Lec 19. Reduction and undecidable languages II

Eunjung Kim

Linear Bounded Automaton

Linear bounded Automaton

A linear bounded automaton is a Turing machine with the following restriction: its header is not allowed to move off the portion of the (single) tape containing the input. When the TM instructs the header to move to the right of the right-end of the input, then it stays where it is.

Key observation: For an input of length n, a linear bounded automaton on w can go through at most

$$|Q| \cdot n \cdot |\Gamma|^n$$

distinct configurations. This means

Halting problem for LBA

If a linear bounded automaton M on an input w of length n does not halt after the first $|Q| \cdot n \cdot |\Gamma|^n$ steps, then it never halts.

Linear Bounded Automaton

$HALT_{LBA} = \{\langle M, w\rangle : M \text{ is LBA and } M \text{ halts on } w\}$.
$A_{LBA} = \{\langle M, w\rangle : M \text{ is LBA and } M \text{ accepts } w\}$.

Decidability of some problems related to LBA

$HALT_{LBA}$ and A_{LBA} are decidable.

We can construct a TM D for $HALT_{LBA}$, which does the following on $\langle M, w\rangle$:

1. D simulates M on w for $|Q| \cdot n \cdot |\Gamma|^n$ steps.
2. If M halts, output Yes.
3. If M did not halt, output No.

Linear Bounded Automaton

$HALT_{LBA} = \{\langle M, w\rangle : M \text{ is LBA and } M \text{ halts on } w\}$.
$A_{LBA} = \{\langle M, w\rangle : M \text{ is LBA and } M \text{ accepts } w\}$.

Decidability of some problems related to LBA

$HALT_{LBA}$ and A_{LBA} are decidable.

We can construct a TM D for $HALT_{LBA}$, which does the following on $\langle M, w\rangle$:

1. D simulates M on w for $|Q| \cdot n \cdot |\Gamma|^n$ steps.
2. If M halts, output Yes.
3. If M did not halt, output No.

The construction of TM for A_{LBA} is similar. When exactly does this TM outputs No?

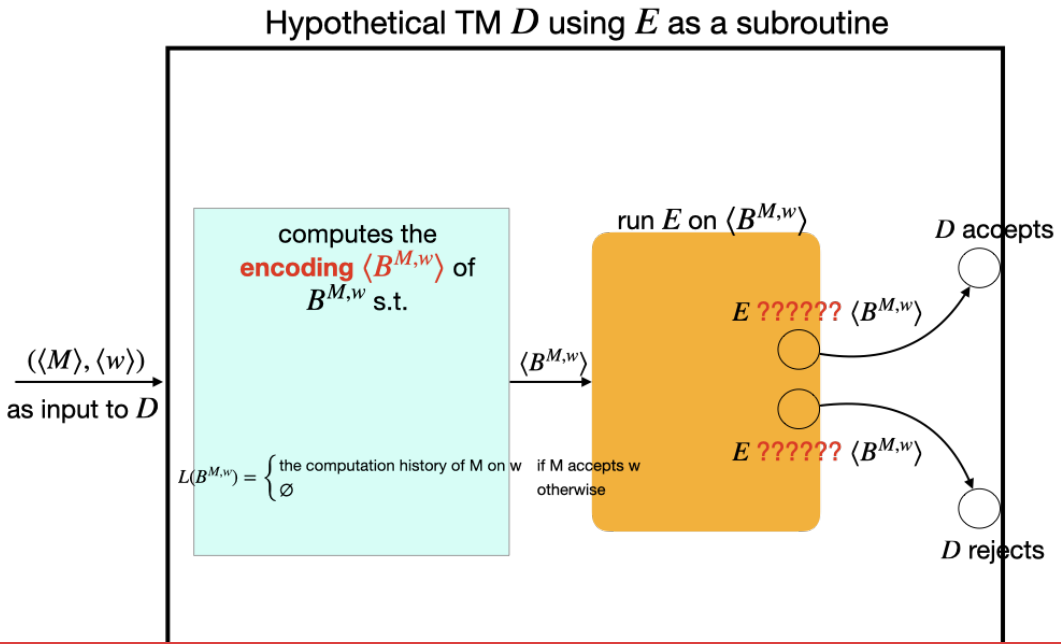
E_{LBA} IS UNDECIDABLE

$E_{LBA} = \{\langle M \rangle : M \text{ is LBA and } L(M) = \emptyset\}$.

