Lecture 33

Reduction

There are two main techniques for showing that problems are undecidable: *diagonalization* and *reduction*. We saw examples of diagonalization in Lecture 31 and reduction in Lecture 32.

Once we have established that a problem such as HP is undecidable, we can show that another problem B is undecidable by *reducing* HP to B. Intuitively, this means we can manipulate instances of HP to make them look like instances of the problem B in such a way that "yes" instances of HP become "yes" instances of B and "no" instances of HP become "no" instances of B. Although we cannot tell effectively whether a given instance of HP is a "yes" instance, the manipulation preserves "yes"-ness and "no"-ness. If there existed a decision procedure for B, then we could apply it to the disguised instances of HP to decide membership in HP. In other words, combining a decision procedure for B with the manipulation procedure would give a decision procedure for HP. Since we have already shown that no such decision procedure for HP can exist, we can conclude that no decision procedure for B can exist.

We can give an abstract definition of reduction and prove a general theorem that will save us a lot of work in undecidability proofs from now on.

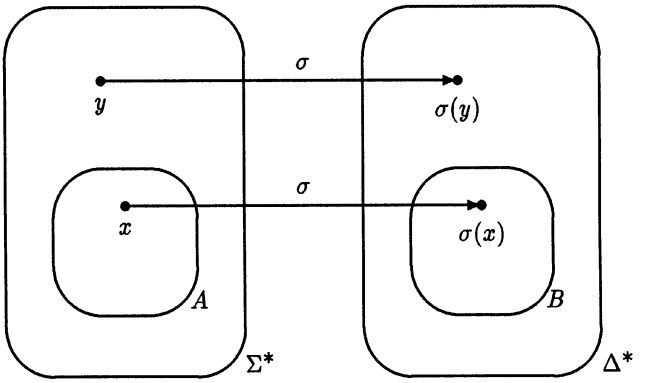
Given sets $A \subseteq \Sigma^*$ and $B \subseteq \Delta^*$, a (many-one) *reduction* of A to B is a computable function

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

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

$$x \in A \iff \sigma(x) \in B. \tag{33.1}$$

In other words, strings in A must go to strings in B under σ, and strings not in A must go to strings not in B under σ.



The function σ need not be one-to-one or onto. It must, however, be *total* and *effectively computable.* This means σ must be computable by a total Turing machine that on any input x halts with $\sigma(x)$ written on its tape. When such a reduction exists, we say that A is *reducible* to B via the map σ, and we write $A \leq_{\mathrm{m}} B$. The subscript m, which stands for "many-one," is used to distinguish this relation from other types of reducibility relations.

The relation $\leq_{\mathrm{m}}$ of reducibility between languages is transitive: if $A \leq_{\mathrm{m}} B$ and $B \leq_{\mathrm{m}} C$, then $A \leq_{\mathrm{m}} C$. This is because if σ reduces A to B and τ reduces B to C, then $\tau \circ \sigma$, the composition of σ and τ, is computable and reduces A to C.

Although we have not mentioned it explicitly, we have used reductions in the last few lectures to show that various problems are undecidable.

Example 33.1 In showing that it is undecidable whether a given TM accepts the null string, we constructed from a given TM M and string x a TM M' that accepted the null string iff M halts on x. In this example,

$$A = \{M\#x \mid M \text{ halts on } x\} = \mathrm{HP},$$
$$B = \{M \mid \epsilon \in L(M)\},$$

and σ is the computable map $M\#x \mapsto M'$. □

Example 33.2 In showing that it is undecidable whether a given TM accepts a regular set, we constructed from a given TM M and string x a TM M'' such that

$L(M'')$ is a nonregular set if M halts on x and $\varnothing$ otherwise. In this example,

$$A = \{M\#x \mid M \text{ halts on } x\} = \text{HP},$$
$$B = \{M \mid L(M) \text{ is regular}\},$$

and σ is the computable map $M\#x \mapsto M''$. □

Here is a general theorem that will save us some work.

Theorem 33.3 *(i) If $A \leq_m B$ and B is r.e., then so is A. Equivalently, if $A \leq_m B$ and A is not r.e., then neither is B.*

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

Proof. (i) Suppose $A \leq_m B$ via the map σ and B is r.e. Let M be a TM such that $B = L(M)$. Build a machine N for A as follows: on input x, first compute $\sigma(x)$, then run M on input $\sigma(x)$, accepting if M accepts. Then

$$\begin{aligned} N \text{ accepts } x &\iff M \text{ accepts } \sigma(x) && \text{definition of } N \\ &\iff \sigma(x) \in B && \text{definition of } M \\ &\iff x \in A && \text{by (33.1).} \end{aligned}$$

(ii) Recall from Lecture 29 that a set is recursive iff both it and its complement are r.e. Suppose $A \leq_m B$ via the map σ and B is recursive. Note that $\sim A \leq_m \sim B$ via the same σ (Check the definition!). If B is recursive, then both B and $\sim B$ are r.e. By (i), both A and $\sim A$ are r.e., thus A is recursive. □

We can use Theorem 33.3(i) to show that certain sets are not r.e. and Theorem 33.3(ii) to show that certain sets are not recursive. To show that a set B is not r.e., we need only give a reduction from a set A we already know is not r.e. (such as $\sim$HP) to B. By Theorem 33.3(i), B cannot be r.e.

