COSE215: Theory of Computation

Lecture 19 — Undecidability (1)

Hakjoo Oh
2015 Spring

Undecidable Problems

Decidable problems (=Languages) are those that can be solved (=accepted) by computers (=Turing machines).

Undecidable Problems

Decidable problems (=Languages) are those that can be solved (=accepted) by computers (=Turing machines).

- Recursive (= decidable)
- Recursively enumerable (=semi-decidable)

Undecidable Problems

Decidable problems (=Languages) are those that can be solved (=accepted) by computers (=Turing machines).

- Recursive (= decidable)
- Recursively enumerable (=semi-decidable)

Undecidable problems are those that cannot be solved by computers.

- Non-recursive (= undecidable)
- Non-recursively enumerable (=semi-undecidable)

Today

1. Define the class of recursively enumerable languages
2. Define the class of recursive languages
3. Define a non-recursively enumerable language L_d and prove it
4. Define a non-recursive language L_u and prove it

Decidable Problems

Definition

A language L is *recursively enumerable* (RE) if there exists a Turing machine that accepts it.

$$L \text{ is RE} \iff \exists M \in TM.\, \forall w \in L.\; q_0 w \vdash^* x_1 q_f x_2$$

Decidable Problems

Definition

A language L is *recursively enumerable* (RE) if there exists a Turing machine that accepts it.

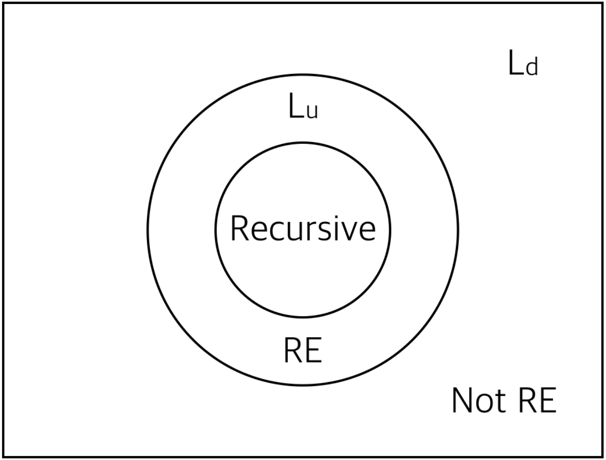
$$L \text{ is RE} \iff \exists M \in TM.\ \forall w \in L.\ q_0 w \vdash^* x_1 q_f x_2$$

Definition

A language L is *recursive* if there exists a Turing machine that accepts it and always terminates.

1. If w is in L, then M accepts
2. If w is not in L, then M eventually halts

Recursive / RE / Non-RE Languages



L_d: A language that is not recursively enumerable

We aim to define a language L_d that is not recursively enumerable:

$$L_d = \{w_i \mid w_i \notin L(M_i)\}$$

Representing Turing Machines as Binary Strings

$$M = (Q, \{0, 1\}, \Gamma, \delta, q_1, B, F)$$

1. $Q = \{q_1, q_2, \ldots, q_r\}$
2. $\Gamma = \{X_1, X_2, X_3, \ldots, X_s\}$

Representing Turing Machines as Binary Strings

$$M = (Q, \{0, 1\}, \Gamma, \delta, q_1, B, F)$$

1. $Q = \{q_1, q_2, \ldots, q_r\}$
2. $\Gamma = \{X_1, X_2, X_3, \ldots, X_s\}$

We encode the transition function

$$\delta(q_i, X_j) = (q_k, X_l, D)$$

by

- $0^i 1 0^j 1 0^k 1 0^l 1 0$ when $D = L$
- $0^i 1 0^j 1 0^k 1 0^l 1 00$ when $D = R$

Representing Turing Machines as Binary Strings

$$M = (Q, \{0,1\}, \Gamma, \delta, q_1, B, F)$$

1. $Q = \{q_1, q_2, \ldots, q_r\}$
2. $\Gamma = \{X_1, X_2, X_3, \ldots, X_s\}$

We encode the transition function

$$\delta(q_i, X_j) = (q_k, X_l, D)$$

by

- $0^i 1 0^j 1 0^k 1 0^l 1 0$ when $D = L$
- $0^i 1 0^j 1 0^k 1 0^l 1 00$ when $D = R$

The entire Turing machine is represented by

$$C_1 11 C_2 11 \cdots C_{n-1} 11 C_n$$

Example

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

$$\begin{array}{rcll}
\delta(q_1, 1) & = & (q_3, 0, R), & 0100100010100 \\
\delta(q_3, 0) & = & (q_1, 1, R), & 0001010100100 \\
\delta(q_3, 1) & = & (q_2, 0, R), & 00010010010100 \\
\delta(q_3, B) & = & (q_3, 1, L), & 0001000100010010
\end{array}$$

