Reductions

Basic idea: we don't have to solve problems from scratch. Use existing problems to solve new problems.

Can be useful in practice (e.g. SAT solvers) but is also a way to show that new problems are e.g., undecidable

Say, e.g., we have a language L and we want to know whether it is decidable.

Suppose we can show that *if* we *could* decide L, *then* we *could* decide A_{TM}

We conclude that we *can't* decide L

How do we show *if* . . . *then*? Reductions!

$HALT_{TM}$

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

Theorem: $HALT_{TM}$ is undecidable.

IDEA: Use machine for $HALT_{TM}$ to solve A_{TM}.
Proof: Assume there is a TM R which decides $HALT_{TM}$. Construct the TM S to decide A_{TM} as follows:

On input $\langle M, w\rangle$, S does the following:

1. Modifiy M to get M' as follows: when M goes into a halting state that is not an *accept* state, M' goes into an *infinite loop*.
2. Run R on input $\langle M', w\rangle$ and accept iff R does

Notice that M accepts w iff M' halts on w. So S accepts $\langle M, w\rangle$ iff R accepts $\langle M', w\rangle$. So if R decides, $HALT_{TM}$, S decides A_{TM}

Test for Emptiness

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

Theorem: E_{TM} is undecidable.

Proof: For any M, w we can let M_1 be the TM which takes as input string x:

1. If $x \neq w$ M_1 rejects.
2. If $x = w$ M_1 runs M on input w and accepts if M does.

Now we construct TM S to decide $\overline{A_{TM}}$. Let R be a hypothetical TM which decides E_{TM}:

S has input $\langle M, w \rangle$

1. Use $\langle M, w \rangle$ to construct $\langle M_1 \rangle$ as described above.
2. Run R on $\langle M_1 \rangle$ and accept iff R accepts

If R decided the emptiness of $L(M_1)$, then S decides $\overline{A_{TM}}$. Therefore R can't exist and E_{TM} is undecidable.

Testing for Regularity

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

Theorem: $REGULAR_{TM}$ is undecidable.

Proof idea: We build a decider for A_{TM} which given $\langle M, w \rangle$ first constructs a machine M_2 which recognizes a non-regular language if M does not accept w and recognizes $\{0, 1\}^*$ (a regular langauge) if M accepts w.

Proof that $REGULAR_{TM}$ is Undecidable

Proof: Let R be a TM that decides $REGULAR_{TM}$. We construct S to decide A_{TM} as follows. S has input $\langle M, w \rangle$:

1. Construct M_2 such that on input x, M_2 does:
   (a) If x has the form $1^n 0^n$ accept.
   (b) If x has any other form, run M on w and accept x if M accepts w.
2. S runs R on $\langle M_2 \rangle$ and accepts iff R does

If M accepts w then M_2 accepts all strings, so $L(M_2)$ is regular, which implies R accepts, which implies S accepts. If M doesn't accept w, then M_2 only accepts strings in $0^n 1^n$, which implies $L(M_2)$ is not regular, which implies R rejects, which implies S rejects. So S decides A_{TM} which gives a contradiction. Hence $REGULAR_{TM}$ is undecidable.

Equivalence of TM's

$EQ_{TM} = \{\langle M_1, M_2\rangle \mid M_1, M_2$ are TM's and $L(M_1) = L(M_2)\}$.

Theorem: EQ_{TM} is undecidable.

IDEA: Can use such a TM to test if a language is empty.

Proof: Let R be the (hypothetical) TM which decides EQ_{TM}. Construct S to decide E_{TM} as follows: S is given $\langle M\rangle$ as input.

1. Run R on $\langle M, M'\rangle$, where M' is the TM which rejects all strings and accept iff R accpets

If $L(M)$ is empty, then R accepts and S accepts. If $L(M)$ is not empty then R rejects and S rejects. So S decides E_{TM}. But E_{TM} is undecidable so R cannot exist and EQ_{TM} is undecidable.

Mapping Reducibility

We don't want to do an *ad hoc* argument every time we get a new problem. We will formalize what we have been doing using *mapping reducibilities*.

A function $f : \Sigma^* \to \Sigma^*$ is a *computable function* if some TM on every input w halts with just $f(w)$ on its tape.

Examples: $f(w) = w^R$, modifying a description of a TM $\langle M \rangle$ in a simple way.

Formal Definition

A language A is *mapping reducible* to a language B (written $A \leq_m B$) if there is a computable function $f : \Sigma^* \to \Sigma^*$ where for every w

$$w \in A \text{ iff } f(w) \in B$$

Observation: $A \leq_m B$ iff $\overline{A} \leq_m \overline{B}$.

