Many-one Reductions

Reductions

A *reduction* is a way of converting one problem into another problem such that a solution to the second problem can be used to solve the first problem. We say the first problem *reduces* to the second problem.

Examples

- The problem of "Is $w \in L_d$" reduces to the problem of "Given $\langle M \rangle \# w$, is $w \in L(M)$ or is $w \notin L(M)$".
- The problem of constructing a DFA for $L_1 \cup L_2$ can be reduced to the problem of constructing a DFA for L_1 and a DFA for L_2.

Precise definition of a reduction will be presented later.

The Halting Problem

Problem: Given M, w, decide if M halts on w. More formally,
$L_{\text{halt}} = \{\langle M \rangle \# w \mid M \text{ halts on } w\}$.

L_{halt} is clearly r.e. Because just run M on w and accept if M halts.

L_{halt} is undecidable!

So there is no algorithm to determine whether a program will get into an infinite loop!

Halting Problem is Undecidable

We will "reduce" L_u to L_{halt}.

Suppose L_{halt} is recursive. Let M_{halt} be the TM that always halts and which recognizes L_{halt}.

We will now construct a decision procedure for L_u. The algorithm for L_u is as follows:

- On input $\langle M \rangle \# w$, it simulates M_{halt} on $\langle M \rangle \# w$.
- If M_{halt} says "no" (meaning M does not halt on w) then the algorithm says "no".
- Otherwise, simulate M on w. If M says "yes" then output "yes", else output "no".

Checking Non-emptiness

Given i, is $L(M_i)$ non-empty? In other words, $L_{ne} = \{i \mid L(M_i) \neq \emptyset\}$

L_{ne} is r.e. Because there is a non-determinisitic algorithm which on input i, guesses a string w, and checks if $w \in L(M_i)$. If M_i accepts w, then M_i is non-empty.

Non-emptiness is Undecidable

Idea: Once again we will reduce L_u to L_{ne}. So the idea will be to show that if we have an algorithm to decide, on input i, if $L(M_i) \neq \emptyset$ then we can get an algorithm to decide given $\langle M \rangle \# w$ whether $w \in L(M)$.

Assume that M_{ne} halts on all inputs and recognizes L_{ne}.

We will describe a transformation which, given input $\langle M \rangle \# w$, will output the code of TM $T_{M,w}$ such that $L(T_{M,w}) \neq \emptyset$ iff $w \in L(M)$.

The decision procedure for L_u will, on input $\langle M \rangle \# w$, generate code $\langle T_{M,w} \rangle$, run M_{ne}, and accept iff M_{ne} accepts $\langle T_{M,w} \rangle$.

The Machine $T_{M,w}$

$T_{M,w}$ executes the following very simple routine:

On input x

Ignore x, and run M on w

If M accepts w, then accept x

Note: If M accepts w then $L(T_{M,w}) = \Sigma^*$. Otherwise $L(T_{M,w}) = \emptyset$.

Reducing L_u to L_{ne}

On input $\langle M \rangle \# w$, we need to output the code of $T_{M,w}$. How does the code of $T_{M,w}$ look?

Code for $T_{M,w}$ is similar to the code for M, except that it has some preprocessing code that writes w onto the tape.

- First write w onto the tape and return to the start state of M, with the head at the begining of the input.
- Execute M

$$111\langle \text{ Code to write } w \text{ and return to start state } \rangle 11\langle M \rangle 111$$

Some Consequences

Corollary 1: $L_e = \{i \mid L(M_i) = \emptyset\}$ is also undecidable.

- Because, $\overline{L_{ne}} = L_e$ and L is recursive iff $\overline{L}$ is recursive.

Corollary 2: L_e is not r.e.

- If L_e were r.e., then both L_e and L_{ne} will be r.e. This would imply that L_{ne} is recursive.

L_e is not R.E. (Direct Proof)

We will reduce L_d to L_e. In other words, we will show that if there is a TM which accepts i iff $L(M_i) = \emptyset$ then we can construct a TM which accepts i iff $i \notin L(M_i)$.

We describe a transformation which given i will output i' such that $i \in L(M_i)$ iff $L(M_{i'}) \neq \emptyset$.

