Reducibility

The Halting Problem

Based on undecidability of one language, A, undecidability of another language, B, can be shown

The Halting Problem

Based on undecidability of one language, A, undecidability of another language, B, can be shown

We will use the concept of **reduction** for this purpose.

This is to show that one could build a Turing machine that decides A assuming that there were Turing machine for deciding B,

Assumption About Coding

The set of all possible inputs to the machine we will build as well as that to the machine we assume to exist (that is, Σ^* of the input alphabet Σ) may contain strings not encoding any meaningful objects.

For completeness our reduction has to handle such strings, but since the way we handle them is very simple (either accept all such strings or reject all such strings depending on how A or B is defined), we will simply ignore such strings.

The Halting Problem

$HALT_{\mathrm{TM}} = \{\langle M, w \rangle \mid M$ is a TM and halts on input $w\}$.

Theorem. $HALT_{\mathrm{TM}}$ **is undecidable.**

The Halting Problem

$HALT_{\mathrm{TM}} = \{\langle M, w\rangle \mid M$ is a TM and halts on input $w\}$.

Theorem. **$HALT_{\mathrm{TM}}$ is undecidable.**

Proof Assume that there is a Turing machine R that decides $HALT_{\mathrm{TM}}$.

We then would be able to construct a Turing machine S that decides

$$A_{\mathrm{TM}} = \{\langle M, w\rangle \mid M \text{ is a Turing machine and accepts } w\},$$

which is, however, known to be undecidable.

Logic Behind the Construction

- We want to know whether a Turing machine M on input w.
- A natural approach to finding an answer to that will be to simulate M on w, but the simulation may not stop.
  1. M **accepts** $w \rightarrow$ **"yes" ... ACCEPT**
  2. M **rejects** $w \rightarrow$ **"no" ... REJECT**
  3. M **on** w **never stops** $\rightarrow$ **"no" ... PROBLEM**
- If there is a TM machine that tells whether we will succumb to Case 3 or not, we can use that machine to avoid the problem.

Our Turing Machine S for A_{TM}

Let the input $x = \langle M, w \rangle$.

1. Simulate R on x.

   Note: R decides $HALT_{\mathrm{TM}}$ and so R on x halts.

2. If R rejects x, reject x.

   Note: This corresponds to Case 3 - the problematic case.

3. If R accepts x, simulate M on w, and accept if and only if M accepts.

   Note: Here we are distinguishing between Case 1 and Case 2.

This machine would correctly decide A_{TM}. ∎

Alternative Approach

- R is a Turing machine that purportedly decides the halting problem.
- Given a machine M we can modify its code to create a new machine M' such that M' enters a non-accepting infinite loop instead of rejecting. Then we have:
  1. M **accepts** w **...** M' **on** w **accepts.**
  2. M **rejects** w **...** M' **on** w **does not halt.**
  3. M **on** w **never stops ...** M' **on** w **does not halt.**
- Then M on w accepts if and only if M' on w halts.

Alternative Algorithm

1. From M construct a new Turing M' that simulates M and instead of entering q_{reject}, M' enters an infinite loop.
2. Simulate R on $\langle M', w \rangle$.
3. Accept if R accepts and reject otherwise.

The Emptiness Problem

Define $E_{\mathrm{TM}} = \{\langle M \rangle \mid M$ is a TM and $L(M) = \emptyset\}$.

Theorem. **E_{TM} is undecidable.**

The Emptiness Problem

Define $E_{\mathrm{TM}} = \{\langle M \rangle \mid M$ is a TM and $L(M) = \emptyset\}$.

Theorem. **E_{TM} is undecidable.**

Proof Assume there is a TM R that decides E_{TM}.
We'll construct a TM S that decides A_{TM}.

Algorithm of S for A_{TM}

1. Input x is $\langle M, w\rangle$ for some M and w.
   We want to know whether M on w accepts.
2. Construct a Turing machine M_1:
   M_1 erases its input y, reproduces w on input tape, and then enters simulation of M. That is, it behaves as M on input w regardless of input. We have:

$$L(M_1) = \begin{cases} \Sigma^* & \text{if } M \text{ accepts } w \\ \emptyset & \text{if } M \text{ does not accept } w \end{cases}$$

   Also, we have $\langle M_1\rangle \in E_{\mathrm{TM}}$ if and only if $L(M_1) = \emptyset$.
   R purportedly decides E_{TM}.
3. Simulate R on $\langle M_1\rangle$. Accept if R rejects and reject otherwise.

Testing Whether a TM Accepts a Regular Language

$REGULAR_{\mathrm{TM}} = \{\langle M \rangle \mid M$ is a TM and $L(M)$ is regular $\}$.