Proving Problems are Decidable

Theorem: If $A \leq_m B$ and B is decidable then A is decidable.

Proof: Let M be a TM which decides B. Construct a TM N which decides A using M as a subroutine. On input w, TM N does the following"

1. Compute $f(w)$

2. Run M on $f(w)$ and accept if M does. Otherwise reject.

Since M is a decider, N is also a decider. ■

Corollary: If $A \leq_m B$ and A is undecidable then B is undecidable.

Proving Turing-recognizability

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

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

Recall that $\overline{A_{TM}}$ is not Turing recognizable. We use this to show that EQ_{TM} is neither Turing recognizable nor co-Turing recognizable.

Unrecognizability of EQ_{TM}

Theorem EQ_{TM} is not Turing recognizable

Proof We show $A_{TM} \leq_m \overline{EQ_{TM}}$, which implies $\overline{A_{TM}} \leq_m EQ_{TM}$.

Define f as follows: $f(\langle M, w \rangle) = \langle M_1, M_2 \rangle$ where M_1, M_2 are machines such that

1. M_1 rejects all inputs;
2. For any input x, M_2 runs M on w and accepts if M accepts.

If $\langle M, w \rangle \in A_{TM}$ then M accepts w. But then $L(M_1) = \emptyset$ and $L(M_2) = \Sigma^*$, so $\langle M_1, M_2 \rangle \in \overline{EQ_{TM}}$.

If $\langle M, w \rangle \notin A_{TM}$ then $L(M_1) = L(M_2) = \emptyset$ and $\langle M_1, M_2 \rangle \notin \overline{EQ_{TM}}$.

So f is a mapping reduction from A_{TM} to $\overline{EQ_{TM}}$.

Unrecognizability of $\overline{EQ_{TM}}$

Theorem: EQ_{TM} is not co-Turing recognizable

Proof: To show $\overline{EQ_{TM}}$ is not Turing recognizable, we show a reduction from A_{TM} to EQ_{TM}. Define g as follows: $g(\langle M, w\rangle) = \langle M_1, M_2\rangle$ where M_1, M_2 are machines such that

1. M_1 accepts all inputs;
2. For any input x, M_2 runs M on w and accepts if M accepts.

Similar argument as before except M accepts w iff both M_1 and M_2 accept.

Rice's Theorem

A *property* of a set is a subset of the set. Some properties of the set of Turing-recognizable languages are: the property of being context-free, the property of being empty (this subset has only one element $\{\emptyset\}$).

A property is *trivial* if it is either empty (i.e., satisfied by no languages) or is all the Turing-recognizable languages.

Rice's Theorem: Every nontrivial property of the Turing-recognizable languages is undecidable.

That is, given a nontrivial property P, there is no TM to decide when given an encoding of a TM M, whether the language $L(M)$ has property P.

Proof of Rice's Theorem: Let P be any nontrivial property, and let L_P be the set of strings which encode all TM's M such that $L(M)$ has property P. There are two cases to consider, based on whether or not the empty language is in P.

Case 1: $\emptyset \notin P$

Let M_L be any TM whose encoding $\langle M_L \rangle$ is in L_P, i.e., $L(M_L)$ is in P. We show there is a reduction f from A_{TM} to L_P.

- $f(\langle M, w\rangle) = M'$ where M' is a machine which on input x works as follows:
  - It first simulates M on w.
  - If M accepts w, then M' goes on to simulate M_L on x. Otherwise M' halts without accepting.

  We then have

  - If $\langle M, w\rangle \in A_{TM}$, then M accepts w, so $L(M') = L(M_L)$, which means $\langle M' \rangle \in L_P$, i.e., $f(\langle M, w\rangle) \in L_P$.
  - If $\langle M, w\rangle \notin A_{TM}$, then M' doesn't doesn't accept any string, i.e., $L(M') = \emptyset$. Since $\emptyset \notin P$, $\langle M' \rangle \notin L_P$, i.e., $f(\langle M, w\rangle) \notin L_P$.

Case 2: $\emptyset \in P$

- If L_P is decidable, so is its complement,

$$\overline{L_P} = L_{\overline{P}} \cup \{\text{strings which don't encode TM's}\}$$

- Since {strings which don't encode TM's} is decidable, then $L_{\overline{P}}$ is decidable.
- But we can apply the same argument as we made above to show $L_{\overline{P}}$ is not decidable.
- This gives a contradiction.

