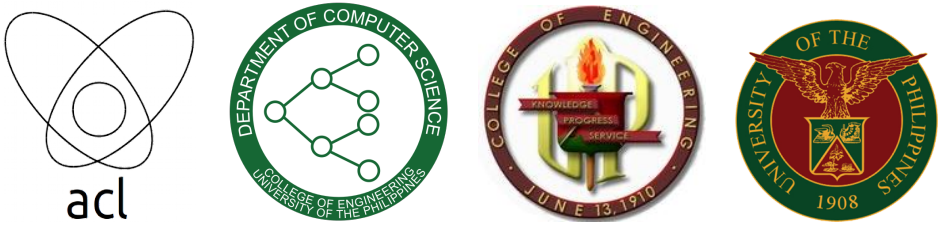
CS 133 : Automata Theory and Computability

Lecture Slides

Turing Machines and Undecidability

Francis George C. Cabarle
fccabarle@up.edu.ph



Algorithms and Complexity Laboratory
Department of Computer Science
College of Engineering
University of the Philippines Diliman

Day 23

Recalling undecidability

Reduction

Recalling undecidability

Reduction

Theorem: $L_d \notin RE$.

Theorem: $L_d \notin RE$.

Proof: Suppose we list all strings in lexicographic order, where w_i is the ith word and M_j is the TM whose encoding in binary is the integer j.

Theorem: $L_d \notin RE$.

Proof: Suppose we list all strings in lexicographic order, where w_i is the ith word and M_j is the TM whose encoding in binary is the integer j.
We can construct an infinite table where i and j label rows and columns, resp., and table entry (i, j) in the table is 0 if $w_i \notin L(M_j)$ and 1 otherwise.

Theorem: $L_d \notin RE$.

Proof: Suppose we list all strings in lexicographic order, where w_i is the ith word and M_j is the TM whose encoding in binary is the integer j.
We can construct an infinite table where i and j label rows and columns, resp., and table entry (i, j) in the table is 0 if $w_i \notin L(M_j)$ and 1 otherwise.
$L_d = \{w_i \mid w_i \notin L(M_j)\}$. Suppose M_j exists s.t. $L(M_j) = L_d$.

Theorem: $L_d \notin RE$.

Proof: Suppose we list all strings in lexicographic order, where w_i is the ith word and M_j is the TM whose encoding in binary is the integer j.
We can construct an infinite table where i and j label rows and columns, resp., and table entry (i, j) in the table is 0 if $w_i \notin L(M_j)$ and 1 otherwise.
$L_d = \{w_i \mid w_i \notin L(M_j)\}$. Suppose M_j exists s.t. $L(M_j) = L_d$.
Only two possible cases exist: either $w_j \in L_d$ or $w_j \notin L_d$

Theorem: $L_d \notin RE$.

Proof: Suppose we list all strings in lexicographic order, where w_i is the ith word and M_j is the TM whose encoding in binary is the integer j.
We can construct an infinite table where i and j label rows and columns, resp., and table entry (i, j) in the table is 0 if $w_i \notin L(M_j)$ and 1 otherwise.
$L_d = \{w_i \mid w_i \notin L(M_j)\}$. Suppose M_j exists s.t. $L(M_j) = L_d$.
Only two possible cases exist: either $w_j \in L_d$ or $w_j \notin L_d$
Case 1: if $w_j \in L_d$, entry (j, j) is 0, then $w_j \notin L(M_j)$ and contradicting $L_d = L(M_j)$.

Theorem: $L_d \notin RE$.

Proof: Suppose we list all strings in lexicographic order, where w_i is the ith word and M_j is the TM whose encoding in binary is the integer j.
We can construct an infinite table where i and j label rows and columns, resp., and table entry (i, j) in the table is 0 if $w_i \notin L(M_j)$ and 1 otherwise.
$L_d = \{w_i \mid w_i \notin L(M_j)\}$. Suppose M_j exists s.t. $L(M_j) = L_d$.
Only two possible cases exist: either $w_j \in L_d$ or $w_j \notin L_d$
Case 1: if $w_j \in L_d$, entry (j, j) is 0, then $w_j \notin L(M_j)$ and contradicting $L_d = L(M_j)$.
Case 2: if $w_j \notin L_d$, entry (j, j) is 1, then $w_j \in L(M_j)$ and contradicting $L_d = L(M_j)$.