Theorem. $REGULAR_{\mathrm{TM}}$ **is undecidable.**

Testing Whether a TM Accepts a Regular Language

$REGULAR_{\mathrm{TM}} = \{\langle M \rangle \mid M$ is a TM and $L(M)$ is regular $\}$.

Theorem. $REGULAR_{\mathrm{TM}}$ **is undecidable.**

Proof Assume there is a TM R that decides $REGULAR_{\mathrm{TM}}$. We'll construct a TM S that decides A_{TM}.

Testing Whether a TM Accepts a Regular Language

Input $x = \langle M, w \rangle$.

1. Let Σ be the input alphabet of M. If Σ has only one symbol add another symbol.
2. Choose two symbols, say a and b, from Σ.
3. Construct a machine M_1 that on input y behaves as follow:
   (a) If $y = a^n b^n$ for some $n \geq 1$, accept.
   (b) Otherwise, erase y, reproduce w, simulate M on w.

   We have:

$$L(M_1) = \begin{cases} \Sigma^* & \textbf{if } M \textbf{ accepts } w \\ \{a^n b^n \mid n \geq 0\} & \textbf{if } M \textbf{ does not accept } w \end{cases}$$

   Also, Σ^* **is regular and** $\{a^n b^n \mid n \geq 0\}$ **is non-regular.**

4. Simulate R on $\langle M_1 \rangle$. Accept if and only if R accepts.

Testing Equivalence Between TMs

Define $EQ_{\rm TM} = \{\langle M_1, M_2\rangle \mid$ both M_1 and M_2 are TMs and $L(M_1) = L(M_2)\}$.

Theorem. **$EQ_{\rm TM}$ is undecidable.**

Testing Equivalence Between TMs

Define $EQ_{\mathrm{TM}} = \{\langle M_1, M_2\rangle \mid$ both M_1 and M_2 are TMs and $L(M_1) = L(M_2)\}$.

Theorem. **EQ_{TM} is undecidable.**

Proof Assume there is a TM R that decides EQ_{TM}. We'll construct a TM S that decides A_{TM}.

In the previous proof, in addition to M_1 construct M_2 that accepts Σ^*. Then we have

$$L(M_1) = L(M_2) \text{ if and only if } M \text{ accepts } w.$$

We simulate R on $\langle M_1, M_2\rangle$. Accept if R accepts and reject otherwise.

Linear Bounded Automata

A **linear bounded automaton** is a Turing machine wherein the head is not permitted to move beyond the region in which the input was written. If the head attempts to move beyond the region it is kept at the same position.

Linear Bounded Automata

A **linear bounded automaton** is a Turing machine wherein the head is not permitted to move beyond the region in which the input was written. If the head attempts to move beyond the region it is kept at the same position.

For example, the machine for deciding $\{0^n\#1^n\#2^n \mid n \geq 0\}$ can be made to be an LBA, by making it to mark each end of the input area.

Linear Bounded Automata

Lemma. **Let M be an LBA with q states and with a tape alphabet of size s. For every $n \geq 1$, for every input of length n, there are precisely qns^n possible configurations.**

The Acceptance Problem for LBA

$A_{\mathrm{LBA}} = \{\langle M, w\rangle \mid M$ is a TM and accepts w when restricted to be an LBA $\}$.

Theorem. **A_{LBA} is decidable.**

Proof Let M be a TM with q states and s symbols in the tape alphabet and let w be an input to M having length n. By the previous lemma, there are at most qns^n possible configurations that M might take on input w. Thus, if M on w accepts, it should do so within qns^n steps. This means that we have only to simulate M on w for at most qns^n steps to find out whether M accepts w or not. ∎

The Emptiness Problem About LBA

$E_{\text{LBA}} = \{\langle M \rangle \mid M$ is a TM and accepts no input when viewed as an LBA $\}$.

Theorem. E_{LBA} **is undecidable.**

The Emptiness Problem About LBA

$E_{\mathrm{LBA}} = \{\langle M \rangle \mid M$ is a TM and accepts no input viewed as an LBA $\}$.

Theorem. E_{LBA} **is undecidable.**

We use A_{TM} again. Given $x = \langle M, w \rangle$ whose membership in A_{TM} to be tested, we will construct a Truing machine S that tests whether a given series of configurations of M represents an accepting computation path of M on input w and show that this S can be made to be an LBA.

Assume there is a TM R that decides E_{LBA}.

