REDUCIBILITY

21 February 2024 04:24 PM

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.

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.

problem, $HALT_{\mathsf{TM}}$, the problem of determining whether a Turing machine halts (by accepting or rejecting) on a given input.[1] We use the undecidability of A_{TM} to prove the undecidability of $HALT_{\mathsf{TM}}$ by reducing A_{TM} to $HALT_{\mathsf{TM}}$. Let

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

THEOREM 5.1

$HALT_{\mathsf{TM}}$ is undecidable.

PROOF Let's assume for the purposes of obtaining a contradiction that TM R decides $HALT_{\mathsf{TM}}$. We construct TM S to decide A_{TM}, with S operating as follows.

S = "On input $\langle M, w\rangle$, an encoding of a TM M and a string w:

1. Run TM R on input $\langle M, w\rangle$.
2. If R rejects, *reject*.
3. If R accepts, simulate M on w until it halts.
4. If M has accepted, *accept*; if M has rejected, *reject*."

Clearly, if R decides $HALT_{\mathsf{TM}}$, then S decides A_{TM}. Because A_{TM} is undecidable, $HALT_{\mathsf{TM}}$ also must be undecidable.

of the reducibility method for proving undecidability. Let

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

THEOREM 5.2

E_{TM} is undecidable.

PROOF Let's write the modified machine described in the proof idea using our standard notation. We call it M_1.

M_1 = "On input x:

1. If $x \neq w$, *reject*.
2. If $x = w$, run M on input w and *accept* if M does."

This machine has the string w as part of its description. It conducts the test of whether $x = w$ in the obvious way, by scanning the input and comparing it character by character with w to determine whether they are the same.

Putting all this together, we assume that TM R decides E_{TM} and construct TM S that decides A_{TM} as follows.

S = "On input $\langle M, w\rangle$, an encoding of a TM M and a string w:

1. Use the description of M and w to construct the TM M_1 just described.
2. Run R on input $\langle M_1\rangle$.
3. If R accepts, *reject*; if R rejects, *accept*."

Note that S must actually be able to compute a description of M_1 from a description of M and w. It is able to do so because it needs only add extra states to M that perform the $x = w$ test.

If R were a decider for E_{TM}, S would be a decider for A_{TM}. A decider for A_{TM} cannot exist, so we know that E_{TM} must be undecidable.

whether the Turing machine recognizes a regular language. Let

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

THEOREM 5.3

$REGULAR_{TM}$ is undecidable.

PROOF IDEA As usual for undecidability theorems, this proof is by reduction from A_{TM}. We assume that $REGULAR_{TM}$ is decidable by a TM R and use this assumption to construct a TM S that decides A_{TM}. Less obvious now is how to use R's ability to assist S in its task. Nonetheless we can do so.

The idea is for S to take its input $\langle M, w \rangle$ and modify M so that the resulting TM recognizes a regular language if and only if M accepts w. We call the modified machine M_2. We design M_2 to recognize the nonregular language $\{0^n1^n \mid n \geq 0\}$ if M does not accept w, and to recognize the regular language Σ^* if M accepts w. We must specify how S can construct such an M_2 from M and w. Here, M_2 works by automatically accepting all strings in $\{0^n1^n \mid n \geq 0\}$. In addition, if M accepts w, M_2 accepts all other strings.

PROOF We let R be a TM that decides $REGULAR_{TM}$ and construct TM S to decide A_{TM}. Then S works in the following manner.

S = "On input $\langle M, w \rangle$, where M is a TM and w is a string:

1. Construct the following TM M_2.
   M_2 = "On input x:
   1. If x has the form 0^n1^n, *accept*.
   2. If x does not have this form, run M on input w and *accept* if M accepts w."
2. Run R on input $\langle M_2 \rangle$.
3. If R accepts, *accept*; if R rejects, *reject*."

Similarly, the problems of testing whether the language of a Turing machine is a context-free language, a decidable language, or even a finite language, can be shown to be undecidable with similar proofs. In fact, a general result, called

proof by reduction from E_{TM}. Let

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

THEOREM 5.4

EQ_{TM} is undecidable.

PROOF IDEA Show that, if EQ_{TM} were decidable, E_{TM} also would be decidable, by giving a reduction from E_{TM} to EQ_{TM}. The idea is simple. E_{TM} is the problem of determining whether the language of a TM is empty. EQ_{TM} is the problem of determining whether the languages of two TMs are the same. If one of these languages happens to be $\emptyset$, we end up with the problem of determining whether the language of the other machine is empty—that is, the E_{TM} problem. So in a sense, the E_{TM} problem is a special case of the EQ_{TM} problem wherein one of the machines is fixed to recognize the empty language. This idea makes giving the reduction easy.

PROOF We let TM R decide EQ_{TM} and construct TM S to decide E_{TM} as follows.

S = "On input $\langle M \rangle$, where M is a TM:

1. Run R on input $\langle M, M_1 \rangle$, where M_1 is a TM that rejects all inputs.
2. If R accepts, *accept*; if R rejects, *reject*."

If R decides EQ_{TM}, S decides E_{TM}. But E_{TM} is undecidable by Theorem 5.2, so EQ_{TM} also must be undecidable.

MAPPING REDUCIBLITY

Our choice is a simple type of reducibility called ***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

DEFINITION 5.17

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

EXAMPLE 5.18

All usual arithmetic operations on integers are computable functions. For example, we can make a machine that takes input $\langle m, n \rangle$ and returns $m + n$, the sum of m and n. We don't give any details here, leaving them as exercises.

