DECIDABLE

A problem P is *decidable* if it can be solved by a Turing machine T that always halt. (We say that P has an effective algorithm.)

Note that the corresponding language of a decidable problem is *recursive*.

UNDECIDABLE

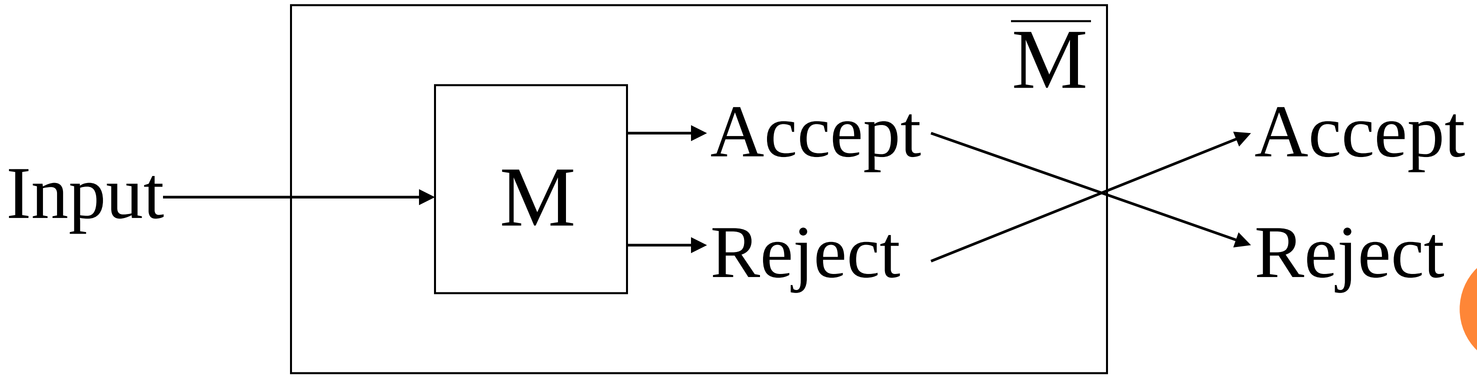
A problem is *undecidable* if it cannot be solved by any Turing machine that halts on all inputs.

Note that the corresponding language of an undecidable problem is *non-recursive*.

Complements of Recursive Languages

Theorem: If L is a recursive language, $\overline{L}$ is also recursive.

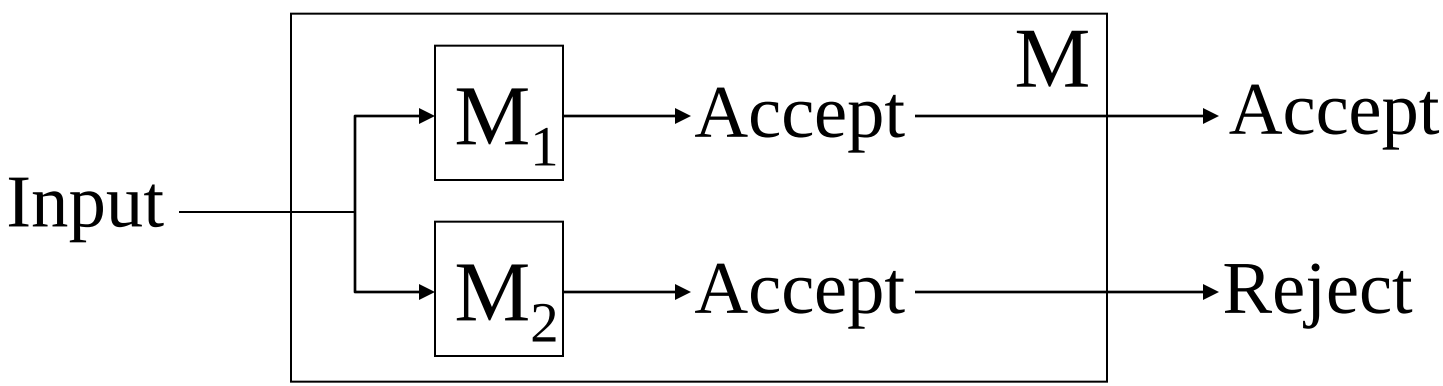
Proof: Let M be a TM for L that always halt. We can construct another TM $\overline{M}$ from M for $\overline{L}$ that always halts as follows:



COMPLEMENTS OF RE LANGUAGES

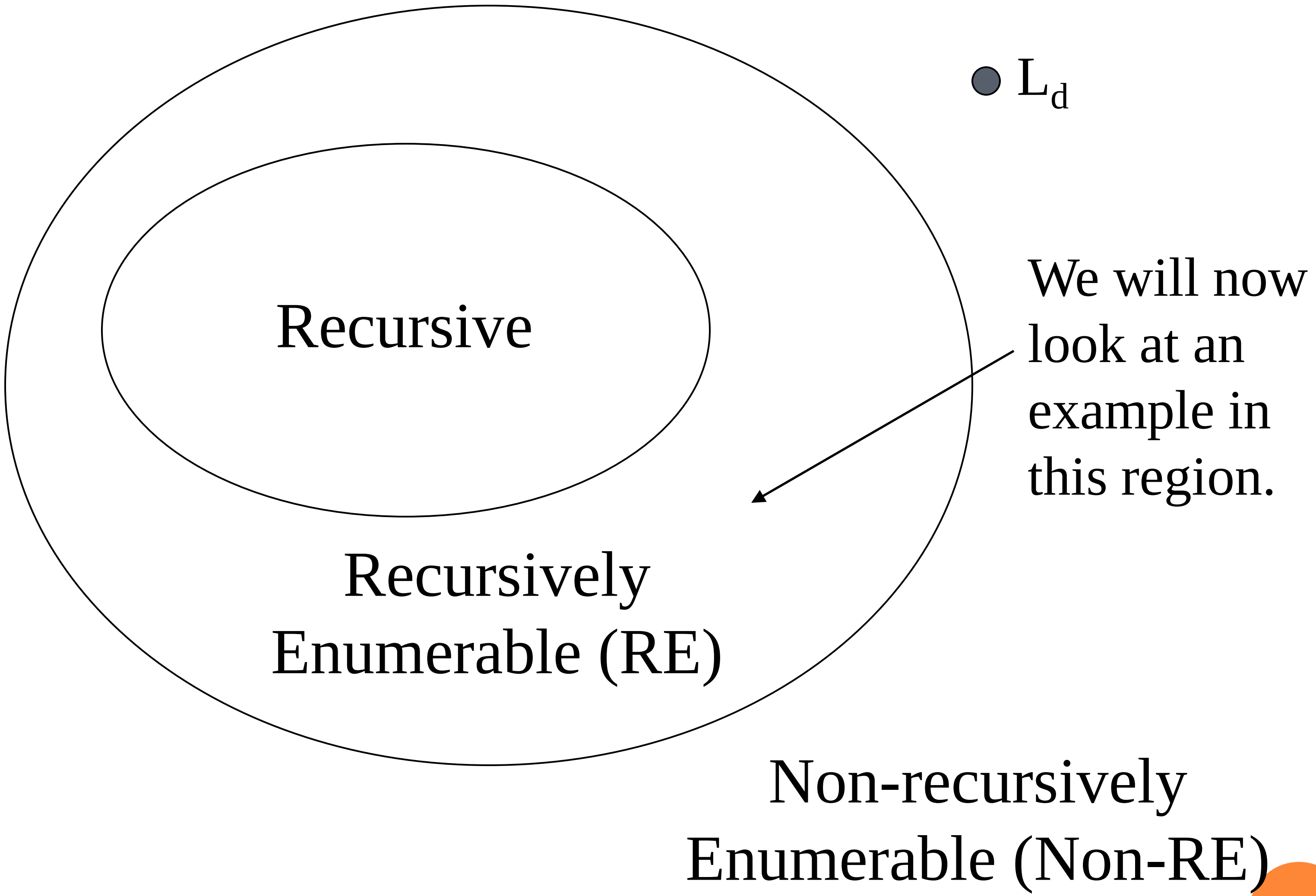
Theorem: If both a language L and its complement L are RE, L is recursive.

