

Mapping Reducibility

Teodor Rus

`rus@cs.uiowa.edu`

The University of Iowa, Department of Computer Science

Formalizing “problem reduction”

- To reduce a problem A to a problem B by mapping reducibility means to find a computable function, $f : A \rightarrow B$ called a *reduction*, that converts instances of A into instances of B .
- If a reduction $f : A \rightarrow B$ exists then we can solve problem A with a solver of problem B , by first mapping A into B and then using the solver of B .

Note

We have applied this method intuitively to both, solvability and unsolvability

Here we will formalize this method and will illustrate the formal approach thus obtained with examples already treated informal.

Computable functions

- A function $f : \Sigma^* \rightarrow \Sigma^*$ is computable if some TM M exists which on every input w halts with $f(w)$ on the tape.
- **Example:** All usual arithmetic functions on integers are computable: $\langle m, n \rangle \mapsto m + n$, $\langle m, n \rangle \mapsto m * n$, etc.
- Computable functions map strings into strings. TM descriptions can be represented by strings.

Conclusion: computable functions may be transformations of machine descriptions

Machine transformation

- A computable function f may take an input w , where w is an encoding of a TM M , i.e., $w = \langle M \rangle$, and may return the description of another TM, $\langle M' \rangle = f(w)$

Example

- For a TM M , M' is a TM that recognizes the same language as M but never attempts to move its head over the left-hand end of its tape.
- f can map $w = \langle M \rangle$ into $w' = \langle M' \rangle$ by adding several states to the description of M ;
- f returns ϵ if w is illegal

Mapping reducibility

To provide a formal definition of mapping reducibility we represent computational problems by languages, as usual.

Definition: language A is mapping reducible to language B , $A \leq_m B$, if there is a computable function $f : \Sigma^* \rightarrow \Sigma^*$ where

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

The function f is called a reduction of A to B .

Note: the arrow \iff above should be interpreted as:

1. For each $w \in A$ there exists $f(w) \in B$
2. The image $f(w) \in B$ of $w \in A$ only states that $w \in A$; it does not required w to be unique.

Pictorial

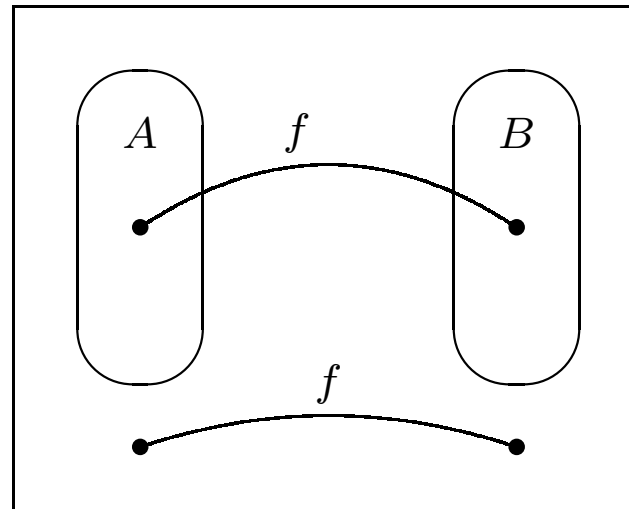


Figure 1: Function f reducing A to B

Facts

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

Observation

If one problem is mapping reducible to a second, previously solved problem, we can thereby obtain a solution to the first problem.

Theorem 5.22

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

Proof: let M be a decider for B and $f : A \rightarrow B$ be the reduction function from A to B . The TM N that decides A is:

$N =$ "On input w :

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

Fact

- If $w \in A$ then $f(w) \in B$ because f is a reduction from A to B ;
- Thus, M accepts $f(w)$ whenever $w \in A$, i.e, $f(w) \in B$, and thus N works as desired.

Corollary 5.23

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

Proof: by contradiction.

Assume that B is decidable. Since $A \leq_m B$, using theorem 5.22 it results that A is decidable. But this is a contradiction, because A is assumed to be undecidable.

Example

To prove that $HALT_{TM}$ is undecidable the proof of Theorem 5.1 used the intuitive method of reduction from A_{TM} .

To reduce A_{TM} to $HALT_{TM}$ we assume that $HALT_{TM}$ is decidable and let TM R decides it. Then we construct the 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. 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 accepted, *accept*; if M rejected, *reject*".

Constructing a reduction $f : A_{TM} \rightarrow HALT_{TM}$

We can demonstrate a mapping reduction from A_{TM} to $HALT_{TM}$ as follows:

- Find a computable function f that inputs $\langle M, w \rangle$ and outputs $\langle M', w' \rangle$ such that

$$\langle M, w \rangle \in A_{TM} \text{ iff } \langle M', w' \rangle \in HALT_{TM}$$

TM F that computes f

F = "On input $\langle M, w \rangle$:

1. Construct the following machine M' :

M' = "On input x :

(a) Run M on x .

(b) If M accepts, *accept*.

(c) If M rejects, enter a loop."