UNDECIDABILITY OF E_{LBA}

E_{LBA} is undecidable.

Key idea: Reduce from A_{TM} to E_{LBA} by building a decider D for A_{TM} using a (hypothetical) decider E for E_{LBA}.



E_{LBA} IS UNDECIDABLE

- D upon an input $\langle M, w \rangle$ does the following:
  1. Compute & write an encoding $\langle B^{M,w} \rangle$ of LBA $B^{M,w}$ s.t.

$$L(B^{M,w}) = \begin{cases} \{\text{the accepting computation history of } M \text{ on } w\} & \text{if } M \text{ accepts } w \\ \emptyset \quad \text{if } M \text{ does not accept } w \end{cases}$$

  2. Run E on $\langle B^{M,w} \rangle$.
  3. D outputs

$$\begin{cases} \text{NO} & \text{if } E \text{ outputs YES} \\ \text{YES} & \text{if } E \text{ outputs NO} \end{cases}$$

E_{LBA} IS UNDECIDABLE

How does the LBA $B^{M,w}$ work internally?

$$L(B^{M,w}) = \begin{cases} \{\text{the accepting computation history of } M \text{ on } w\} & \text{if } M \text{ accepts } w \\ \emptyset \quad \text{if } M \text{ does not accept } w & \end{cases}$$

Upon an input string $x \in \Sigma^*$, we want:

- $B^{M,w}$ rejects x if it is not in the form

$$\#C_1\#C_2\#\cdots\#C_\ell\#$$

for some ℓ, where

- each C_i is a configuration of M; in the form $s\ q\ s'$ where q is the state of M and s, s' are strings over the alphabet of M,
- C_1 is a starting configuration of M on w, i.e. $q_{init}\ w$,
- C_ℓ is an accepting configuration of M, i.e. $y\ q_{accept}\ z$ for some $y, z \in \Gamma^*$.

- $B^{M,w}$ zig-zags between C_i and C_{i+1} and check $C_i \vdash_M C_{i+1}$. Reject if not.
- Accept the input x if nothing went wrong for all $i \leq \ell - 1$.

TM COMPUTING A FUNCTION

Let's use the writing power of TM to have more than 'yes'-'no' answers.

TM COMPUTING A FUNCTION IN GENERAL

Consider a single-tape TM $M = (Q, \Sigma, \delta, q_0, q_{final})$:

- the contents of the tape when M reaches q_{final} (halting/final state, and M terminates instantly) is said to be the output of M on w, written as $M(w)$.

We say that M computes a function $f : \Sigma^* \to \Sigma^*$ if for every input $w \in \Sigma^*$,

$$M(w) = f(w).$$

Especially, TM computing a function must halt on every input w.

TM COMPUTING A FUNCTION

Let's use the writing power of TM to have more than 'yes'-'no' answers.

TM COMPUTING A FUNCTION IN GENERAL

Consider a single-tape TM $M = (Q, \Sigma, \delta, q_0, q_{final})$:

- the contents of the tape when M reaches q_{final} (halting/final state, and M terminates instantly) is said to be the output of M on w, written as $M(w)$.

We say that M computes a function $f : \Sigma^* \to \Sigma^*$ if for every input $w \in \Sigma^*$,

$$M(w) = f(w).$$

Especially, TM computing a function must halt on every input w.

A function f is (Turing-)computable if there exists TM that computes f.

TM COMPUTING A FUNCTION

Let's use the writing power of TM to have more than 'yes'-'no' answers.

TM COMPUTING A FUNCTION IN GENERAL

Consider a single-tape TM $M = (Q, \Sigma, \delta, q_0, q_{final})$:

- the contents of the tape when M reaches q_{final} (halting/final state, and M terminates instantly) is said to be the output of M on w, written as $M(w)$.

