Reductions

Undecidability of the Halting Problem

Introduction

The reduction technique enables us to use the undecidability of A_{TM} to prove many other languages undecidable.

Introduction

A reduction always involves two computational problems. Generally speaking, the idea is to show that a solution for some problem B induces a solution for problem A. If we know that A does not have a solution, we may deduce that B is also insolvable. In this case we say that A is reducible to B.

Introduction

In the context of undecidability, to prove that a certain language L is undecidable:

Assume by way of contradiction that L is decidable, and show that a decider for L can be used to devise a decider for A_{TM}. Since A_{TM} is undecidable, so is the language L.

Introduction

Using a decider for L to construct a decider for A_{TM}, is called ***reducing*** A_{TM} ***to*** L.

Note: Once we prove that a certain language L_1 is undecidable, we can prove that some other language, say L_2, is undecidable, by reducing L_1 to L_2.

Schematic of a Reduction

1. We know that A is undecidable.
2. We want to prove B is undecidable.
3. We assume that B is decidable and use this assumption to prove that A is decidable.
4. We conclude that B is undecidable.

Note: The reduction is ***from*** A ***to*** B.

Demonstration

1. We know that A is undecidable.

   The only undecidable language we know, so far, is A_{TM} whose undecidability was proven directly. So we pick A_{TM} to play the role of A.

2. We want to prove B is undecidable.

Demonstration

2. We want to prove B is undecidable.
   We pick $HALT_{TM}$ to play the role of B that is:
   We want to prove that $HALT_{TM}$ is undecidable.
3. We assume that B is decidable and use this assumption to prove that A is decidable.

Demonstration

3. We assume that B is decidable and use this assumption to prove that A is decidable.
   In the following slides we assume (towards a contradiction) that $HALT_{TM}$ is decidable and use this assumption to prove that A_{TM} is decidable.

4. We conclude that B is undecidable.

The Halting Problem

Consider

$$HALT_{TM} = \{\langle M, w \rangle \mid M \text{ is a TM that halts on } w\}$$

Theorem

$HALT_{TM}$ is undecidable.

Proof

By reducing A_{TM} to $HALT_{TM}$.

Discussion

Assume by way of contradiction that $HALT_{TM}$ is decidable.

Recall that a decidable set has a ***decider*** R: A TM that halts on every input and either accepts or rejects, but ***never loops!***

We will use the assumed decider of $HALT_{TM}$ to devise a decider for A_{TM} .

Discussion

Recall the definition of A_{TM} :

$$A_{TM} = \{\langle M, w \rangle \mid M \text{ is a TM that accepts } w\}$$

Why is it impossible to decide A_{TM} ?

Because as long as M runs, we cannot determine whether it will eventually halt.

Well, now we can, using the ***decider*** R for $HALT_{TM}$.

Proof

Assume by way of contradiction that $HALT_{TM}$ is decidable and let R be a TM deciding it. In the next slide we present TM S that uses R as a subroutine and decides A_{TM} . Since A_{TM} is undecidable this constitutes a contradiction, so R does not exist.

Proof (cont.)

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

1. Run R on input $\langle M, w \rangle$ until it halts.
2. If R rejects, (i.e. M loops on w) - *reject*.

(At this stage we know that R accepts, and we conclude that M halts on input w.)

3. Simulate M on w until it halts.
4. If M accepts - *accept*, otherwise - *reject*. "

Another Example

In the discussion, you saw how Diagonalization can be used to prove that $HALT_{TM}$ is not decidable.

We can use this result to prove by reduction that A_{TM} is not decidable.

Another Example

Note: Since we already know that both A_{TM} and $HALT_{TM}$ are undecidable, this new proof does not add any new information. We bring it here only for the the sake of demonstration.

Demonstration

1. We know that A is undecidable.
   Now we pick $HALT_{TM}$ to play the role of A.
2. We want to prove B is undecidable.
   We pick A_{TM} to play the role of B, that is: We want to prove that A_{TM} is undecidable.
3. We assume that B is decidable and use this assumption to prove that A is decidable.

Demonstration

3. We assume that B is decidable and use this assumption to prove that A is decidable.
   In the following slides we assume that A_{TM} is decidable and use this assumption to prove that $HALT_{TM}$ is decidable.
4. We conclude that B is undecidable.

Discussion

Let R be a decider for A_{TM}. Given an input for $\langle M, w \rangle$, R can be run with this input :
If R accepts, it means that $\langle M, w \rangle \in A_{TM}$.
This means that M accepts on input w. In particular, M stops on input w. Therefore, a decider for $HALT_{TM}$ must accept $\langle M, w \rangle$ too.

Discussion

If however R rejects on input $\langle M, w \rangle$, a decider for $HALT_{TM}$ cannot safely reject: M may be halting on w to ***reject it***. So if M rejects w, a decider for $HALT_{TM}$ ***must*** accept $\langle M, w \rangle$.

Discussion

How can we use our decider for A_{TM} ?

The answer here is more difficult. The new decider should first ***modify the input TM***, M, so the modified TM, M_1, ***accepts, whenever*** TM M ***halts***.

Since M is a part of the input, the modification must be ***a part of the computation***.