2. Output $\langle M', w \rangle$."

Note: Since $A_{TM} = \{\langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$ and $HALT_{TM} = \{\langle M, w \rangle \mid M \text{ is a TM and } M \text{ halts on } w\}$ we have:

1. If $\langle M, w \rangle \in A_{TM}$ then M' accepts w and halts, that is

$\langle M', w \rangle \in HALT_{TM}$

2. If $\langle M', w \rangle \in HALT_{TM}$ then $\langle M', w \rangle \in A_{TM}$ because it halts on w .

Mapping reducibility is transitive

Proof: suppose $A \leq_m B$ and $B \leq_m C$

- There are computable functions f and g such that:
 $\forall x \in A \Leftrightarrow f(x) \in B$ and $\forall y \in B \Leftrightarrow g(y) \in C$.
- Consider the function composition $h(x) = g(f(x))$. We can build the TM M_h that compute h as follows:

TM M_h

$M_h =$ "On input x :

1. Simulate a TM that computes f on input x and let the result be y (such TM exists because f is computable);
2. Simulate a TM that computes g on input y and let the result be $z = h(f(x)) \in C$ (such TM exists because g is computable)."

Note: h is computable and $\forall x \in A \Leftrightarrow h(x) \in C$, i.e., $A \leq_m C$

Application

Proof of the undecidability of PCP contains two mapping reductions:

- $A_{TM} \leq_m MPCP$ by $f_1 : A_{TM} \rightarrow MPCP$

Note: this was done by mapping a $\langle M, w \rangle$ into a particular instance of PCP where a match starts with a given domino ($[\#/\#q_0w_1 \dots w_n\#]$)

- $MPCP \leq_m PCP$ by $f_2 : MPCP \rightarrow PCP$

Note: this is done by removing the separators $\#$ from the puzzles.

- $A_{TM} \leq_m PCP$ is defined by $f = f_2(f_1(\langle M, w \rangle))$

Note: we have effectively constructed the mapping reductions f_1 , f_2 , and f .

Another example

Theorem 5.2. $E_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$ is undecidable.

Proof: Construct the following reduction $f : A_{TM} \rightarrow E_{TM}$:
 $S =$ "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 M_1 :
 $M_1 =$ "On input x :
 - (a) If $x \neq w$, *reject*
 - (b) If $x = w$, run M on input w and accept if M does."
2. Run R on input $\langle M_1 \rangle$
3. If R accept, *rejects*; if R rejects, *accept*."

Converting S to a mapping reduction

1. The function $f : \langle M, w \rangle \rightarrow \langle M_1 \rangle$ is defined in S .
2. The problem arises because M accepts w iff $L(M_1) \neq \emptyset$, hence f perform the reduction $f : A_{TM} \rightarrow \overline{E_{TM}}$.
3. This reduction still shows that E_{TM} is undecidable because decidability is not affected by complementation.

Note: This construction does not give a reduction

$f : A_{TM} \rightarrow E_{TM}$ because such a reduction does not exist.

$f : A_{TM} \rightarrow E_{TM}$ does not exist

Proof: by contradiction (see Selected Solutions, 5.5 page 214)

1. Suppose for a contradiction that $A_{TM} \leq_m E_{TM}$ via a function f .
2. From the definition of mapping reduction follows that $\overline{A_{TM}} \leq_m \overline{E_{TM}}$ via the same reduction function f .

Remember: $\forall w \in A_{TM} \iff f(w) \in E_{TM}$.

3. But $\overline{E_{TM}}$ is Turing-recognizable and $\overline{A_{TM}}$ is Turing-unrecognizable (remember Corollary 4.23). This is a contradiction (as we will see next).

Observations

- The sensitivity of mapping reducibility to complementation is important in the use of reducibility to prove unrecognizability of some languages;
- We can use mapping reducibility to show that problems are not Turing-recognizable.

Theorem 5.28

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

Proof idea: the same as in Theorem 5.22, except that M and N are recognizers instead of deciders.

Proof: Assume that B is Turing recognizable. Let M , be a recognizer for B , $f : A \rightarrow B$ a reduction function from A to B . The TM N that recognizes A is defined by

$N =$ "On input w :

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

Note: since M is a recognizer for B and f is a mapping reduction of A to B , N is a recognizer for A .

Application

All Turing-recognizable languages mapping reduce to A_{TM}
(Problem 5.9 from Sipser, Edition 1)

Proof: Assume that L is the language of a Turing-recognizable problem and let M be the TM that recognizes L .

- To reduce L to A_{TM} we map any string x in the alphabet of L into $\langle M, x \rangle$. Since M and x are given this is computable;
- Testing $x \in L$ is the same as testing $\langle M, x \rangle \in A_{TM}$.

Corollary 5.29

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

Proof: by contradiction, as usual.

1. Assuming that B is Turing-recognizable and $A \leq_m B$;
2. On the basis of Theorem 5.28, it results that A is Turing-recognizable;
3. But we know that A is not Turing-recognizable, what represents a contradiction.

Conclusion: B cannot be Turing-recognizable.