Examples

Does M halt on the empty tape?

Is there any string which M halts on?

Given a TM and a state q, does the TM ever enter state q?

(Remaining slides are optional)

Computation History

A *computation history* for a TM on an input is the sequence of configurations that the machine goes through as it processes the input.

An *accepting computation history* for M on w is the sequence of configurations $C_1, C_2, \ldots, C_t$ where C_1 is a start configuration and each C_i follows according to the rules of M and C_t is an accepting configuration. *A rejecting computation history* is similar except C_t is a rejecting configuration.

Suppose M is a deterministic TM. Then $\langle M, w \rangle \in A_{TM}$ iff there is *exactly one* accepting computation history for M on w. We will try to use this to get some more undecidability results.

Linear Bounded Automaton

A *linear-bounded automaton* is a type of TM where the tape head isn't permitted to move off the portion of the tape containing the input. If the machine tries to move off, it stays put. The tape alphabet may be larger than the input alphabet, giving in effect a constant factor increase in tape size.

Theorem: $A_{LBA} = \{\langle M, w\rangle \mid M \text{ is an LBA that accepts string } w\}$ is decidable.

Lemma: Let M be an LBA with q states and g symbols in the tape alphabet. There are exactly qng^n distinct configurations of M for a tape of length n.

Proof of lemma: A configuration is determined by the state, tape head position, and tape contents. Number of states $= q$. Number of tape head positions $= n$. Number of possible tape contents $= g^n$.

Decidability of LBA Acceptance

Proof of theorem:
IDEA: If LBA M doesn't halt within qng^n steps then it repeats forever.
A TM S to decide A_{LBA} simulates M on w for qng^n steps or until it halts.

1. If M accepts, S accepts
2. If M rejects, S rejects
3. If M doesn't halt, S rejects.

Emptiness for LBA's is Undecidable

Theorem: $E_{LBA} = \{\langle M \rangle \mid M \text{ is an LBA where } L(M) = \emptyset\}$ is undecidable.

Proof Idea: We will show that $A_{TM} \leq_m \overline{E_{LBA}}$. Define a mapping reduction f such that $f(\langle M, w \rangle) = \langle B \rangle$, where B is a LBA that recognizes accepting computation histories of M on w.

Clearly, if M does not accept w, then $L(B) = \emptyset$. Otherwise, $L(B)$ contains one element (what is it?)

So $\langle M, w \rangle \in A_{TM}$ iff $\langle B \rangle \in \overline{E_{LBA}}$.

Defining B

B operates as follows:

It checks its input string to see if it is a sequence of configurations, with the start configuration equal to $q_0 w_1 w_2 \ldots w_n$ (w is hard-coded into B and the final configuration an accepting configuration (state is q_{accept}.)

It then checks back and forth (using marking) to see if each configuration follows from the next according to the transitions of M. That is, it checks that the tapes are the same except where the head is, the state change is according to δ of M. B accepts if the configuration history is according to δ and ends in an accepting state (δ is hard-coded into B.)

Since $\overline{E_{LBA}}$ is not decidable, neither is E_{LBA}.

Does a CFG Generate Σ^*?

Theorem: $ALL_{CFG} = \{\langle G\rangle \mid G \text{ is a CFG and } L(G) = \Sigma^*\}$ is undecidable.

Proof Idea: Show that $A_{TM} \leq_m \overline{ALL_{CFG}}$ by designing a mapping reduction $f(\langle M, w\rangle) = \langle D\rangle$, where D is a PDA accepting all strings *except* the accepting computation history of M on w

So M accepts w iff $L(D) \neq \Sigma^*$.

D *guesses* where its input fails to be an accepting computation history of M on w, uses stack to verify this – need to assume that every second configuration is written in reverse for this to work.

See textbook for details.

An Undecidable Problem that doesn't Involve Automata

Post Correspondence Problem (PCP)

Input – a set of *tiles* of the form $\left[\frac{u}{v}\right]$, were u, v are strings over Σ^*

Question: can we form a sequence $\left[\frac{u_1}{v_1}\right], \ldots, \left[\frac{u_k}{v_k}\right]$ of tiles from the set (with possible repetitions), such that $u_1 u_2 \ldots u_k = v_1 v_2 \ldots v_k$?

E.g., for the following set of tiles

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

we have the following sequence.

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

PCP is Undecidable

Show that $A_{TM} \leq_m PCP$ – given $\langle M, w\rangle$, tiles are used to encode configurations such that there is a match iff M accepts w.