Reduction

Video Lecture "Reductions and Undecidability", related practice problems and their solutions are on:
http://cse.iitkgp.ac.in/ abhij/course/theory/FLAT/Spring20/

1 Undecidability and Diagonalization

Given a Turing Machine $(Q, \Sigma, \Gamma, \vdash, B, \delta, s, t, r)$, on any input string x it can do one of the following:

- halt and accept x by entering accept state t,
- halt and reject x by entering reject state r,
- loop for infinite steps.

Next, recall the integer representation of Turing Machines. This was a way of representing a Turing machine by a string in $\{0,1\}^*$. Let us extend this definition to all strings in $\{0,1\}^*$ - those strings that are not in the form of an integer representation of a Turing machine are mapped to a fixed Turing machine, let us say a trivial TM M_0 with one state (accept state t) that immediately halts and accepts no matter what the input from $\{0,1\}^*$.

So now, for each string x in $\{0,1\}^*$, there is an associated TM M_x. Also, because of encoding a TM as a binary string, it can be part of the input of another TM!

Consider the Halting Problem: $HP = \{M\#x | M \text{ halts on } x\}$.

This is a subset of strings in $\{0,1,\#\}^*$. A Turing machine T accepting this language HP would do the following:

- Check if the substring before # is indeed a valid integer representation for a TM M, or assume that the substring represents the trivial TM $M = M_0$.
- Check whether the substring x after # is a substring over the alphabet of Σ of M. If $M = M_0$ then by definition, the alphabet for M is $\Sigma = \{0,1\}$.
- Simulate M on x: (i) If M halts and accepts x then so does T, (ii) if M halts and rejects x then so does T, (iii) if M loops on x then so does T.

Next, we ask whether there can be a Total TM accepting HP. In other words, is it possible that HP is a recursive set (the problem HP is decidable)? We can

show by a method called *diagonalization* that HP is in fact an undecidable language.

Exercise: Where else have you heard the term diagonalization? Hint: Mathematics.

Now, let us see the argument for why HP is undecidable. Suppose, for the sake of contradiction that there was a Total TM K accepting HP. Then K would be doing the following on an input $M\#x$:

- halt and accept if M halts on x,
- halt and reject if M loops on x.

Consider a TM N that on input $x \in \{0,1\}^*$ that does the following:

- constructs the integer representation of M_x for x (it is x if x is a valid integer representation, or it is the integer representation of M_0) and writes down $M_x\#x$ on its tape,
- simulates the total TM K on $M_x\#x$, accepting x if K rejects and going into a trivial loop if K accepts.

Then for any $x \in \{0,1\}^*$:
N halts on x
$\iff$ K rejects $M_x\#x$, by definition of N,
$\iff$ M_x loops on x, because K is a total TM for HP.

But N is a TM, so there is a x_0 for which $M_{x_0} = N$. Consider x_0:
N halts on x_0
$\iff$ K rejects $M_{x_0}\#x_0$
$\iff$ $M_{x_0} = N$ loops on x, which is a contradiction to how N behaves on x_0!

This the assumption that such a Total TM K exists for HP is wrong, and therefore HP is undecidable.

2 Examples of Undecidable languages

So far, we have seen several examples of problems that are not decidable, ie. languages that cannot be accepted by any total Turing machine.

The first example we saw was the Halting Problem.

$HP = \{(M,x) | M \text{ halts on } x\}$.

For this language, we used a diagonalization argument, similar to the Cantor diagonalization argument, to show that there can be no total Turing machine accepting the language HP.

Then, we considered the Membership problem.

$MP = \{(M,x) | M \text{ accepts } x\}$.

To show that this language is not recursive, we showed that if there was a total Turing Machine K for MP, then we could design a total Turing Machine T for HP that (i) took $M\#x$ as input, (ii) wrote down the description of a Turing machine N, whose description was based on the input machine M: It

accepts a string y if and only if M halts on y, (iii) Had the description of the total Turing machine K hardwired (this means that either there is an extra tape on the Turing machine permanently has the description of K written on it, or that the entire finite description of K is remembered using states of T); Depending on the answer given by K on the input $N\#x$, the machine T would be able to accept or reject the input $M\#x$ (T accepts $M\#x$ if and only if K accepts $N\#x$ if and only if N accepts x if and only if M halts on x).

This would imply that T is total, which is a contradiction to the fact that HP is not recursive. Thus, it must be the case that the assumption of K being a total Turing machine for MP is wrong. Therefore, MP is not recursive.