Proof: Let M_1 and M_2 be TM for L and L respectively. We can construct a TM M from M_1 and M_2 for L that always halt as follows:



A Non-recursive RE Language

- We are going to give an example of a RE language that is not recursive, i.e., a language L that can be accepted by a TM, but there is no TM for L that always halt.
- Again, we need to make use of the binary encoding of a TM.


L_d
We will now look at an example in this region.
Recursive
Recursively Enumerable (RE)
Non-recursively Enumerable (Non-RE)

A Non-recursive RE Language

- Recall that we can encode each TM uniquely as a binary number and enumerate all TM's as T_1, T_2, …, T_k, … where the encoded value of the k^{th} TM, i.e., T_k, is k.
- Consider the language L_u:

$$L_u = \{(k, w) \mid T_k \text{ accepts input } w\}$$

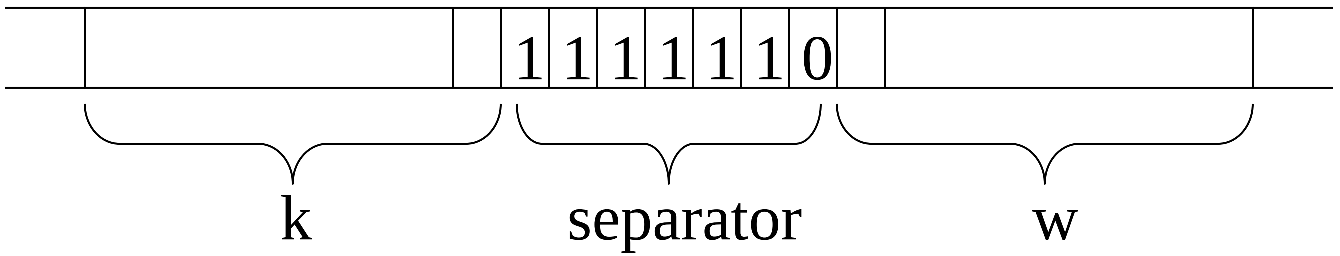
This is called the *universal language*.

Universal Language

- Note that designing a TM to recognize L_u is the same as solving the problem of *given k and w, decide whether T_k accepts w as its input*.
- We are going to show that L_u is RE but non-recursive, i.e., L_u can be accepted by a TM, but there is no TM for L_u that always halt.