The entire Turing machine:

01001000101001100010101001001100010010010100110001000100

Binary strings can be ordered

$$
\begin{aligned}
w_1 &= \lambda \\
w_2 &= 0 \\
w_3 &= 1 \\
w_4 &= 00 \\
w_5 &= 01 \\
w_6 &= 10 \\
w_7 &= 11 \\
w_8 &= 000 \\
w_9 &= 001 \\
&\vdots
\end{aligned}
$$

Binary strings can be ordered

$$
\begin{aligned}
w_1 &= \lambda \\
w_2 &= 0 \\
w_3 &= 1 \\
w_4 &= 00 \\
w_5 &= 01 \\
w_6 &= 10 \\
w_7 &= 11 \\
w_8 &= 000 \\
w_9 &= 001 \\
&\vdots
\end{aligned}
$$

The order of binary string w is the integer value of $1w$

Turing machines can be ordered

M_i: The ith Turing machine

Definition

We define M_i to be the Turing machine whose binary representation is w_i.

Turing machines can be ordered

M_i: The ith Turing machine

Definition

We define M_i to be the Turing machine whose binary representation is w_i.

When M_i is not a valid Turing machine, define M_i to be a Turing machine with one state and no transitions, e.g., M_1.

The definition of L_d

Definition

$L_d = \{w_i \mid w_i \notin L(M_i)\}$

Theorem

L_d is not a recursively enumerable language.

Proof Sketch.

Suppose $L_d = L(M)$ for some TM M. Let k be the number of M, i.e., $M = M_k$.
Ask if w_k is in L_d.

- If $w_k \in L_d$, then M accepts w_k. But then, by definition of L_d, $w_i \notin L_d$. Contradiction.
- If $w_k \notin L_d$, then M does not accept w_k. But then, by definition of L_d, $w_k \in L_d$. Contradiction.

□

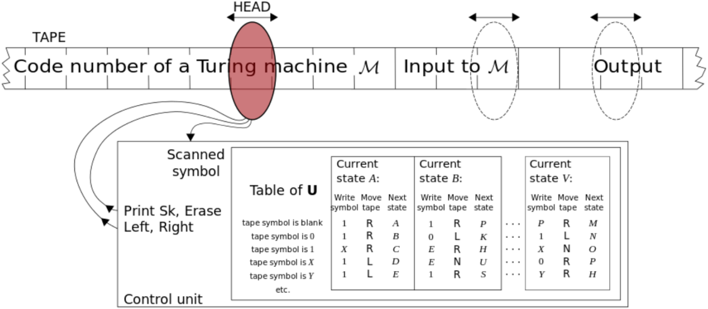
L_u: A language that is RE but not recursive

We define a language L_u that is not recursively enumerable:

$$L_u = \{(M, w) \mid w \in L(M)\}$$

L_u is recursively enumerable

The *universal Turing machine* accepts $L_u = \{(M, w) \mid w \in L(M)\}$.



A property of complements

Lemma

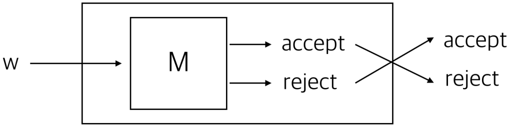
If L is a recursive language, then so is $\bar{L}$.

A property of complements

Lemma

If L is a recursive language, then so is $\bar{L}$.

Let $L = L(M)$ for some TM M that always halts. We construct a TM $\bar{M}$ such that $\bar{L} = L(\bar{M})$ as follows:



L_u is not recursive

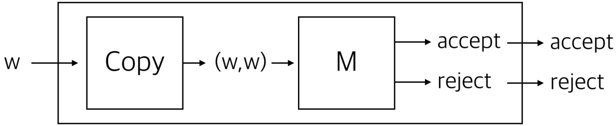
Theorem

L_u is RE but not recursive.

- Suppose L_u were recursive.
- Then by the property of complements, $\bar{L_u}$ is also recursive.
- However, if we have a TM M to accept $\bar{L_u}$, then we can construct a TM to accept L_d (explained next).
- We already know that L_d is not RE, contradiction.

Construction of TM to accept L_d from TM to accept $\bar{L_u}$

Suppose $L(M) = \bar{L_u}$. We construct M' s.t. $L(M') = L_d$ as follows:



Summary

- decidable / undecidable problems
- concrete examples of undecidable languages

Summary

- decidable / undecidable problems
- concrete examples of undecidable languages

Note that undecidable languages are different from intractable problems:

- undecidable problems: fundamentally unsolvable
- intractable problems: solvable but no efficient algorithms are known