Theorem: $L_d \notin RE$.

Proof: Suppose we list all strings in lexicographic order, where w_i is the ith word and M_j is the TM whose encoding in binary is the integer j.
We can construct an infinite table where i and j label rows and columns, resp., and table entry (i, j) in the table is 0 if $w_i \notin L(M_j)$ and 1 otherwise.
$L_d = \{w_i \mid w_i \notin L(M_j)\}$. Suppose M_j exists s.t. $L(M_j) = L_d$.
Only two possible cases exist: either $w_j \in L_d$ or $w_j \notin L_d$
Case 1: if $w_j \in L_d$, entry (j, j) is 0, then $w_j \notin L(M_j)$ and contradicting $L_d = L(M_j)$.
Case 2: if $w_j \notin L_d$, entry (j, j) is 1, then $w_j \in L(M_j)$ and contradicting $L_d = L(M_j)$.
Either case allows a contradiction, ergo, assumption $L_d = L(M_j)$ is false, i.e. M_j cannot exist.

More on undecidability

Theorem: $L_u \in RE$ and $L_u \notin REC$.

[1] L_u or the *universal language* due to the use of a UTM.

More on undecidability

Theorem: $L_u \in RE$ and $L_u \notin REC$.

Proof: $L_u = \{\langle M, w\rangle \mid M \text{ accepts } w\}$.

[1] L_u or the *universal language* due to the use of a UTM.

More on undecidability

Theorem: $L_u \in RE$ and $L_u \notin REC$.

Proof: $L_u = \{\langle M, w\rangle \mid M \text{ accepts } w\}$.
Easy to show L_u is RE, i.e. we can construct[1] a TM (e.g. a UTM) that simulates M on input w.

[1] L_u or the *universal language* due to the use of a UTM.

More on undecidability

Theorem: $L_u \in RE$ and $L_u \notin REC$.

Proof: $L_u = \{\langle M, w\rangle \mid M \text{ accepts } w\}$.
Easy to show L_u is RE, i.e. we can construct[1] a TM (e.g. a UTM) that simulates M on input w.
We need to show $L_u \notin REC$, by reducing the task of deciding L_d to deciding $\overline{L}_u$.

[1] L_u or the *universal language* due to the use of a UTM.

More on undecidability

Theorem: $L_u \in RE$ and $L_u \notin REC$.

Proof: $L_u = \{\langle M, w\rangle \mid M \text{ accepts } w\}$.
Easy to show L_u is RE, i.e. we can construct[1] a TM (e.g. a UTM) that simulates M on input w.
We need to show $L_u \notin REC$, by reducing the task of deciding L_d to deciding $\overline{L}_u$.
Suppose a TM A decides $\overline{L}_u$, i.e. $\overline{L}_u \in REC$, then $L_u \in REC$ also.

[1] L_u or the *universal language* due to the use of a UTM.

More on undecidability

Theorem: $L_u \in RE$ and $L_u \notin REC$.

Proof: $L_u = \{\langle M, w\rangle \mid M \text{ accepts } w\}$.
Easy to show L_u is RE, i.e. we can construct[1] a TM (e.g. a UTM) that simulates M on input w.
We need to show $L_u \notin REC$, by reducing the task of deciding L_d to deciding $\overline{L}_u$.
Suppose a TM A decides $\overline{L}_u$, i.e. $\overline{L}_u \in REC$, then $L_u \in REC$ also.
Convert w to its encoding in binary, represented by integer i, so $w = w_i$. Let $w_i = M_i$.

[1] L_u or the *universal language* due to the use of a UTM.

More on undecidability

Theorem: $L_u \in RE$ and $L_u \notin REC$.

Proof: $L_u = \{\langle M, w\rangle \mid M \text{ accepts } w\}$.
Easy to show L_u is RE, i.e. we can construct[1] a TM (e.g. a UTM) that simulates M on input w.
We need to show $L_u \notin REC$, by reducing the task of deciding L_d to deciding $\overline{L}_u$.
Suppose a TM A decides $\overline{L}_u$, i.e. $\overline{L}_u \in REC$, then $L_u \in REC$ also.
Convert w to its encoding in binary, represented by integer i, so $w = w_i$. Let $w_i = M_i$.
Use $\langle M_i, w_i\rangle$ as input to A: A accepts w iff M_i accepts w_i.