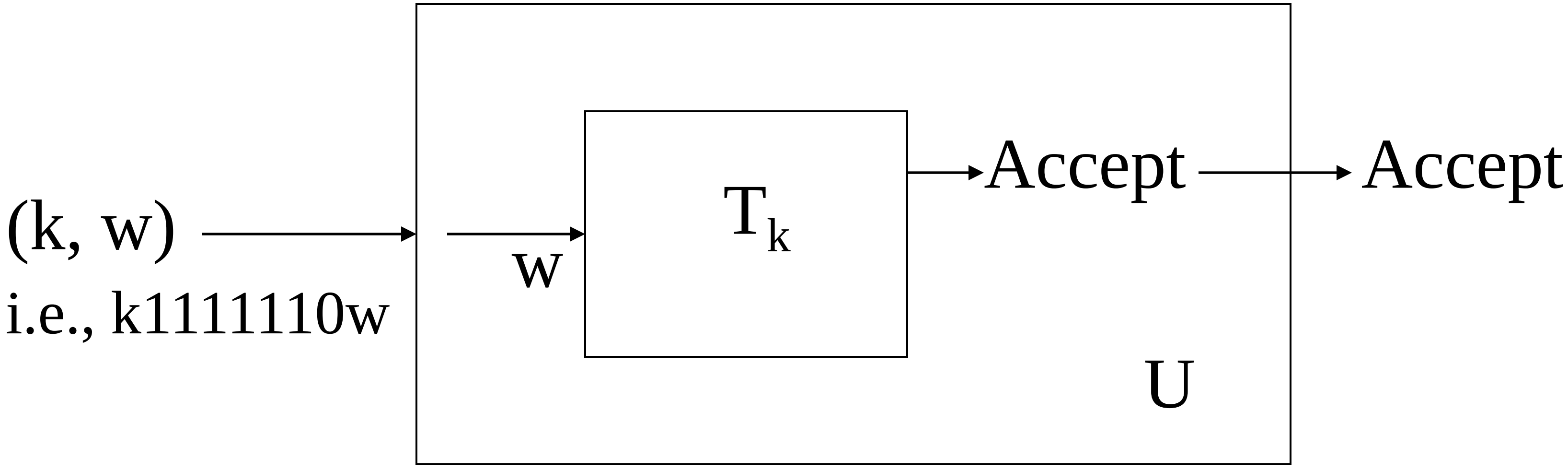
UNIVERSAL TURING MACHINE

- To show that L_u is RE, we construct a TM U, called the *universal Turing machine*, such that $L_u = L(U)$.
- U is designed in such a way that given k and w, it will mimic the operation of T_k on input w:



U will move back and forth to mimic T_k on input w.

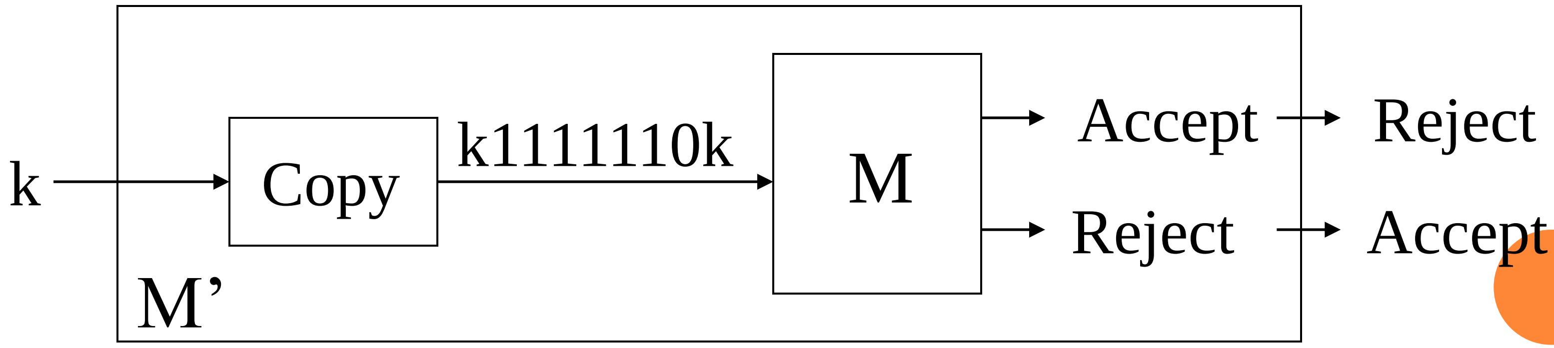
UNIVERSAL TURING MACHINE



Why cannot we use a similar method to construct a TM for L_d?

Universal Language

- Since there is a TM that accepts L_u, L_u is RE. We are going to show that L_u is non-recursive.
- If L_u is recursive, there is a TM M for L_u that always halt. Then, we can construct a TM M' for L_d as follows:



A Non-recursive RE Language

- Since we have already shown that L_d is non-recursively enumerable, so M' does not exist and there is no such M.
- Therefore the universal language is recursively enumerable but non-recursive.

HALTING PROBLEM

Consider the halting problem:

Given (k,w), determine if T_k halts on w.

It's corresponding language is:

$$L_h = \{ (k, w) \mid T_k \text{ halts on input } w\}$$

The halting problem is also undecidable, i.e., L_h is non-recursive. To show this, we can make use of the universal language problem.

Halting Problem

- We want to show that if the halting problem can be solved (decidable), the universal language problem can also be solved.
- So we will try to reduce an instance (a particular problem) in L_u to an instance in L_h in such a way that if we know the answer for the latter, we will know the answer for the former.

Class Discussion

Consider a particular instance (k,w) in L_u, i.e., we want to determine if T_k will accept w. Construct an instance I=(k',w') in L_h from (k,w) so that if we know whether $T_{k'}$ will halt on w', we will know whether T_k will accept w.