We saw several other examples of how a total Turing machine for any of the following languages would mean the existence of a total Turing machine for HP:

- $\{M|M$ accepts $\epsilon\}$
- $\{M|M$ accepts any string $\}$
- $\{M|M$ accepts all strings$\}$
- $\{M|M$ accepts a finite set $\}$

More undecidable problems/properties

Next, let us consider the following language:

$REC = \{M|M$ accepts a recursive set $\}$

We show that if there is a total Turing machine K for REC then we can design a total Turing machine for HP. This is how we do it. We know from the above that MP, although a recursively enumerable language, is not a recursive language. Let A be a Turing machine (not total!) for MP. Now we design a Turing machine T for HP, that uses K and A as subroutines:

1. T takes as input $M\#x$.
2. It writes down the description of a Turing machine N based on the input M and the Turing machine A for the language MP: for any input Y, N first simulates M on x.
   If M halts on x then N simulates A on y and accepts y if and only if A accepts y.
3. T simulates K on the input N and accepts $M\#x$ if and only if K rejects N.

Now let us look at what T is doing:

Firstly, the machine N will not accept any strings if M does not halt on x - this is because N will keep simulating M on x. Thus, if M does not halt on x, then $L(N) = \emptyset$.

Suppose M halts on x then N accepts y if and only if $y \in MP$. This is because once M halts on x, N runs the machine A on input y and accepts y if and only if A accepts y. This, when M halts on x then $L(N) = MP$.

Now, $\emptyset$ is trivially a recursive set, while MP is not a recursive set (we have already argued this before). Therefore,

T accepts $M\#x$ if and only if K rejects N

if and only if N is not a recursive language

if and only if $L(N) = MP$

if and only if M halts on x.

Thus, if K is a total Turing machine for the language REC, then we can design a total Turing machine for HP, which is a contradiction. Therefore, there can be no total Turing machine for REC, and REC is not recursive.

Simple Exercises: Are the following languages decidable:

1. $REG = \{M | M$ accepts a regular set $\}$
2. $CF = \{M | M$ accepts a context free grammar $\}$

3 Undecidability: Reductions

There is a formal name for the procedure we have described above: of designing a total Turing machine for a problem known to be undecidable in case a total Turing machine for a new problem exists, thereby concluding that the new problem must be undecidable as well. This tool is called a *reduction*.

Intuitively, given a problem Π known to be undecidable and a problem Π' whose decidability status is unknown, using this tool called reduction we will be manipulating instances of Π to make them instances of Π' in such a way that "yes" instances of (instances belonging to) Π become "yes" instances of Π' while "no" instances of (instances not belonging to) Π becomes "no" instances of Π'. Notice that if it was possible, using a total Turing machine, to differentiate between a "yes" instance and a "no" instance of Π' then this would also answer "yes" and "no" for instances of Π. This would lead to a contradiction to the fact that Π is undecidable. Therefore, no total Turing machine can exist for Π' and Π' is undecidable as well.

Let us formalize the definition of a reduction.

Given a set $A \in \Sigma^*$ and $B \in \Delta^*$ (the alphabets for the two sets can be different), a many-one reduction of A to B is a computable function

$$\sigma : \Sigma^* \to \Delta^*$$

such that for each $x \in \Sigma^*$,

$$x \in A \iff \sigma(x) \in B$$

.

Note that the function σ can be many-one, and need not be onto. The only condition on σ is that it is *computable by a total Turing machine*: that on any input x this total Turing machine halts with $\sigma(x)$ written on its tape.

When such a reduction exists, then we say that A is *reducible* to B via the map σ and we also use the notation $A \leq_m B$.

By definition, $\leq_m$ is transitive: if $A \leq_m B$ via a reduction σ_1 and $B \leq_m C$ via a reduction σ_2, then it must be that $A \leq_m C$ because of $\sigma_2 \cdot \sigma_1$ (Work out how this composite function will be computable by a total Turing machine if σ_1 and σ_2 are.)

Consequences of reductions

Theorem:

1. If $A \leq_m B$ and B is recursively enumerable then so is A. Equivalently, if $A \leq_m B$ and A is not recursively enumerable, then neither is B.

