CS605
26 March 2025

Defn
A language is co-T-R if it is the complement of a T-R language.

THEOREM 4.22

A language is decidable iff it is Turing-recognizable and co-Turing-recognizable.

Proof
We will prove both directions to prove this theorem.

First we prove that if a lang is dec. then it is T-r and co-T-r. The former is true by definition. The latter is true for the following reason. If we take a TM that decides a lang and modify it so when it rejects, we accept, then this machine recognises the complement of the language proving the original lang. is co-T-R.

Secondly, we prove that if a lang. is T-r and co-T-r then it is decidable.

Let A be such a language. Let M_1 be the recogniser for A. Let M_2 be the recogniser for $\bar{A}$. Construct M as follows.

M = "On input w:
1. Run M_1 on w and M_2 on w, in parallel.
2. If M_1 accepts, accept, if M_2 accepts, reject."

M is guaranteed to halt. M accepts words w in A and rejects words w not in A. So M decides A.
This proves that A is decidable.
Our proof was completely general and works for any A that is T-r and co-T-r.

Combining these two proofs, proves the "if and only if" property of this theorem.

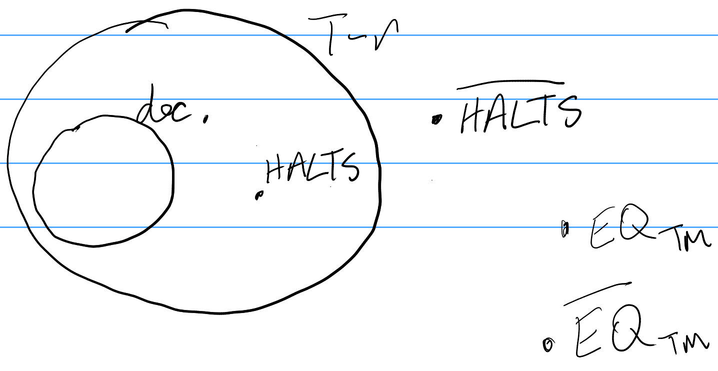
Thm HALTS is T-r but not co-T-r
(i.e. $\overline{HALTS}$ is outside T-r).

Proof

The proof that HALTS is T-r was done earlier.

recap: N = "On input $\langle M, w \rangle$:
1. Run M on w.
2. Accept." N is a recognizer for HALTS

The proof that $\overline{HALTS}$ is not T-r (equivalently that HALTS is not co-T-r) is as follows.
Let's assume, to create a contradiction, that $\overline{HALTS}$ is T-r. Now, according to Thm 4.22 this means that HALTS is decidable. A contradiction, because we have already proved HALTS is undecidable.
Therefore $\overline{HALTS}$ must not be T-r (equivalently HALTS must not be co-T-r.



Mapping reductions

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

For example, suppose that you want to find your way around a new city. You know that doing so would be easy if you had a map. Thus you can reduce the problem of finding your way around the city to the problem of obtaining a map of the city.

The following are further examples of reducibilities. The problem of traveling from Boston to Paris reduces to the problem of buying a plane ticket between the two cities. That problem in turn reduces to the problem of earning the money for the ticket. And that problem reduces to the problem of finding a job.

Reducibility also occurs in mathematical problems. For example, the problem of measuring the area of a rectangle reduces to the problem of measuring its length and width. The problem of solving a system of linear equations reduces to the problem of inverting a matrix.

Reducibility plays an important role in classifying problems by decidability and later in complexity theory as well. When A is reducible to B, solving A cannot be harder than solving B because a solution to B gives a solution to A. In terms of computability theory, if A is reducible to B and B is decidable, A also is decidable. Equivalently, if A is undecidable and reducible to B, B is undecidable. This last version is key to proving that various problems are undecidable.

In short, our method for proving that a problem is undecidable will be to show that some other problem already known to be undecidable reduces to it.

5.3

MAPPING REDUCIBILITY

Roughly speaking, being able to reduce problem A to problem B by using a mapping reducibility means that a computable function exists that converts instances of problem A to instances of problem B. If we have such a conversion function, called a *reduction*, we can solve A with a solver for B. The reason is that any instance of A can be solved by first using the reduction to convert it to an instance of B and then applying the solver for B. A precise definition of mapping reducibility follows shortly.