Given $x = \langle M, w \rangle$, define L_x to be the set of all strings of the form $\#C_1\#C_2\#\cdots\#C_m\#$ such that

1. **$C_1, \ldots, C_m$ are configurations of M,**
2. **C_1 is the initial configuration** of M on w,
3. **C_m is an accepting configuration** of M on w, and
4. for every i, $1 \leq i \leq m-1$, **C_{i+1} is the next configuration of C_i.**

Then **$L_x \neq \emptyset$ if and only M accepts w.**

L_x can be decided by an LBA S.

We have only to test whether R accepts L_x.

The Equivalence Problem About CFG

Define $ALL_{\mathrm{CFG}} = \{\langle G \rangle \mid G$ is a CFG and $L(G) = \Sigma^*\}$.

The Equivalence Problem About CFG

Define $ALL_{\rm CFG} = \{\langle G \rangle \mid G$ is a CFG and $L(G) = \Sigma^*\}$.

Theorem. $ALL_{\rm CFG}$ **is undecidable.**

Proof For a sting $x = \langle M, w \rangle$ such that M is a Turing machine and w is an input to M, let L_x be the set of all $\#D_1\# \cdots \#D_m\#$ for which there exist $C_1, \ldots, C_m$ such that:

1. $C_1, \ldots, C_m$ **are configurations of** M**,**
2. C_1 **is the initial configuration** of M on w,
3. C_m **is an accepting configuration** of M on w,
4. for every i, $2 \leq i \leq m$, C_i **is the next configuration of** C_{i-1}, and
5. for every i, $1 \leq i \leq m$, $D_i = C_i$ **if** i **is odd and** $D_i = C_i^R$ **otherwise.**

Proof (cont'd)

Then L_x is empty if and only if M does not accept w,
and so
$\overline{L_x} = \Sigma^*$ if and only if M does not accept w.

$\overline{L_x}$ is a CFL.

Why Is $\overline{L_x}$ a CFL?

$\overline{L_x}$ consists of all words w for which at least one of the following properties holds:

(I) w does not start with a $\#$.

(II) w does not end with a $\#$.

(III) w contains as a substring $\#y\#$ such that y is free of $\#$ but is not a configuration.

(IV) w starts with $\#y\#$ such that y is free of $\#$ and y is not the initial configuration.

(V) w ends with $\#y\#$ such that y is free of $\#$ and y is not an accepting configuration or its reverse.

(Cont'd)

(VI) w contains a pattern $\#D_i\#D_{i+1}\#$ such that i is an odd number, $D_i = upa$, $\delta(p,a) = (q,b,R)$, and $D_{i+1} \neq (ubq\sqcup)^R$.

(VII) w contains a pattern $\#D_i\#D_{i+1}\#$ such that i is an odd number, $D_i = upav$, $|v| \geq 1$, $\delta(p,a) = (q,b,R)$, and $D_{i+1} \neq (ubqv)^R$.

(VIII) w contains a pattern $\#D_i\#D_{i+1}\#$ such that i is an odd number, $D_i = pav$, $\delta(p,a) = (q,b,L)$, and $D_{i+1} \neq (qbv)^R$. ($D_{i+1} \neq uqcbv$ in the case where $c \neq \epsilon$).

(IX) w contains a pattern $\#D_i\#D_{i+1}\#$ such that i is an odd number, $D_i = ucpav$, $\delta(p,a) = (q,b,L)$, $D_{i+1} \neq (uqcbv)^R$.

(X) The even-i versions of (VI) – (IX), where the D_i side is reversed instead.

Proof (cont'd)

Now given a TM R that decides ALL_{CFG} we will construct a TM S that decides A_{TM}

S's algorithm: on input $x = \langle M, w \rangle$,

1. Construct a CFG G for $\overline{L_x}$.
2. **Simulate R on $\langle G \rangle$.** Accept x if R accepts $\langle G \rangle$ and reject x otherwise. ∎

The Equivalence Problem

Define $EQ_{\text{CFG}} = \{\langle G, H\rangle \mid G$ and H are CFGs that generate the same language $\}$.

Corollary. **EQ_{CFG} is undecidable.**

The Equivalence Problem

Define $EQ_{\text{CFG}} = \{\langle G, H\rangle \mid G$ and H are CFGs that generate the same language $\}$.

Corollary. EQ_{CFG} **is undecidable.**

Proof From a TM R that decides EQ_{CFG} we can construct a TM S that decides ALL_{CFG}.

On input $x = \langle G\rangle$, S behaves as follows:

1. Let Σ be the terminals of G. Construct a grammar H that generates Σ^*.
2. **Simulate R on $\langle G, H\rangle$.** Accept x if R accepts and reject x otherwise. ∎