Application of corollary 5.29

- Let A be $\overline{A_{TM}}$. From corollary 4.23 (4.17) (if A and \overline{A} are Turing recognizable then A is decidable) we know that $\overline{A_{TM}}$ is not Turing-recognizable.
- The definition of mapping reducibility implies that $A \leq_m B$ is the same as $\overline{A} \leq_m \overline{B}$ ($f : \Sigma^* \rightarrow \Sigma^*$ such that $x \in A \iff f(x) \in B$ implies $x \notin A \Rightarrow f(x) \notin B$, i.e., $x \in \overline{A} \Rightarrow f(x) \in \overline{B}$, that is $\overline{A} \leq_m \overline{B}$);
- To prove that B isn't recognizable we may show that $A_{TM} \leq_m \overline{B}$. Since $A \leq_m B$ means the same as $\overline{A} \leq_m \overline{B}$ we have $\overline{A_{TM}} \leq_m \overline{\overline{B}}$, i.e., $\overline{A_{TM}} \leq_m B$;
- We can also use mapping reducibility to show that certain problems are neither Turing-recognizable nor co-Turing-recognizable.

Theorem 5.30

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

Proof: First, show that EQ_{TM} is not Turing-recognizable.

We show first that A_{TM} is reducible to $\overline{EQ_{TM}}$ by a reduction f .

Note: Since $A_{TM} \leq_m \overline{EQ_{TM}}$ implies $\overline{A_{TM}} \leq_m \overline{\overline{EQ_{TM}}}$ this means $\overline{A_{TM}} \leq_m EQ_{TM}$, as required.

$$f : A_{TM} \rightarrow \overline{EQ_{TM}}$$

The TM F that defines the reduction $f : A_{TM} \rightarrow \overline{EQ_{TM}}$ works as follows:

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

1. Construct the two machines M_1 and M_2 :

M_1 = "On any input:

(a) *Reject*."

M_2 = "On any input:

(a) Run M on w . If it accepts, *accept*".

2. Output $\langle M_1, M_2 \rangle$."

Fact

- If M accepts w , M_1 and M_2 are not equivalent:
 - M_1 accepts nothing
 - M_2 accepts everything.
- If M doesn't accept w , M_1 and M_2 are equivalent:
 - M_1 accepts nothing
 - M_2 accepts nothing.

Thus, f reduces A_{TM} to $\overline{EQ_{TM}}$ as desired.

Conclusion :

taking the complements we obtain $\overline{A_{TM}} \leq_m EQ_{TM}$ which (by Theorem 5.29) shows that EQ_{TM} is not Turing recognizable.

Proof, continuation

Now to show that EQ_{TM} is not co-Turing recognizable we will show that $\overline{EQ_{TM}}$ is not Turing recognizable.

- To show that $\overline{EQ_{TM}}$ is not Turing-recognizable we give a reduction function g from A_{TM} to the complement of $\overline{EQ_{TM}}$ which is EQ_{TM} , i.e., $A_{TM} \leq_m EQ_{TM}$
- The TM G that computes this reduction follows:

$$g : A_{TM} \rightarrow EQ_{TM}$$

The TM G that computes function g follows:

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

1. Construct the two machines M_1 and M_2 :

$M_1 =$ "On any input:

(a) *Accept*."

$M_2 =$ "On any input:

(a) Run M on w . If it accepts, *accept*".

2. Output $\langle M_1, M_2 \rangle$."

Note

- If M accepts w , M_1 and M_2 are equivalent:
 - M_1 accepts everything
 - M_2 accepts everything.
- If M doesn't accept w , M_1 and M_2 are not equivalent:
 - M_1 accepts everything
 - M_2 accepts nothing.

Thus, g reduces A_{TM} to EQ_{TM} as desired.

EQ_{TM} is not co-Turing recognizable

1. Taking the complements of $A_{TM} \leq_m EQ_{TM}$ established by g we have $\overline{A_{TM}} \leq_m \overline{EQ_{TM}}$.
2. But we know that $\overline{A_{TM}}$ is not Turing-recognizable.
3. Hence, (by Theorem 5.29) $\overline{EQ_{TM}}$ is not Turing-recognizable.

Observations

1. The only difference between f and g is in the machine M_1 which rejects everything in f and accepts everything in g .
2. In both f and g , M accepts w iff M_2 always accepts.
3. In g , M accepts w iff M_1 and M_2 are equivalent. This is why g is a reduction from A_{TM} to EQ_{TM} .

Application

Problem: test whether a two-tape TM ever writes a non-blank symbol on its second tape. Formulate this problem as a language and show that this language is undecidable.

Solution

- **The language:** $B = \{\langle M \rangle \mid M \text{ is two tape TM which writes a non-blank symbol on its second tape when it is simulated on some particular input } w\}$.
- We will show that $A_{TM} \leq_m B$ by mapping $\langle M, w \rangle$ to $\langle M' \rangle$, where M' has the description:
 $M' =$ "On input x :
 1. Simulate M on w only using the first tape.
 2. If M accepts w then write blah blah ! on the second tape."

Note: observe that $\langle M, w \rangle \in A_{TM} \Leftrightarrow M' \in B$. That is, M' performs the reduction.