So if L_e is recognized by a TM M_e, then the TM recognizing L_d does the following: On input i produces i', runs M_e on i' and accepts if M_e does.

Reducing L_d to L_e

On input i, i' is the code of the machine that does the following:

On input x

Run M_i on i for $|x|$ steps.

If M_i accepts i within $|x|$ steps then accept x, else reject x.

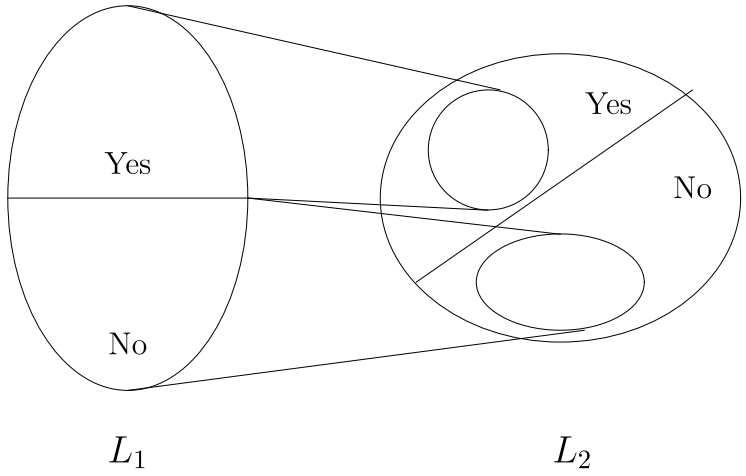
Proposition: $i \notin L(M_i)$ iff $L(M_{i'}) = \emptyset$. (exercise)

Many-one Reductions

Definition: A *many-one reduction* from L_1 to L_2 is a **computable** function $f : \Sigma^* \to \Sigma^*$ such that

$$w \in L_1 \Leftrightarrow f(w) \in L_2$$

The language L_1 is said to be *many-one reducible* to language L_2, and it is denoted by $L_1 \leq_m L_2$.



Many-one Reductions (contd)

Note 1: A function f is computable if there is a TM M with an output tape, such that M on input x halts after writing $f(x)$ on the output tape.

Note 2: Based on the class of languages we are studying, we will find it useful to refine this definition, by imposing addition constraints on the function f, like f should be computed *within some resource bounds.*

Note 3: There are more general classes of reductions, where solving L_1 involves "subroutine" calls to solving instances of L_2. Example, reducing regularity of $L_1 \cup L_2$ to the regularity of L_1 and L_2. We will not look at such reductions in depth.

Properties of Many-one Reductions

Proposition: If $L_1 \leq_m L_2$ and L_2 is recursively enumerable then L_1 is recursively enumerable.

- Let M_2 be the TM recognizing L_2. The the TM M_1 for L_1 does the following: On input w, it computes $f(w)$, runs M_2 on $f(w)$, and accepts if M_2 does.

Corollary 1: If $L_1 \leq_m L_2$ and L_1 is not recursively enumerable then L_2 is not recursively enumerable.

Corollary 2: If $L_1 \leq_m L_2$ and L_2 is recursive then L_1 is recursive.

- Observe that if f is a many-one reduction between from L_1 to L_2, then f is *also* a many-one reduction from $\overline{L_1}$ to $\overline{L_2}$.
- L_2 recursive means L_2 and $\overline{L_2}$ are r.e., and so L_1 and $\overline{L_1}$ are r.e. Hence L_1 is recursive/decidable.

Corollary 3: If $L_1 \leq_m L_2$ and L_1 is undecidable then L_2 is undecidable.

Checking Properties

Given i

- Does $L(M_i)$ contain i?
- Is $L(M_i)$ non-empty?
- Is $L(M_i)$ empty?

} Undecidable

- Is $L(M_i)$ infinite?
- Is $L(M_i)$ finite?
- Is $L(M_i)$ co-finite (i.e., is $\overline{L(M_i)}$ finite)?
- Is $L(M_i) = \Sigma^*$?

} Undecidable

Which of these properties can be checked? None! (Rice's Theorem)