[1] L_u or the *universal language* due to the use of a UTM.

More on undecidability

Theorem: $L_u \in RE$ and $L_u \notin REC$.

Proof: $L_u = \{\langle M, w\rangle \mid M \text{ accepts } w\}$.
Easy to show L_u is RE, i.e. we can construct[1] a TM (e.g. a UTM) that simulates M on input w.
We need to show $L_u \notin REC$, by reducing the task of deciding L_d to deciding $\overline{L}_u$.
Suppose a TM A decides $\overline{L}_u$, i.e. $\overline{L}_u \in REC$, then $L_u \in REC$ also.
Convert w to its encoding in binary, represented by integer i, so $w = w_i$. Let $w_i = M_i$.
Use $\langle M_i, w_i\rangle$ as input to A: A accepts w iff M_i accepts w_i.
It is easy to see that A accepts w iff $w = w_i$ and $w_i \in L(M_i)$.

[1] L_u or the *universal language* due to the use of a UTM.

More on undecidability

Theorem: $L_u \in RE$ and $L_u \notin REC$.

Proof: $L_u = \{\langle M, w\rangle \mid M \text{ accepts } w\}$.
Easy to show L_u is RE, i.e. we can construct[1] a TM (e.g. a UTM) that simulates M on input w.
We need to show $L_u \notin REC$, by reducing the task of deciding L_d to deciding $\overline{L}_u$.
Suppose a TM A decides $\overline{L}_u$, i.e. $\overline{L}_u \in REC$, then $L_u \in REC$ also.
Convert w to its encoding in binary, represented by integer i, so $w = w_i$. Let $w_i = M_i$.
Use $\langle M_i, w_i\rangle$ as input to A: A accepts w iff M_i accepts w_i.
It is easy to see that A accepts w iff $w = w_i$ and $w_i \in L(M_i)$.
Thus, we have a TM for L_d! But such TM cannot exist, so our assumption that A exists is false.

[1] L_u or the *universal language* due to the use of a UTM.

More on undecidability

Theorem: $L_u \in RE$ and $L_u \notin REC$.

Proof: $L_u = \{\langle M, w\rangle \mid M \text{ accepts } w\}$.
Easy to show L_u is RE, i.e. we can construct[1] a TM (e.g. a UTM) that simulates M on input w.
We need to show $L_u \notin REC$, by reducing the task of deciding L_d to deciding $\overline{L}_u$.
Suppose a TM A decides $\overline{L}_u$, i.e. $\overline{L}_u \in REC$, then $L_u \in REC$ also.
Convert w to its encoding in binary, represented by integer i, so $w = w_i$. Let $w_i = M_i$.
Use $\langle M_i, w_i\rangle$ as input to A: A accepts w iff M_i accepts w_i.
It is easy to see that A accepts w iff $w = w_i$ and $w_i \in L(M_i)$.
Thus, we have a TM for L_d! But such TM cannot exist, so our assumption that A exists is false.
Ergo, L_u is RE but not REC.

A popular variant of L_u is the **Halting problem**.

[1] L_u or the *universal language* due to the use of a UTM.