DEFINITION 5.17

A function $f\colon \Sigma^* \longrightarrow \Sigma^*$ is a ***computable function*** if some Turing machine M, on every input w, halts with just $f(w)$ on its tape.

FORMAL DEFINITION OF MAPPING REDUCIBILITY

Now we define mapping reducibility. As usual we represent computational problems by languages.

DEFINITION 5.20

Language A is ***mapping reducible*** to language B, written $A \leq_{\mathrm{m}} B$, if there is a computable function $f\colon \Sigma^* \longrightarrow \Sigma^*$, where for every w,

$$w \in A \iff f(w) \in B.$$

The function f is called the ***reduction*** of A to B.

The following figure illustrates mapping reducibility.

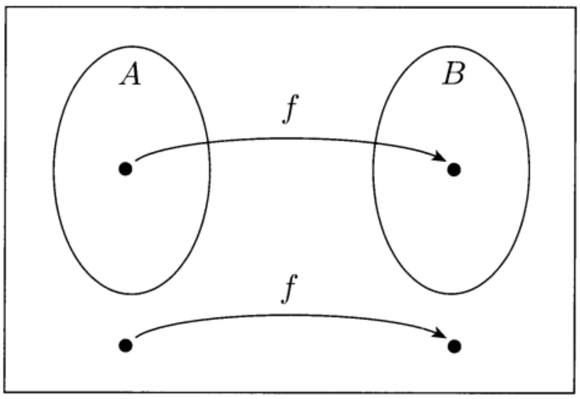


FIGURE 5.21
Function f reducing A to B

A mapping reduction of A to B provides a way to convert questions about membership testing in A to membership testing in B. To test whether $w \in A$, we use the reduction f to map w to $f(w)$ and test whether $f(w) \in B$. The term *mapping reduction* comes from the function or mapping that provides the means of doing the reduction.

If one problem is mapping reducible to a second, previously solved problem, we can thereby obtain a solution to the original problem. We capture this idea in the following theorem.

THEOREM 5.22

If $A \leq_m B$ and B is decidable, then A is decidable.

PROOF We let M be the decider for B and f be the reduction from A to B. We describe a decider N for A as follows.

N = "On input w:

1. Compute $f(w)$.
2. Run M on input $f(w)$ and output whatever M outputs."

Clearly, if $w \in A$, then $f(w) \in B$ because f is a reduction from A to B. Thus M accepts $f(w)$ whenever $w \in A$. Therefore N works as desired.

and rejects $f(w)$ whenever $w \notin A$.

The following corollary of Theorem 5.22 ~~has been~~ will be our main tool for proving undecidability.

COROLLARY 5.23

If $A \leq_m B$ and A is undecidable, then B is undecidable.

3 Prove that the problem associated with language A_{TM} defined below is undecidable. You are given that HALT = {<*M*, *w*>: *M* is a Turing machine and *M* halts on *w*} is undecidable. Use the template provided to perform a mapping reduction. You must give your answer on this exam question sheet.

The language L_3 is defined as
A_{TM} = {<*M, w*>: *M* is a TM that accepts *w*}.

Note, in the template below, some blanks have a small subscript number. Blanks with the same subscript number must have the same value.

<u>Proof</u>

We will use a mapping reduction to prove the reduction

. . . HALT. . . . . ≤ . . . . A_{TM}. . . . .

Assume that . . . . A_{TM} . . . .$_1$ is decidable.

The transition function *f* that maps instances of . . . HALT . . . . to instances of . A_{TM}. . . . . is given by TM F given by the following pseudocode.

F = "On input <. . . M, w. . . . .$_2$>:

1. Construct the following N given by the following pseudocode.

   N = "On input u:. . . . .

   . .1. Run M on w. . . . .

   . .2. Accept . . . .

   . . . .

   . . . ."

2. Output <. . . N, u . . . .$_3$>."

Now, <. . . .N, u . . . .$_3$> is an element of . . . A_{TM} . . . . iff <. . M,w. . .$_2$> is an element of . .HALT. . ..

So, using *f* and the assumption that . . A_{TM}. .$_1$ is decidable, we can decide . . HALT. ..

A contradiction.

Therefore, . .A_{TM}. .$_1$ is undecidable. (This also means that the complement of . . A_{TM}. . .$_1$ is undecidable; the complement of any undecidable language is itself undecidable.)

As explained by Theorem 5.22 above