuration in the second. (Rule A)
- We construct a pair for every valid TM transition! (Rule B) In such a pair, the first line/entry is the old configuaration and the second the new.
- We have/need a few more rules to make all strings equal and deal with final states. Note how we have to move the first line to get matchings strings. (Rules C, D)

PCP is undecidable (More detailed proof at the end)

Proof of Theorem 9.8.2 (Short Example)

Before we look at an example, recap:

- A TM ID looks as: $X_1 \ldots, X_{i-1} q X_i \ldots X_\ell$ where X_i is below the head.

Now, with TM's start state q_0 and initial tape $w = s_1 s_2 s_3$ let:

- Word in line 1: $\diamond$
- Word in line 2: $\diamond q_0 s_1 s_2 s_3 \diamond$

We get this by our first pair, created by Rule A:

- First entry in 1st list: $\diamond$
- First entry in 2nd list: $\diamond q_0 s_1 s_2 s_3 \diamond$

PCP is undecidable (More detailed proof at the end)

Proof of Theorem 9.8.2 (Short Example)

Before we look at an example, recap:

- A TM ID looks as: $X_1 \ldots, X_{i-1} q X_i \ldots X_\ell$ where X_i is below the head.

Now, with TM's start state q_0 and initial tape $w = s_1 s_2 s_3$ let:

- Word in line 1: $\diamond$
- Word in line 2: $\diamond q_0 s_1 s_2 s_3 \diamond$

We get this by our first pair, created by Rule A:

- First entry in 1st list: $\diamond$
- First entry in 2nd list: $\diamond q_0 s_1 s_2 s_3 \diamond$

What's next? Create the transitions! (Via Rules in B)

- Assume $\delta(q_0, s_1) = (p, t_1, R)$, then $q_0 s_1 s_2 s_3 \underset{M}{\vdash}$
- So we put this into a new pair!

PCP is undecidable (More detailed proof at the end)

Proof of Theorem 9.8.2 (Short Example)

Before we look at an example, recap:

- A TM ID looks as: $X_1 \ldots, X_{i-1} q X_i \ldots X_\ell$ where X_i is below the head.

Now, with TM's start state q_0 and initial tape $w = s_1 s_2 s_3$ let:

- Word in line 1: $\diamond$
- Word in line 2: $\diamond q_0 s_1 s_2 s_3 \diamond$

We get this by our first pair, created by Rule A:

- First entry in 1st list: $\diamond$
- First entry in 2nd list: $\diamond q_0 s_1 s_2 s_3 \diamond$

What's next? Create the transitions! (Via Rules in B)

- Assume $\delta(q_0, s_1) = (p, t_1, R)$, then $q_0 s_1 s_2 s_3 \underset{M}{\vdash} t_1 p s_2 s_3$
- So we put this into a new pair!

PCP is undecidable (More detailed proof at the end)

Proof of Theorem 9.8.2 (Short Example)

Before we look at an example, recap:

- A TM ID looks as: $X_1 \ldots, X_{i-1} q X_i \ldots X_\ell$ where X_i is below the head.

Now, with TM's start state q_0 and initial tape $w = s_1 s_2 s_3$ let:

- Word in line 1: $\diamond q_0 s_1$
- Word in line 2: $\diamond q_0 s_1 s_2 s_3 \diamond t_1 p$

We get this by another pair, created by Rule B:

- Entry in 1st list: $q_0 s_1$ since $\delta(q_0, s_1) = (p, t_1, R)$
- Entry in 2nd list: $t_1 p$ and thus $q_0 s_1 s_2 s_3 \underset{M}{\vdash} t_1 p s_2 s_3$

PCP is undecidable (More detailed proof at the end)

Proof of Theorem 9.8.2 (Short Example)

Before we look at an example, recap:

- A TM ID looks as: $X_1 \ldots, X_{i-1} q X_i \ldots X_\ell$ where X_i is below the head.

Now, with TM's start state q_0 and initial tape $w = s_1 s_2 s_3$ let:

- Word in line 1: $\diamond q_0 s_1$
- Word in line 2: $\diamond q_0 s_1 s_2 s_3 \diamond t_1 p$

We get this by another pair, created by Rule B:

- Entry in 1st list: $q_0 s_1$ since $\delta(q_0, s_1) = (p, t_1, R)$
- Entry in 2nd list: $t_1 p$ and thus $q_0 s_1 s_2 s_3 \underset{M}{\vdash} t_1 p s_2 s_3$

What's next? The remaining symbols from last configuration are missing...