More on undecidability

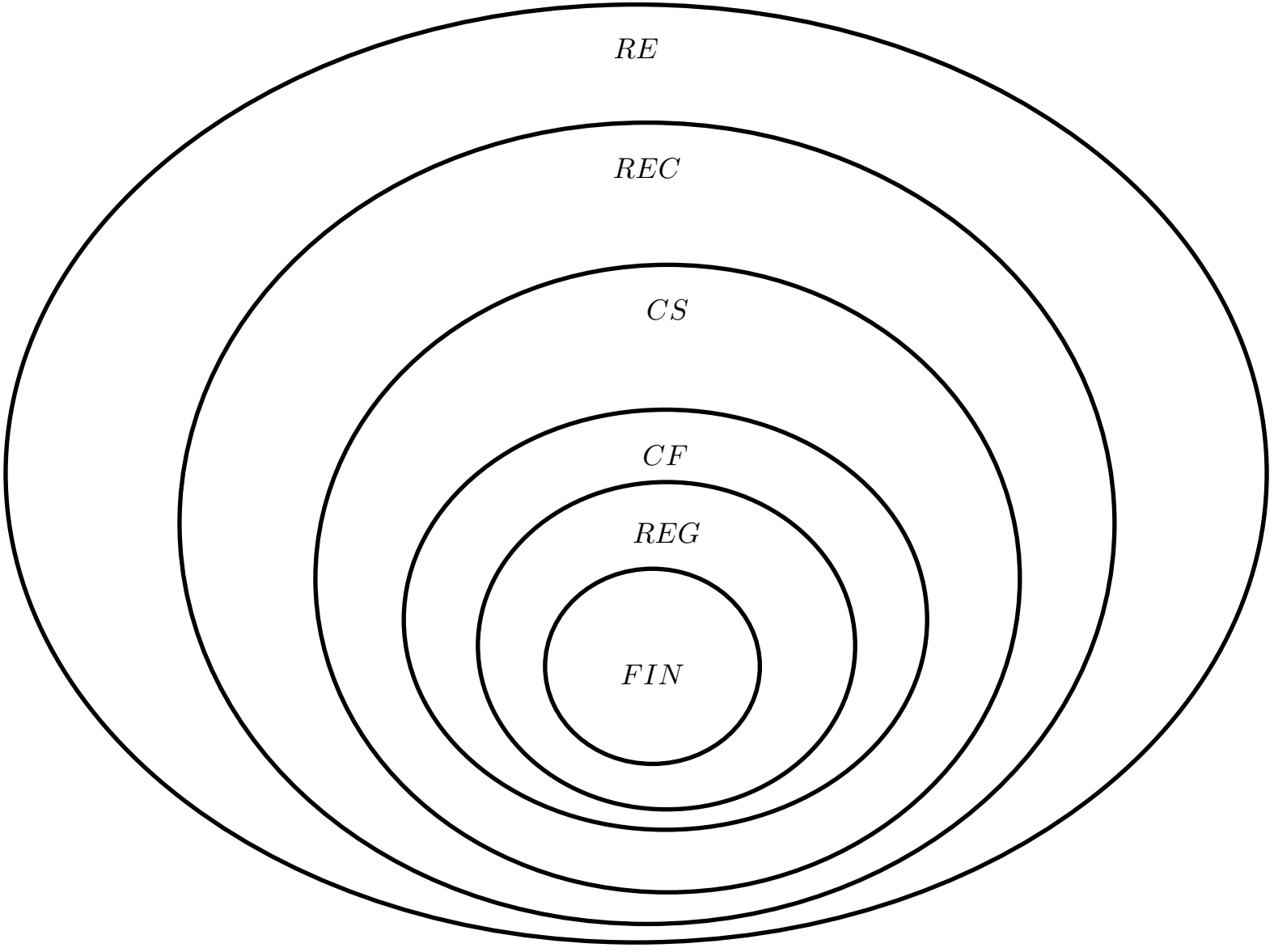
Theorem: $L_u \in RE$ and $L_u \notin REC$.

Proof: $L_u = \{\langle M, w\rangle \mid M \text{ accepts } w\}$.
Easy to show L_u is RE, i.e. we can construct[1] a TM (e.g. a UTM) that simulates M on input w.
We need to show $L_u \notin REC$, by reducing the task of deciding L_d to deciding $\overline{L}_u$.
Suppose a TM A decides $\overline{L}_u$, i.e. $\overline{L}_u \in REC$, then $L_u \in REC$ also.
Convert w to its encoding in binary, represented by integer i, so $w = w_i$. Let $w_i = M_i$.
Use $\langle M_i, w_i\rangle$ as input to A: A accepts w iff M_i accepts w_i.
It is easy to see that A accepts w iff $w = w_i$ and $w_i \in L(M_i)$.
Thus, we have a TM for L_d! But such TM cannot exist, so our assumption that A exists is false.
Ergo, L_u is RE but not REC.

A popular variant of L_u is the **Halting problem**. Also, since L_u is not decidable then so is software verification *in general*!