We say that M computes a function $f : \Sigma^* \to \Sigma^*$ if for every input $w \in \Sigma^*$,

$$M(w) = f(w).$$

Especially, TM computing a function must halt on every input w.

A function f is (Turing-)computable if there exists TM that computes f.

When considering a multitape TM, one can designate a specific tape so that the output $M(w)$ of M is the content of the said tape.

TM COMPUTING A PARTIAL FUNCTION

TM COMPUTING A PARTIAL FUNCTION

Consider a TM $M = (Q, \Sigma, \delta, q_0, q_{final})$ as before.

We say that M computes a partial function $f : \Sigma^* \to \Sigma^*$ if for every input $w \in \Sigma^*$,

$$M(w) = f(w)$$

whenever $f(w)$ is defined and M does not halt if $f(w)$ is not defined.

Mapping-reducibility

Mapping-reducibility: definition

Let $A \subseteq \Sigma^*$ and $B \subseteq \Sigma^*$ be two languages.

We say that A is mapping-reducible (or many-one reducible) to B, written as $A \leq_m B$, if there is a computable function $f : \Sigma^* \to \Sigma^*$ such that for every input $w \in \Sigma^*$,

$$w \in A \text{ if and only if } f(w) \in B.$$

$A \leq_m B$ means B is as hard as A

Decidability propagates backwards

If $A \leq_m B$ and B is decidable, then A is decidable.

$A \leq_m B$ means B is as hard as A

Decidability propagates backwards

If $A \leq_m B$ and B is decidable, then A is decidable.

Proof: build a TM M_A which decides A, using the decider M_B for B and the TM R performing the many-one reduction; R halts on every input $x \in \Sigma^*$ and $R(x) \in B$ if and only if $x \in A$.

M_A upon an input string $w \in \Sigma^*$ does the following.

1. Run R on w and output $f(w)$.
2. Run M_B on $f(w)$: if M_B accepts $f(w)$, then M_A accepts. Otherwise, M_A rejects.

$A \leq_m B$ means B is as hard as A

Decidability propagates backwards

If $A \leq_m B$ and B is decidable, then A is decidable.

Proof: build a TM M_A which decides A, using the decider M_B for B and the TM R performing the many-one reduction; R halts on every input $x \in \Sigma^*$ and $R(x) \in B$ if and only if $x \in A$.

M_A upon an input string $w \in \Sigma^*$ does the following.

1. Run R on w and output $f(w)$.
2. Run M_B on $f(w)$: if M_B accepts $f(w)$, then M_A accepts. Otherwise, M_A rejects.

Undecidability propagates forwards

If $A \leq_m B$ and A is undecidable, then B is undecidable.

Basics about mapping-reducibility

Recognizability propagates backwards

If $A \leq_m B$ and B is Turing-recognizable, then A is recognizable.

Basics about mapping-reducibility

Recognizability propagates backwards

If $A \leq_m B$ and B is Turing-recognizable, then A is recognizable.

Unrecognizability propagates forwards

If $A \leq_m B$ and A is not Turing-recognizable, then B is not recognizable.

Basics about mapping-reducibility

Transitivity

If $A \leq_m B$ and $B \leq_m C$, then $A \leq_m C$.

Mapping-reducibility for complements

If $A \leq_m B$, then $\neg A \leq_m \neg B$.

Halting problem is undecidable

Halting problem: $HALT_{TM} = \{(M, w) : M \text{ is TM and } M \text{ halts on } w\}$.

$HALT_{TM}$ is undecidable via mapping-reducibility

$HALT_{TM}$ is undecidable.

Halting problem is undecidable

Halting problem: $HALT_{TM} = \{(M, w) : M \text{ is TM and } M \text{ halts on } w\}$.

$HALT_{TM}$ is undecidable via mapping-reducibility

$HALT_{TM}$ is undecidable.

Proof: We build TM R which transforms an input (M, w) to A_{TM} to an equivalent input (M', w') to $HALT_{TM}$.

Halting problem is undecidable

Halting problem: $HALT_{TM} = \{(M, w) : M \text{ is TM and } M \text{ halts on } w\}$.