HALTING PROBLEM

Therefore, if we have a method to solve the halting problem, we can also solve the universal language problem. (Since for any particular instance I of the universal language problem, we can construct an instance of the halting problem, solve it and get the answer for I.) However, since the universal problem is undecidable, we can conclude that the halting problem is also undecidable.

Modified Post Correspondence Problem

- We have seen an undecidable problem, that is, given a Turing machine M and an input w, determine whether M will accept w (universal language problem).
- We will study another undecidable problem that is not related to Turing machine directly.

Modified Post Correspondence Problem (MPCP)

Given two lists A and B:

$A = w_1, w_2, \ldots, w_k \quad B = x_1, x_2, \ldots, x_k$

The problem is to determine if there is a sequence of one or more integers $i_1, i_2, \ldots, i_m$ such that:

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

(w_i, x_i) is called a corresponding pair.

EXAMPLE

	A	B
i	w_i	x_i
1	11	1
2	1	111
3	0111	10
4	10	0

This MPCP instance has a solution: 3, 2, 2, 4:

$$w_1w_3w_2w_2w_4 = x_1x_3x_2x_2x_4 = 1101111110$$

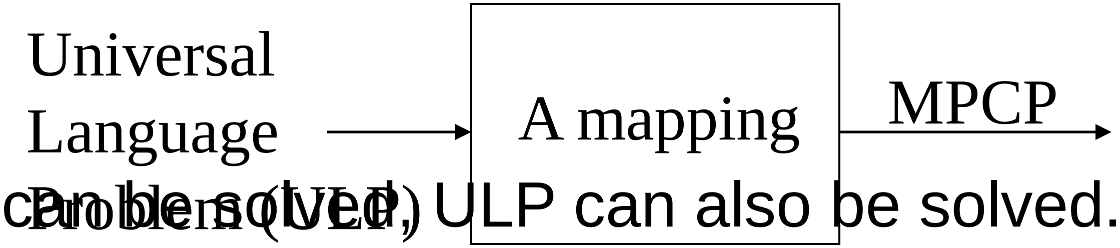
Class Discussion

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

Does this MPCP instance have a solution?

UNDECIDABILITY OF PCP

To show that MPCP is undecidable, we will reduce the universal language problem (ULP) to MPCP:

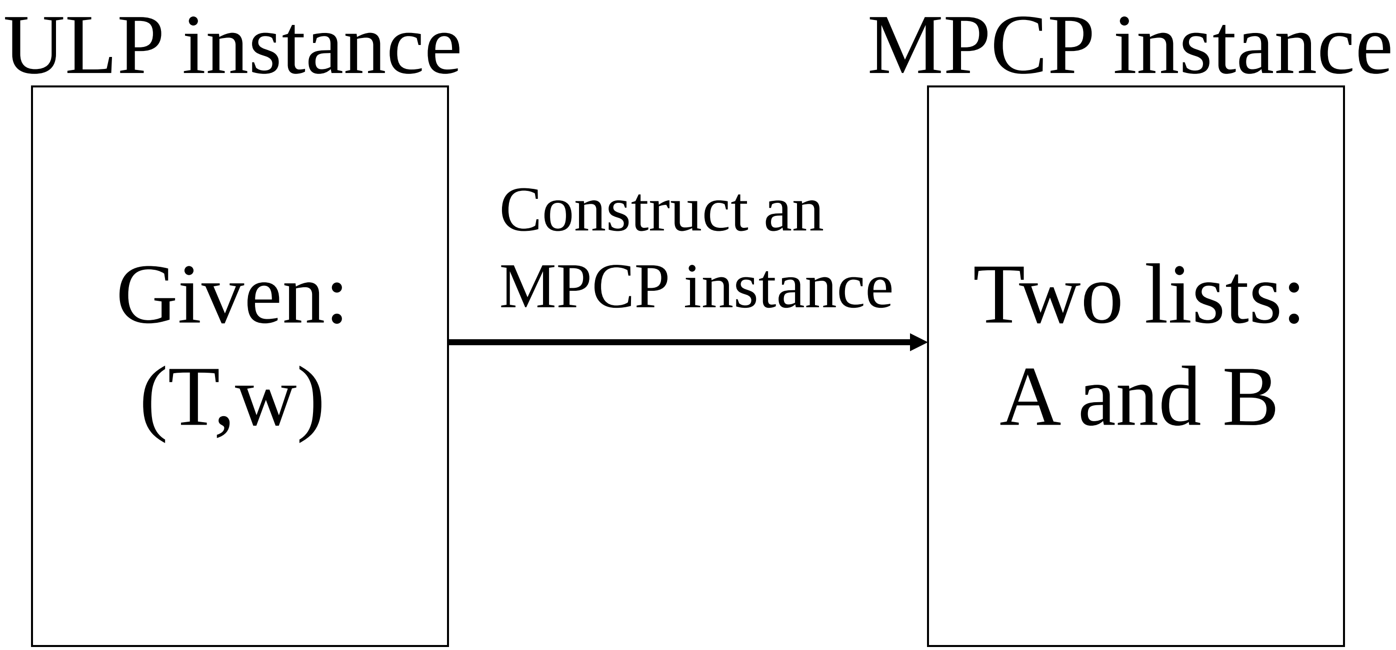


If MPCP can be solved, ULP can also be solved. Since we have already shown that ULP is un-decidable, MPCP must also be undecidable.

Mapping ULP to MPCP

- Mapping a universal language problem instance to an MPCP instance is not as easy.
- In a ULP instance, we are given a Turing machine M and an input w, we want to determine if M will accept w. To map a ULP instance to an MPCP instance success-fully, the mapped MPCP instance should have a solution if and only if M accepts w.

MAPPING ULP TO MPCP



If T accepts w, the two lists can be matched.
Otherwise, the two lists cannot be matched.

MAPPING ULP TO MPCP

- We assume that the input Turing machine T:
  - Never prints a blank
  - Never moves left from its initial head position.
- These assumptions can be made because:
  - **Theorem** (p.346 in Textbook): Every language accepted by a TM M_2 will also be accepted by a TM M_1 with the following restrictions: (1) M_1's head never moves left from its initial position. (2) M_1 never writes a blank.

Mapping ULP to MPCP

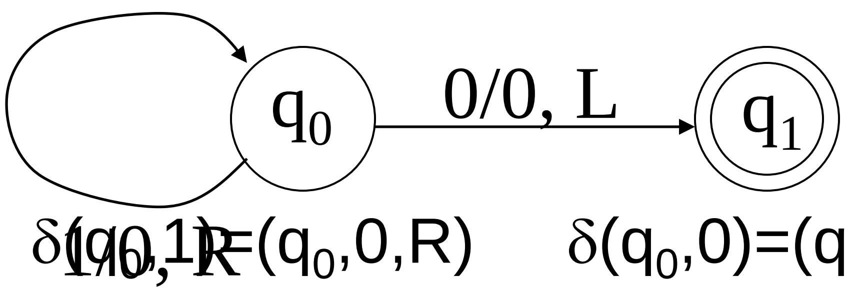
Given T and w, the idea is to map the transition function of T to strings in the two lists in such a way that a matching of the two lists will correspond to a concatenation of the tape contents at each time step.

We will illustrate this with an example first.

Example of ULP to MPCP

- Consider the following Turing machine:

$$T = (\{q_0, q_1\},\{0,1\},\{0,1,\#\}, \delta, q_0, \#, \{q_1\})$$



$\delta(q_0,1)=(q_0,0,R)$ $\delta(q_0,0)=(q_1,0,L)$

- Consider input w=110.