[1] L_u or the *universal language* due to the use of a UTM.

Chomsky hierarchy: complete!



Problem or language?

A *problem* is really just a *language*:

Problem or language?

A *problem* is really just a *language*:
Problem of deciding whether a program (i.e. TM or algorithm) with given input prints `Hello world!`, we can mean *strings* made of a C program and any input file(s) for the program.
Set of strings is a language over the alphabet of ASCII characters.

Problem or language?

A *problem* is really just a *language*:
Problem of deciding whether a program (i.e. TM or algorithm) with given input prints `Hello world!`, we can mean *strings* made of a C program and any input file(s) for the program.
Set of strings is a language over the alphabet of ASCII characters.

What about "more general" problems, e.g. transducers or computing functions?

Problem or language?

A *problem* is really just a *language*:
Problem of deciding whether a program (i.e. TM or algorithm) with given input prints `Hello world!`, we can mean *strings* made of a C program and any input file(s) for the program.
Set of strings is a language over the alphabet of ASCII characters.

What about "more general" problems, e.g. transducers or computing functions?
Convert the following problems to their language counterparts: Given $a, b \in \mathbb{N}$, output $c = a + b$; Given $\Sigma = \{x, y, z\}$ and $w \in \Sigma^*$, output $w' = x^{|w|_x} y^{|w|_y} z^{|w|_z}$.

Recalling undecidability

Reduction

Reduction

Reducing one problem to another: convert problem P_1 to problem P_2, so solving P_2 can be used to solve P_1.

Reduction

Reducing one problem to another: convert problem P_1 to problem P_2, so solving P_2 can be used to solve P_1.

e.g. Problem of travelling from Manila to Tokyo

Reduction

Reducing one problem to another: convert problem P_1 to problem P_2, so solving P_2 can be used to solve P_1.

e.g. Problem of travelling from Manila to Tokyo reduces to buying a plane ticket between the two cities

Reduction

Reducing one problem to another: convert problem P_1 to problem P_2, so solving P_2 can be used to solve P_1.

e.g. Problem of travelling from Manila to Tokyo reduces to buying a plane ticket between the two cities which reduces to earning money for the ticket

Reduction

Reducing one problem to another: convert problem P_1 to problem P_2, so solving P_2 can be used to solve P_1.

e.g. Problem of travelling from Manila to Tokyo reduces to buying a plane ticket between the two cities which reduces to earning money for the ticket which reduces to finding a job.

Reduction

Reducing one problem to another: convert problem P_1 to problem P_2, so solving P_2 can be used to solve P_1.

e.g. Problem of travelling from Manila to Tokyo reduces to buying a plane ticket between the two cities which reduces to earning money for the ticket which reduces to finding a job.

A *technique for proving undecidability*: if P_1 is known to be undecidable

Reduction

Reducing one problem to another: convert problem P_1 to problem P_2, so solving P_2 can be used to solve P_1.

e.g. Problem of travelling from Manila to Tokyo reduces to buying a plane ticket between the two cities which reduces to earning money for the ticket which reduces to finding a job.

A *technique for proving undecidability*: if P_1 is known to be undecidable to show a new problem P_2 is undecidable

Reduction

Reducing one problem to another: convert problem P_1 to problem P_2, so solving P_2 can be used to solve P_1.

e.g. Problem of travelling from Manila to Tokyo reduces to buying a plane ticket between the two cities which reduces to earning money for the ticket which reduces to finding a job.

A *technique for proving undecidability*: if P_1 is known to be undecidable to show a new problem P_2 is undecidable we reduce P_1 to P_2

Reduction

Reducing one problem to another: convert problem P_1 to problem P_2, so solving P_2 can be used to solve P_1.

e.g. Problem of travelling from Manila to Tokyo reduces to buying a plane ticket between the two cities which reduces to earning money for the ticket which reduces to finding a job.

A *technique for proving undecidability*: if P_1 is known to be undecidable to show a new problem P_2 is undecidable we reduce P_1 to P_2 i.e. we prove implication "if P_1 is undecidable, then P_2 is undecidable"

Reduction

Reducing one problem to another: convert problem P_1 to problem P_2, so solving P_2 can be used to solve P_1.

e.g. Problem of travelling from Manila to Tokyo reduces to buying a plane ticket between the two cities which reduces to earning money for the ticket which reduces to finding a job.