$HALT_{TM}$ is undecidable via mapping-reducibility

$HALT_{TM}$ is undecidable.

Proof: We build TM R which transforms an input (M, w) to A_{TM} to an equivalent input (M', w') to $HALT_{TM}$.

That is, we want the output $R(\langle M \rangle, \langle w \rangle)$ of R on the input $(\langle M \rangle, \langle w \rangle)$ to be $(\langle M' \rangle, \langle w' \rangle)$ such that

- If M accepts w, then M' halts on w'.
- If M rejects or loops on w, then M' loops on w'.
- (If $\langle M \rangle$ is ill-formed (not an encoding of a TM), then M' is ill-formed too.)

Halting problem is undecidable

R works as follows on input (M, w):

1. R checks if $\langle M \rangle$ is well-formed; if not, output (M, w).
2. If $\langle M \rangle$ is well-formed, R internally writes a description of a new TM M' which, on any input string x,
   - simulates M on x,
   - if $M(x) = 1$, then M' halts on x, (so (M', x) is YES-instance to $HALT_{TM}$)
   - if $M(x) = 0$, then M' loops on x (so (M', x) is NO-instance to $HALT_{TM}$).
3. R outputs $\langle M', w \rangle$.

Remark: if M loops on x, then M' will loop on x anyway. But R does not simulate M' (only writes its encoding), so it doesn't matter for R to halt.

Post Correspondence Problem (PCP)

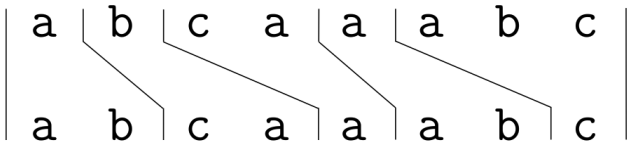
$$\left\{\left[\frac{\mathtt{b}}{\mathtt{ca}}\right], \left[\frac{\mathtt{a}}{\mathtt{ab}}\right], \left[\frac{\mathtt{ca}}{\mathtt{a}}\right], \left[\frac{\mathtt{abc}}{\mathtt{c}}\right]\right\}$$ Chapter 5.2, Sipser 2012.

Emil Post's Correspondence Problem

Input: a (finite) set $P = \{(\alpha_1, \beta_1), (\alpha_2, \beta_2), \ldots (\alpha_k, \beta_k)\}$ of ordered pairs (called dominoes) of strings over Σ.

Question: Is there a match, i.e. a sequence $i_1, \ldots, i_m \in [k]$ such that $\alpha_{i_1} \cdots \alpha_{i_m} = \beta_{i_1} \cdots \beta_{i_m}$?

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



Post Correspondence Problem

Key idea: many-one reduction (mapping-reduction).

- Many-one reduce from $A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM accepting } w\}$ to PCP.
- As an intermediary problem we introduce a decision problem called the Modified PCP (MPCP), in which an instance of PCP is a YES-instance iff there is a match which begins with the first domino (α_1, β_1).
- Combine two (many-one) reductions: from A_{TM} to MPCP, and one from MPCP to PCP.

Post Correspondence Problem

Set-up

1. We assume that the TM M of instance $\langle M, w \rangle$ satisfies:
   - it is deterministic, with left/right move only.
   - M never attempts to move the header to the left when it is in the left-most cell of the tape.
   - if $w = \epsilon$, the string w is encoded as B, where B is a symbol in the alphabet.
2. Reduction from MPCP to PCP is simple:

$$\left\{ \left[\frac{t_1}{b_1}\right], \left[\frac{t_2}{b_2}\right], \left[\frac{t_3}{b_3}\right], \cdots, \left[\frac{t_k}{b_k}\right] \right\}$$

$$\left\{ \left[\frac{\star t_1}{\star b_1 \star}\right], \left[\frac{\star t_1}{b_1 \star}\right], \left[\frac{\star t_2}{b_2 \star}\right], \left[\frac{\star t_3}{b_3 \star}\right], \cdots, \left[\frac{\star t_k}{b_k \star}\right], \left[\frac{* \diamond}{\diamond}\right] \right\}$$

Chapter 5.2, Sipser 2012.