- We add a pair (s, s) for all $s \in \Gamma$ (Rule I)
- and one pair $(\diamond, \diamond)$ (Rule I)

PCP is undecidable (More detailed proof at the end)

Proof of Theorem 9.8.2 (Short Example)

Before we look at an example, recap:

- A TM ID looks as: $X_1 \ldots, X_{i-1} q X_i \ldots X_\ell$ where X_i is below the head.

Now, with TM's start state q_0 and initial tape $w = s_1 s_2 s_3$ let:

- Word in line 1: $\diamond q_0 s_1 s_2 s_3 \diamond$
- Word in line 2: $\diamond q_0 s_1 s_2 s_3 \diamond t_1 p s_2 s_3 \diamond$

We get this by several new pairs, created by Rule I:

- $(s_0, s_0), (s_1, s_1), (s_2, s_2), \ldots$ (for all $s \in \Gamma$)
- and the pair $(\diamond, \diamond)$

PCP is undecidable (More detailed proof at the end)

Proof of Theorem 9.8.2 (Short Example)

Before we look at an example, recap:

- A TM ID looks as: $X_1 \ldots, X_{i-1} q X_i \ldots X_\ell$ where X_i is below the head.

Now, with TM's start state q_0 and initial tape $w = s_1 s_2 s_3$ let:

- Word in line 1: $\diamond q_0 s_1 s_2 s_3 \diamond$
- Word in line 2: $\diamond q_0 s_1 s_2 s_3 \diamond t_1 p s_2 s_3 \diamond$

We get this by several new pairs, created by Rule I:

- $(s_0, s_0), (s_1, s_1), (s_2, s_2), \ldots$ (for all $s \in \Gamma$)
- and the pair $(\diamond, \diamond)$

What's next? We continue! Next transition!

- Assume $\delta(p, s_2) = (r, t_2, L)$, then $t_1 p s_2 s_3 \underset{M}{\vdash}$
- So we put this into a new pair!

PCP is undecidable (More detailed proof at the end)

Proof of Theorem 9.8.2 (Short Example)

Before we look at an example, recap:

- A TM ID looks as: $X_1 \ldots, X_{i-1} q X_i \ldots X_\ell$ where X_i is below the head.

Now, with TM's start state q_0 and initial tape $w = s_1 s_2 s_3$ let:

- Word in line 1: $\diamond q_0 s_1 s_2 s_3 \diamond$
- Word in line 2: $\diamond q_0 s_1 s_2 s_3 \diamond t_1 p s_2 s_3 \diamond$

We get this by several new pairs, created by Rule I:

- $(s_0, s_0), (s_1, s_1), (s_2, s_2), \ldots$ (for all $s \in \Gamma$)
- and the pair $(\diamond, \diamond)$

What's next? We continue! Next transition!

- Assume $\delta(p, s_2) = (r, t_2, L)$, then $t_1 p s_2 s_3 \underset{M}{\vdash} r t_1 t_2 s_3$
- So we put this into a new pair!

PCP is undecidable (More detailed proof at the end)

Proof of Theorem 9.8.2 (Short Example)

Before we look at an example, recap:

- A TM ID looks as: $X_1 \ldots, X_{i-1} q X_i \ldots X_\ell$ where X_i is below the head.

Now, with TM's start state q_0 and initial tape $w = s_1 s_2 s_3$ let:

- Word in line 1: $\diamond q_0 s_1 s_2 s_3 \diamond t_1 p s_2$
- Word in line 2: $\diamond q_0 s_1 s_2 s_3 \diamond t_1 p s_2 s_3 \diamond r t_1 t_2$

We get this by another pair, created by Rule B:

- Entry in 1st list: $t_1 p s_2$ — since $\delta(p, s_2) = (r, t_2, L)$
- Entry in 2nd list: $r t_1 t_2$ — and thus $t_1 p s_2 s_3 \underset{M}{\vdash} r t_1 t_2 s_3$

PCP is undecidable (More detailed proof at the end)

Proof of Theorem 9.8.2 (Short Example)

Before we look at an example, recap:

- A TM ID looks as: $X_1 \ldots, X_{i-1} q X_i \ldots X_\ell$ where X_i is below the head.