A *technique for proving undecidability*: if P_1 is known to be undecidable to show a new problem P_2 is undecidable we reduce P_1 to P_2 i.e. we prove implication "if P_1 is undecidable, then P_2 is undecidable" (proving the converse is incorrect!).

More undecidable problems (aka undecidable problems for TMs)

$$L_{ne} = \{\langle M \rangle \mid L(M) \neq \emptyset\} \text{ and } \overline{L}_{ne} = L_e = \{\langle M \rangle \mid L(M) = \emptyset\}.$$

More undecidable problems (aka undecidable problems for TMs)

$$L_{ne} = \{\langle M \rangle \mid L(M) \neq \emptyset\} \text{ and } \overline{L}_{ne} = L_e = \{\langle M \rangle \mid L(M) = \emptyset\}.$$

Theorem: $L_{ne} \in RE$ and $L_{ne} \notin REC$

Theorem: $L_e \notin RE$.

More undecidable problems (aka undecidable problems for TMs)

$$L_{ne} = \{\langle M \rangle \mid L(M) \neq \emptyset\} \text{ and } \overline{L}_{ne} = L_e = \{\langle M \rangle \mid L(M) = \emptyset\}.$$

Theorem: $L_{ne} \in RE$ and $L_{ne} \notin REC$

Theorem: $L_e \notin RE$.

To prove $L_{ne} \in RE$, we simply construct a TM M for L_{ne}.

More undecidable problems (aka undecidable problems for TMs)

$$L_{ne} = \{\langle M \rangle \mid L(M) \neq \emptyset\} \text{ and } \overline{L}_{ne} = L_e = \{\langle M \rangle \mid L(M) = \emptyset\}.$$

Theorem: $L_{ne} \in RE$ and $L_{ne} \notin REC$

Theorem: $L_e \notin RE$.

To prove $L_{ne} \in RE$, we simply construct a TM M for L_{ne}.
Given input $\langle M_i \rangle$, M simulates M_i (like a universal TM) on some input x (**nondeterministically** chosen, for example): if M_i accepts at least one string, M will eventually accept.

More undecidable problems (aka undecidable problems for TMs)

$$L_{ne} = \{\langle M \rangle \mid L(M) \neq \emptyset\} \text{ and } \overline{L}_{ne} = L_e = \{\langle M \rangle \mid L(M) = \emptyset\}.$$

Theorem: $L_{ne} \in RE$ and $L_{ne} \notin REC$

Theorem: $L_e \notin RE$.

To prove $L_{ne} \in RE$, we simply construct a TM M for L_{ne}.
Given input $\langle M_i \rangle$, M simulates M_i (like a universal TM) on some input x (**nondeterministically** chosen, for example): if M_i accepts at least one string, M will eventually accept.
If $L(M) = \emptyset$, then no x leads to acceptance.

More undecidable problems (aka undecidable problems for TMs)

To prove $L_{ne} \notin REC$, reduce solving L_u to solving L_{ne}.

More undecidable problems (aka undecidable problems for TMs)

To prove $L_{ne} \notin REC$, reduce solving L_u to solving L_{ne}.
Assume TM M_{ne} is an algorithm for L_{ne}, i.e. $L_{ne} \in REC$, then we can also have an algorithm M_u for L_u as follows:

More undecidable problems (aka undecidable problems for TMs)

To prove $L_{ne} \notin REC$, reduce solving L_u to solving L_{ne}.
Assume TM M_{ne} is an algorithm for L_{ne}, i.e. $L_{ne} \in REC$, then we can also have an algorithm M_u for L_u as follows:

1. M_u constructs a TM M' given input $\langle M, w \rangle$ such that $L(M') \neq \emptyset$ iff M accepts w.

More undecidable problems (aka undecidable problems for TMs)

To prove $L_{ne} \notin REC$, reduce solving L_u to solving L_{ne}.
Assume TM M_{ne} is an algorithm for L_{ne}, i.e. $L_{ne} \in REC$, then we can also have an algorithm M_u for L_u as follows:

1. M_u constructs a TM M' given input $\langle M, w \rangle$ such that $L(M') \neq \emptyset$ iff M accepts w.
2. Use M_{ne} to decide whether or not $L(M') = \emptyset$: reject, if M does not accept w since $L(M') = \emptyset$;

More undecidable problems (aka undecidable problems for TMs)

To prove $L_{ne} \notin REC$, reduce solving L_u to solving L_{ne}.
Assume TM M_{ne} is an algorithm for L_{ne}, i.e. $L_{ne} \in REC$, then we can also have an algorithm M_u for L_u as follows:

1. M_u constructs a TM M' given input $\langle M, w \rangle$ such that $L(M') \neq \emptyset$ iff M accepts w.
2. Use M_{ne} to decide whether or not $L(M') = \emptyset$: reject, if M does not accept w since $L(M') = \emptyset$; otherwise, accept, since $L(M') \neq \emptyset$ and M accepts w.

More undecidable problems (aka undecidable problems for TMs)

To prove $L_{ne} \notin REC$, reduce solving L_u to solving L_{ne}.
Assume TM M_{ne} is an algorithm for L_{ne}, i.e. $L_{ne} \in REC$, then we can also have an algorithm M_u for L_u as follows:

1. M_u constructs a TM M' given input $\langle M, w \rangle$ such that $L(M') \neq \emptyset$ iff M accepts w.
2. Use M_{ne} to decide whether or not $L(M') = \emptyset$: reject, if M does not accept w since $L(M') = \emptyset$; otherwise, accept, since $L(M') \neq \emptyset$ and M accepts w.

Since we know that $L_u \notin REC$, i.e. an M_u for L_u cannot exist, we contradict the assumption that an M_{ne} for L_{ne} exists.

More undecidable problems (aka undecidable problems for TMs)

To prove $L_{ne} \notin REC$, reduce solving L_u to solving L_{ne}.
Assume TM M_{ne} is an algorithm for L_{ne}, i.e. $L_{ne} \in REC$, then we can also have an algorithm M_u for L_u as follows:

1. M_u constructs a TM M' given input $\langle M, w \rangle$ such that $L(M') \neq \emptyset$ iff M accepts w.
2. Use M_{ne} to decide whether or not $L(M') = \emptyset$: reject, if M does not accept w since $L(M') = \emptyset$; otherwise, accept, since $L(M') \neq \emptyset$ and M accepts w.

Since we know that $L_u \notin REC$, i.e. an M_u for L_u cannot exist, we contradict the assumption that an M_{ne} for L_{ne} exists.
Ergo, $L_{ne} \notin REC$.

How to prove $L_e \notin RE$?

More undecidable problems (aka undecidable problems for TMs)

To prove $L_{ne} \notin REC$, reduce solving L_u to solving L_{ne}.
Assume TM M_{ne} is an algorithm for L_{ne}, i.e. $L_{ne} \in REC$, then we can also have an algorithm M_u for L_u as follows:

1. M_u constructs a TM M' given input $\langle M, w \rangle$ such that $L(M') \neq \emptyset$ iff M accepts w.
2. Use M_{ne} to decide whether or not $L(M') = \emptyset$: reject, if M does not accept w since $L(M') = \emptyset$; otherwise, accept, since $L(M') \neq \emptyset$ and M accepts w.

Since we know that $L_u \notin REC$, i.e. an M_u for L_u cannot exist, we contradict the assumption that an M_{ne} for L_{ne} exists.
Ergo, $L_{ne} \notin REC$.

How to prove $L_e \notin RE$? **Homework!**

A simple undecidable problem

We give an example of an undecidable problem concerning simple manipulation of strings.

A simple undecidable problem

We give an example of an undecidable problem concerning simple manipulation of strings.
Post correspondence problem (PCP): determine whether a collection of dominos has a **match**.

A simple undecidable problem

We give an example of an undecidable problem concerning simple manipulation of strings.
Post correspondence problem (PCP): determine whether a collection of dominos has a **match**.
An instance of the PCP is a collection P of dominos:

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

A simple undecidable problem

We give an example of an undecidable problem concerning simple manipulation of strings.
Post correspondence problem (PCP): determine whether a collection of dominos has a **match**.
An instance of the PCP is a collection P of dominos:

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

A **match** is a sequence $i_1, i_2, \ldots, i_m$, where
$t_{i_1} t_{i_2} \cdots t_{i_m} = b_{i_1} b_{i_2} \cdots b_{i_m}$.