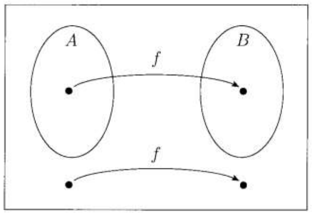
DEFINITION 5.20

Language A is ***mapping reducible*** to language B, written $A \leq_m B$, if there is a computable function $f : \Sigma^* \longrightarrow \Sigma^*$, 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.



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.

COROLLARY 5.23

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

THEOREM 5.28

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

COROLLARY 5.29

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

THEOREM 5.30

EQ_{TM} is neither Turing-recognizable nor co-Turing-recognizable.

PROOF First we show that EQ_{TM} is not Turing-recognizable. We do so by showing that A_{TM} is reducible to $\overline{EQ_{\mathsf{TM}}}$. The reducing function f works as follows.

F = "On input $\langle M, w\rangle$ where M is a TM and w a string:
1. Construct the following two machines M_1 and M_2.
   M_1 = "On any input:
   1. *Reject*."

   M_2 = "On any input:
   1. Run M on w. If it accepts, *accept*."
2. Output $\langle M_1, M_2\rangle$."

Here, M_1 accepts nothing. If M accepts w, M_2 accepts everything, and so the two machines are not equivalent. Conversely, if M doesn't accept w, M_2 accepts nothing, and they are equivalent. Thus f reduces A_{TM} to $\overline{EQ_{\mathsf{TM}}}$, as desired.

To show that $\overline{EQ_{\mathsf{TM}}}$ is not Turing-recognizable we give a reduction from A_{TM} to the complement of $\overline{EQ_{\mathsf{TM}}}$—namely, EQ_{TM}. Hence we show that $A_{\mathsf{TM}} \leq_{\mathrm{m}} EQ_{\mathsf{TM}}$. The following TM G computes the reducing function g.

G = "The input is $\langle M, w\rangle$ where M is a TM and w a string:
1. Construct the following two machines M_1 and M_2.
   M_1 = "On any input:
   1. *Accept*."

   M_2 = "On any input:
   1. Run M on w.
   2. If it accepts, *accept*."
2. Output $\langle M_1, M_2\rangle$."

The only difference between f and g is in machine M_1. In f, machine M_1 always rejects, whereas in g it always accepts. In both f and g, M accepts w iff M_2 always accepts. In g, M accepts w iff M_1 and M_2 are equivalent. That is why g is a reduction from A_{TM} to EQ_{TM}.

Check-in 9.2

Suppose $A \leq_{\mathrm{m}} B$.

What can we conclude?

Check all that apply.

(a) $B \leq_{\mathrm{m}} A$

(b) $\overline{A} \leq_{\mathrm{m}} \overline{B}$

(c) None of the above

MIT

Noteworthy difference:

- A is reducible to $\overline{A}$
- A may not be mapping reducible to $\overline{A}$.
  For example $\overline{A_{\mathrm{TM}}} \not\leq_{\mathrm{m}} A_{\mathrm{TM}}$

Check-in 9.3

We showed that if $A \leq_{\mathrm{m}} B$ and B is T-recognizable then so is A.

Is the same true if we use general reducibility instead of mapping reducibility?

(a) Yes

(b) No

To prove B is undecidable:

- Show undecidable A is reducible to B. (often A is A_{TM})
- Template: Assume TM R decides B.
  Construct TM S deciding A. Contradiction.

To prove B is T-unrecognizable:

- Show T-unrecognizable A is mapping reducible to B. (often A is $\overline{A_{\text{TM}}}$)
- Template: give reduction function f.

9. Reducibility

E_{TM} is T-unrecognizable

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

Theorem: E_{TM} is T-unrecognizable

Proof: Show $\overline{A_{\text{TM}}} \le_{\text{m}} E_{\text{TM}}$

Reduction function: $f(\langle M, w\rangle) = \langle M_w\rangle$

Recall TM M_w = "On input x

1. If $x \neq w$, *reject*.
2. else run M on w
3. *Accept* if M accepts."

Explanation: $\langle M, w\rangle \in \overline{A_{\text{TM}}}$ iff $\langle M_w\rangle \in E_{\text{TM}}$

M rejects w iff $L(\langle M_w\rangle) = \emptyset$

EQ_{TM} and EQ_{TM} are T-unrecognizable

$EQ_{\mathrm{TM}} = \{\langle M_1, M_2\rangle \mid M_1$ and M_2 are TMs and $L(M_1) = L(M_2)\}$

Theorem: Both EQ_{TM} and $\overline{EQ_{\mathrm{TM}}}$ are T-unrecognizable

Proof: (1) $\overline{A_{\mathrm{TM}}} \le_{\mathrm{m}} EQ_{\mathrm{TM}}$

(2) $\overline{A_{\mathrm{TM}}} \le_{\mathrm{m}} \overline{EQ_{\mathrm{TM}}}$

For any w let T_w = "On input x

1. Ignore x.
2. Simulate M on w."

(1) Here we give f which maps $\overline{A_{\mathrm{TM}}}$ problems (of the form $\langle M, w\rangle$) to EQ_{TM} problems (of the form $\langle T_1, T_2\rangle$).

$f(\langle M, w\rangle) = \langle T_w, T_{\text{reject}}\rangle$ T_{reject} is a TM that always rejects.

(2) Similarly $f(\langle M, w\rangle) = \langle T_w, T_{\text{accept}}\rangle$ T_{accept} always accepts.