Now, with TM's start state q_0 and initial tape $w = s_1 s_2 s_3$ let:

- Word in line 1: $\diamond q_0 s_1 s_2 s_3 \diamond t_1 p s_2$
- Word in line 2: $\diamond q_0 s_1 s_2 s_3 \diamond t_1 p s_2 s_3 \diamond r t_1 t_2$

We get this by another pair, created by Rule B:

- Entry in 1st list: $t_1 p s_2$ since $\delta(p, s_2) = (r, t_2, L)$
- Entry in 2nd list: $r t_1 t_2$ and thus $t_1 p s_2 s_3 \underset{M}{\vdash} r t_1 t_2 s_3$

What's next?

- First, we again add the missing symbols, until
- eventually we find a final state. We have more rules for that (see appendix).

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

Theorem 9.9.1

The problem if a CFG is ambiguous is undecidable.

Outline of Proof of Theorem 9.8.2

› We'll reduce ... which one? (1) PCP to CFG or (2) CFG to PCP?

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

Theorem 9.9.1

The problem if a CFG is ambiguous is undecidable.

Outline of Proof of Theorem 9.8.2

- We'll reduce every instance of a PCP problem to a CFG.
- Given a PCP problem with $A = (w_1, \ldots, w_k)$ 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

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

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

Theorem 9.9.1

The problem if a CFG is ambiguous is undecidable.

Outline of Proof of Theorem 9.8.2

- We'll reduce every instance of a PCP problem to a CFG.
- Given a PCP problem with $A = (w_1, \ldots, w_k)$ 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

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

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

Overview of (Some) Undecidable Problems Concerning CFGs

- Given a CFG G, is it ambiguous? (We just had that.)

Overview of (Some) Undecidable Problems Concerning CFGs

- Given a CFG G, is it ambiguous? (We just had that.)
- Given CFL L, is it inherently ambiguous?

Overview of (Some) Undecidable Problems Concerning CFGs

- Given a CFG G, is it ambiguous? (We just had that.)
- Given CFL L, is it inherently ambiguous?
- Given CFGs G_1 and G_2, is $L(G_1) \cap L(G_2) = \emptyset$?
  (As mentioned before, this is used to show that HTN planning is undeciable)

Overview of (Some) Undecidable Problems Concerning CFGs

- Given a CFG G, is it ambiguous? (We just had that.)
- Given CFL L, is it inherently ambiguous?
- Given CFGs G_1 and G_2, is $L(G_1) \cap L(G_2) = \emptyset$?
  (As mentioned before, this is used to show that HTN planning is undeciable)
- Given CFGs G_1 and G_2, is $L(G_1) \subseteq L(G_2)$?

Overview of (Some) Undecidable Problems Concerning CFGs

- Given a CFG G, is it ambiguous? (We just had that.)
- Given CFL L, is it inherently ambiguous?
- Given CFGs G_1 and G_2, is $L(G_1) \cap L(G_2) = \emptyset$?
  (As mentioned before, this is used to show that HTN planning is undeciable)
- Given CFGs G_1 and G_2, is $L(G_1) \subseteq L(G_2)$?
- Given CFGs G_1 and G_2, is $L(G_1) = L(G_2)$?

Overview of (Some) Undecidable Problems Concerning CFGs

- Given a CFG G, is it ambiguous? (We just had that.)
- Given CFL L, is it inherently ambiguous?
- Given CFGs G_1 and G_2, is $L(G_1) \cap L(G_2) = \emptyset$?
  (As mentioned before, this is used to show that HTN planning is undeciable)
- 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 $L(G) = L$?

Overview of (Some) Undecidable Problems Concerning CFGs

- Given a CFG G, is it ambiguous? (We just had that.)
- Given CFL L, is it inherently ambiguous?
- Given CFGs G_1 and G_2, is $L(G_1) \cap L(G_2) = \emptyset$?
  (As mentioned before, this is used to show that HTN planning is undeciable)
- 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 $L \subseteq L(G)$?

Overview of (Some) Undecidable Problems Concerning CFGs

- Given a CFG G, is it ambiguous? (We just had that.)
- Given CFL L, is it inherently ambiguous?
- Given CFGs G_1 and G_2, is $L(G_1) \cap L(G_2) = \emptyset$?
  (As mentioned before, this is used to show that HTN planning is undeciable)
- 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 $L(G) = \Sigma^*$?

PCP is undecidable

Proof Details of Theorem 9.8.2 (Rule Definitions)

- For the proof we construct an MPCP for each TM M and input w.

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

Rule I: Add corresponding pairs (X, X) (for 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

Proof Details of Theorem 9.8.2 (Construction/Explanation)

- 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 an 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.