Example 33.4 Let's illustrate by showing that neither the set

$$\text{FIN} = \{M \mid L(M) \text{ is finite}\}$$

nor its complement is r.e. We show that neither of these sets is r.e. by reducing $\sim$HP to each of them, where

$$\sim\text{HP} = \{M\#x \mid M \text{ does not halt on } x\}:$$

(a) $\sim\text{HP} \leq_m \text{FIN}$,

(b) $\sim\text{HP} \leq_m \sim\text{FIN}$.

Since we already know that $\sim$HP is not r.e., it follows from Theorem 33.3(i) that neither FIN nor $\sim$FIN is r.e.

For (a), we want to give a computable map σ such that

$$M\#x \in \sim\text{HP} \iff \sigma(M\#x) \in \text{FIN}.$$

In other words, from $M\#x$ we want to construct a Turing machine $M' = \sigma(M\#x)$ such that

$$M \text{ does not halt on } x \iff L(M') \text{ is finite.} \tag{33.2}$$

Note that the description of M' can depend on M and x. In particular, M' can have a description of M and the string x hard-wired in its finite control if desired.

We have actually already given a construction satisfying (33.2). Given $M\#x$, construct M' such that on all inputs y, M' takes the following actions:

(i) erases its input y;

(ii) writes x on its tape (M' has x hard-wired in its finite control);

(iii) runs M on input x (M' also has a description of M hard-wired in its finite control);

(iv) accepts if M halts on x.

If M does not halt on input x, then the simulation in step (iii) never halts, and M' never reaches step (iv). In this case M' does not accept its input y. This happens the same way for all inputs y, therefore in this case, $L(M) = \varnothing$. On the other hand, if M does halt on x, then the simulation in step (iii) halts, and y is accepted in step (iv). Moreover, this is true for all y. In this case, $L(M) = \Sigma^*$. Thus

$$\begin{array}{rcccl} M \text{ halts on } x & \Rightarrow & L(M') = \Sigma^* & \Rightarrow & L(M') \text{ is infinite,} \\ M \text{ does not halt on } x & \Rightarrow & L(M') = \varnothing & \Rightarrow & L(M') \text{ is finite.} \end{array}$$

Thus (33.2) is satisfied. Note that this is all we have to do to show that FIN is not r.e.: we have given the reduction (a), so by Theorem 33.3(i) we are done.

There is a common pitfall here that we should be careful to avoid. It is important to observe that the computable map σ that produces a description of M' from M and x does not need to execute the program (i) through (iv). It only produces the description of a machine M' that does so. The computation of σ is quite simple—it does not involve the simulation of any other machines or anything complicated at all. It merely takes a description of a Turing machine M and string x and plugs them into a general description of a machine that executes (i) through (iv). This can be done quite easily by a total TM, so σ is total and effectively computable.

Now (b). By definition of reduction, a map reducing $\sim$HP to $\sim$FIN also reduces HP to FIN, so it suffices to give a computable map τ such that

$$M\#x \in \text{HP} \iff \tau(M\#x) \in \text{FIN}.$$

In other words, from M and x we want to construct a Turing machine $M'' = \tau(M\#x)$ such that

$$M \text{ halts on } x \iff L(M'') \text{ is finite.} \tag{33.3}$$

Given $M\#x$, construct a machine M'' that on input y

(i) saves y on a separate track;

(ii) writes x on the tape;

(iii) simulates M on x for $|y|$ steps (it erases one symbol of y for each step of M on x that it simulates);

(iv) accepts if M has *not* halted within that time, otherwise rejects.

Now if M never halts on x, then M'' halts and accepts y in step (iv) after $|y|$ steps of the simulation, and this is true for all y. In this case $L(M'') = \Sigma^*$. On the other hand, if M does halt on x, then it does so after some finite number of steps, say n. Then M'' accepts y in (iv) if $|y| < n$ (since the simulation in (iii) has not finished by $|y|$ steps) and rejects y in (iv) if $|y| \geq n$ (since the simulation in (iii) does have time to complete). In this case M'' accepts all strings of length less than n and rejects all strings of length n or greater, so $L(M'')$ is a finite set. Thus

$$\begin{aligned} M \text{ halts on } x &\Rightarrow L(M'') = \{y \mid |y| < \text{running time of } M \text{ on } x\} \\ &\Rightarrow L(M'') \text{ is finite,} \end{aligned}$$

$$\begin{aligned} M \text{ does not halt on } x &\Rightarrow L(M'') = \Sigma^* \\ &\Rightarrow L(M'') \text{ is infinite.} \end{aligned}$$

Then (33.3) is satisfied.

It is important that the functions σ and τ in these two reductions can be computed by Turing machines that always halt. □

Historical Notes

The technique of diagonalization was first used by Cantor [16] to show that there were fewer real algebraic numbers than real numbers.

Universal Turing machines and the application of Cantor's diagonalization technique to prove the undecidability of the halting problem appear in Turing's original paper [120].

Reducibility relations are discussed by Post [101]; see [106, 116].