Example of ULP to MPCP

- Now we will construct an MPCP instance from T and w. There are five types of strings in list A and B:
- Starting string (first pair):

List A	List B
#	$\#q_0110\#$

Example of ULP to MPCP

- Strings from the transition function δ:

List A	List B	
$q_0 1$	$0 q_0$	(from $\delta(q_0,1)=(q_0,0,R)$)
$0 q_0 0$	$q_1 00$	(from $\delta(q_0,0)=(q_1,0,L)$)
$1 q_0 0$	$q_1 10$	(from $\delta(q_0,0)=(q_1,0,L)$)

Example of ULP to MPCP

- Strings for copying:

List A	List B
#	#
0	0
1	1

Example of ULP to MPCP

- Strings for consuming the tape symbols at the end:

List A	List B	List A	List B
$0q_1$	q_1	$0q_11$	q_1
$1q_1$	q_1	$1q_10$	q_1
q_10	q_1	$0q_10$	q_1
q_11	q_1	$1q_10$	q_1

EXAMPLE OF ULP TO MPCP

- Ending string:

List A	List B
q_1##	#

Now, we have constructed an MPCP instance.

Example of ULP to MPCP

	List A	List B		List A	List B
1.	#	$\#q_0110\#$	9.	$0q_1$	q_1
2.	q_01	$0q_0$	10.	$1q_1$	q_1
3.	$0q_00$	q_100	11.	q_10	q_1
4.	$1q_00$	q_110	12.	q_11	q_1
5.	#	#	13.	$0q_11$	q_1
6.	0	0	14.	$1q_10$	q_1
7.	1	1	15.	$0q_10$	q_1
8.	$q_1\#\#$	#	16.	$1q_10$	q_1

EXAMPLE OF ULP TO MPCP

- This ULP instance has a solution:

  $q_0 110 \rightarrow 0q_0 10 \rightarrow 00q_0 0 \rightarrow 0q_1 00$ (halt)

- Does this MPCP instance has a solution?

List A:

$\# \mid q_0 1 \mid 1 \mid 0 \# \mid 0 \mid q_0 1 \mid 0 \# \mid 0 \mid 0 \; q_0 0 \mid \# \mid 0 \; q_1 0 \mid 0 \mid \# \mid q_1 0 \mid \# \mid q_1 \# \# \mid$

List B:

$\# \; q_0 1 \; 1 \; 0 \; \# \mid 0 \; q_0 \mid 1 \; 0 \mid \# \mid 0 \mid 0 \; q_0 \mid 0 \mid \# \mid 0 \mid q_1 \; 0 \; 0 \mid \# \mid q_1 \mid 0 \mid \# \mid q_1 \# \mid \# \mid$

The solution is the sequence of indices:

2, 7, 6, 5, 6, 2, 6, 5, 6, 3, 5, 15, 6, 5, 11, 5, 8

Class Discussion

Consider the input w = 101. Construct the corresponding MPCP instance I and show that T will accept w by giving a solution to I.

Class Discussion (cont'd)

	List A	List B		List A	List B
1.	#	$\#q_0101\#$	9.	$0q_1$	q_1
2.	q_01	$0q_0$	10.	$1q_1$	q_1
3.	$0q_00$	q_100	11.	q_10	q_1
4.	$1q_00$	q_110	12.	q_11	q_1
5.	#	#	13.	$0q_11$	q_1
6.	0	0	14.	$1q_10$	q_1
7.	1	1	15.	$0q_10$	q_1
8.	$q_1\#\#$	#	16.	$1q_10$	q_1

Mapping ULP to MPCP

- We summarize the mapping as follows. Given T and w, there are five types of strings in list A and B:
- Starting string (first pair):

List A	List B
#	$\#q_0w\#$

where q_0 is the starting state of T.

Mapping ULP to MPCP

- Strings from the transition function δ:

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

where Z is any tape symbol except the blank.

Mapping ULP to MPCP

- Strings for copying:

List A	List B
X	X

where X is any tape symbol (including the blank).

Mapping ULP to MPCP

- Strings for consuming the tape symbols at the end:

List A	List B
Xq	q
qY	q
XqY	q

where q is an accepting state, and each X and Y is any tape symbol except the blank.

Mapping ULP to MPCP

- Ending string:

List A	List B
q##	#

  where q is an accepting state.

- Using this mapping, we can prove that the original ULP instance has a solution if and only if the mapped MPCP instance has a solution. (Textbook, p.402, Theorem 9.19)