2. If $A \leq_m B$ and B is recursive then so is A. Equivalently, if A is not recursive then neither is B.

Proof: (i) Suppose $A \leq_m B$ via σ and B is recursively enumerable. This means that there is a Turing machine M for B. We design a Turing machine N for A as follows:

- For each input $x \in \Sigma^*$ ($A \subseteq \Sigma^*$), N computes $\sigma(x)$
- Then N simulates M on $\sigma(x)$.
- N accepts x if and only if M accepts $\sigma(x)$.

Thus, $L(N) = A$ if and only if $L(M) = B$.

(ii) Here, we have to be more careful. The Turing machine M for B is a total Turing machine and we need to design a total Turing machine N for A. Recall that recursive languages are by definition closed under complementation. Note that B is recursive if and only if B and $\overline{B}$ are recursively enumerable.

Notice that by definition of a reduction, both $A \leq_m B$ and $\overline{A} \leq_m \overline{B}$ via the same computable function σ. Now, we can use part (i) to show that if B is recursively enumerable then so is A and if $\overline{B}$ is recursively enumerable then so if $\overline{A}$. This implies that both A and $\overline{A}$ are recursively enumerable $\iff$ A is recursive.

Using reductions to show undecidability, non-semidecidability

We can use the theorem above to show that a given set is not recursive: Give a reduction from a set that is known to be non-recursive to the given set. Eg: HP, MP and all the problems we saw above.

We can also use the theorem above to show that a given set is not recursively enumerable: Give a reduction from a set that is known to be non-recursively

enumerable to the given set. Eg: $\overline{HP}$ (HP is r.e but not recursive. So it must be that $\overline{HP}$ is non-r.e), the complement of any problem that is semidecidable/r.e but not decidable/recursive.

Eg: Neither $FIN = \{M | L(M) \text{ is finite } \}$ nor $\overline{FIN}$ is recursively enumerable.

In order to show this, we give a reduction from $\overline{HP}$ to each of the problems. Note that $\overline{HP} = \{M\#x | M \text{ does not halt on } x\}$.

First, we describe a computable function σ such that $\overline{HP} \leq_m FIN$ via σ. In other words $M\#x \in \overline{HP} \iff \sigma(M\#x) \in FIN$.

Thus, from $M\#x$ we will construct a Turing machine $M' = \sigma(M\#x)$ such that M does not halt on x if and only if $L(M')$ is a finite set. Here is the description of M' based on M and x:

1. M' takes a y as input.
2. simulates M on x.
3. accepts y if M halts on x.

Note that:
M halts on $x \implies L(M') = \Sigma^*$ (infinite language)
M does not halt on $x \implies L(M') = \emptyset$ (finite set).

Thus, M does not halt on x if and only if $L(M')$ is a finite set and we have given a reduction from $\overline{HP}$ to FIN. This establishes the fact that FIN is not recursively enumerable.

Next, we describe a computable function σ such that $\overline{HP} \leq_m \overline{FIN}$ via σ. In other words $M\#x \in \overline{HP} \iff \sigma(M\#x) \in \overline{FIN}$. By definition of a reduction, this implies that $M\#x \in HP \iff \sigma(M\#x) \in FIN$ ("no" instances become "no" instances and "yes" instances become "yes" instances).

Thus, from $M\#x$ we will construct a Turing machine $M' = \sigma(M\#x)$ such that M halts on x if and only if $L(M')$ is a finite set. Here is the description of M' based on M and x:

1. M' takes a y as input and makes sure y is not overwritten on (Think how this is possible).
2. simulates M on x for $|y|$ steps (How can you keep this count?).
3. accepts y if M has not halted on x in $|y|$ steps and rejects otherwise.

Let us see what is the language accepted by M' based on whether M halts on x or not:

M halts on $x \implies L(M') = \{y | |y| <$ number of steps that M runs on x $\}$ (finite language as M halts on x in finite number of steps and all the y's accepted will have length less than these number of steps)

M does not halt on $x \implies L(M') = \Sigma^*$ (infinite set as no matter what the length of the input string y is, M would not have halted on x in $|y|$ steps).

Thus, M halts on x if and only if $L(M')$ is a finite set and we have given a reduction from HP to FIN, and established that $\overline{FIN}$ is not recursively enumerable.

The important thing to note in the constructions here, and in previous constructions is that if I give a reduction $A \leq_m B$ via σ:

(i) The instance constructed for set B only dependents on the computation of σ and on the input instance of A.

(ii) The function σ is not simulating any Turing machine - it is only writing down the description of a Turing machine in case the Turing machine is part of the input instance of B.