A simple undecidable problem

We give an example of an undecidable problem concerning simple manipulation of strings.
Post correspondence problem (PCP): determine whether a collection of dominos has a **match**.
An instance of the PCP is a collection P of dominos:

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

A **match** is a sequence $i_1, i_2, \ldots, i_m$, where
$t_{i_1} t_{i_2} \cdots t_{i_m} = b_{i_1} b_{i_2} \cdots b_{i_m}$.

Example

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

Let $PCP = \{\langle P \rangle \mid P \text{ is an instance of the PCP with a match}\}$.

Let $PCP = \{\langle P \rangle \mid P \text{ is an instance of the PCP with a match}\}$.

Theorem

PCP is undecidable.

Let $PCP = \{\langle P \rangle \mid P \text{ is an instance of the PCP with a match}\}$.

Theorem

PCP is undecidable.

Suppose TM R decides the PCP.
We construct from R the TM R' that decides L_u

Let $PCP = \{\langle P \rangle \mid P \text{ is an instance of the PCP with a match}\}$.

Theorem

PCP is undecidable.

Suppose TM R decides the PCP.
We construct from R the TM R' that decides L_u
i.e., we show that from any TM M and input w, we can construct an instance P where a *match corresponds* to M *accepting* w.

Let $PCP = \{\langle P \rangle \mid P \text{ is an instance of the PCP with a match}\}$.

Theorem

PCP is undecidable.

Suppose TM R decides the PCP.
We construct from R the TM R' that decides L_u
i.e., we show that from any TM M and input w, we can construct an instance P where a *match corresponds* to M *accepting* w.
If we could determine whether the instance has a match, we would be able to determine whether M accepts w.

Let $PCP = \{\langle P \rangle \mid P \text{ is an instance of the PCP with a match}\}$.

Theorem

PCP is undecidable.

Suppose TM R decides the PCP.
We construct from R the TM R' that decides L_u
i.e., we show that from any TM M and input w, we can construct an instance P where a *match corresponds* to M *accepting* w.
If we could determine whether the instance has a match, we would be able to determine whether M accepts w.

Before getting into the construction, for convenience, we make the modifications (resulting to the modified PCP or MPCP):

- we assume that M on w never attempts to move its head off the left-hand end of the tape
- if $w = \varepsilon$, we use the string $\sqcup$ in place of w
- we modify PCP to require that a match starts with the first domino, $\left[\frac{t_1}{b_1}\right]$

Let $M = (Q, \Sigma, \Gamma, \delta, q_1, q_{accept}, q_{reject})$.

Let $M = (Q, \Sigma, \Gamma, \delta, q_1, q_{accept}, q_{reject})$.

Part 1. Put $\left[\frac{\#}{\#q_1 w_1 w_2 \cdots w_n \#}\right]$

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

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

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

Part 5. Put $\left[\frac{\#}{\#}\right]$ and $\left[\frac{\#}{\sqcup\#}\right]$ into P'.

Part 6. For every $a \in \Gamma$, put $\left[\frac{aq_{accept}}{q_{accept}}\right]$ and $\left[\frac{q_{accept}a}{q_{accept}}\right]$ into P'.

Part 7. Finally, we add the domino $\left[\frac{q_{accept}\#\#}{\#}\right]$ into P'.

Fin (wakas)

Thanks for the attention.

Questions?

Reading assignment(s)

[Sipser 2005] Chapter 5.1, 5.2, or

[Hopcroft et al 2001] Chapter 8.1.3, 9.1, 9.2, 9.3, 9.4

References:

[Sipser 2005] M. Sipser. *Introduction to the Theory of Computation*: 2ed. PWS Publishing Company, 2005.

[Hopcroft, Ullman 1979] J. Hopcroft, J. Ullman. *Introduction to Automata Theory, Languages, and Computation*: Addison-Wesley, 1979.

[Hopcroft et al 2001] J. Hopcroft, R. Motwani, J. Ullman. *Introduction to Automata Theory, Languages, and Computation*: Addison-Wesley, 2001.