Discussion

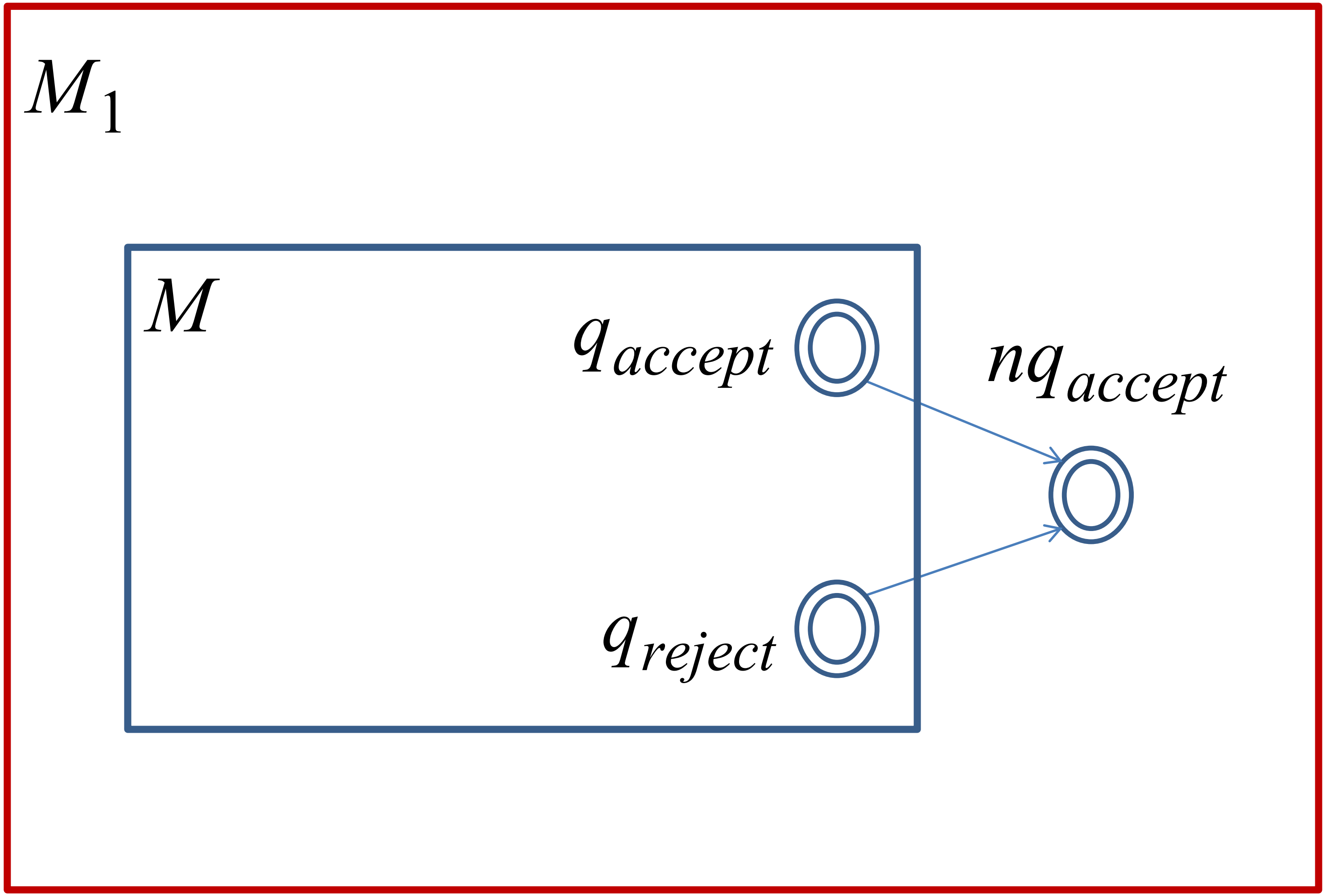
Faithful to our principal *"If it ain't broken don't fix it"*, the modified TM keeps M as a subroutine, and the idea is quite simple:
Let q_{accept} and q_{reject} be the accepting and rejecting states of TM M, respectively. In the modified TM, M_1 ,q_{accept} and q_{reject} are kept as ordinary states.

Discussion

We continue the modification of M by adding a new accepting sate nq_{accept}. Then we add two new transitions: A transition from q_{accept} to nq_{accept}, and another transition from q_{reject} to nq_{accept}.

This completes the description of M_1. It is not hard to verify that M_1 accepts ***iff*** M halts.

Discussion



Discussion

The final description of a decider S for A_{TM} is:

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

1. Modify M as described to get M_1.
2. Run R, the decider of $HALT_{TM}$ with input $\langle M_1, w \rangle$.
3. If R accepts - *accept*, otherwise - *reject*."

Discussion

It should be noted that modifying TM M to get M_1, is part of TM S, the new decider for $HALT_{TM}$, and can be carried out by it.

It is not hard to see that S decides $HALT_{TM}$. Since $HALT_{TM}$ is undecidable, we conclude that A_{TM} is undecidable too.

The TM Emptiness Problem

We continue to demonstrate reductions by showing that the language E_{TM} , defined by

$$E_{TM} = \{\langle M \rangle \mid M \text{ is a TM And } L(\mathrm{M}) = \phi\}$$

is undecidable.

Theorem

E_{TM} is undecidable.

Proof Outline

The proof is by reduction ***from*** A_{TM} :

1. We know that A_{TM} is undecidable.
2. We want to prove E_{TM} is undecidable.
3. We assume toward a contradiction that E_{TM} is decidable and devise a decider for A_{TM} .
4. We conclude that E_{TM} is undecidable.

Proof

Assume by way of contradiction that E_{TM} is decidable and let R be a TM deciding it. In the next slides we devise TM S that uses R as a subroutine and decides A_{TM} .

Proof

Given an instance for A_{TM} , $\langle M, w \rangle$, we may try to run R on this instance. If R accepts, we know that $L(M) = \phi$. In particular, M does not accept w so a decider for A_{TM} must reject $\langle M, w \rangle$.

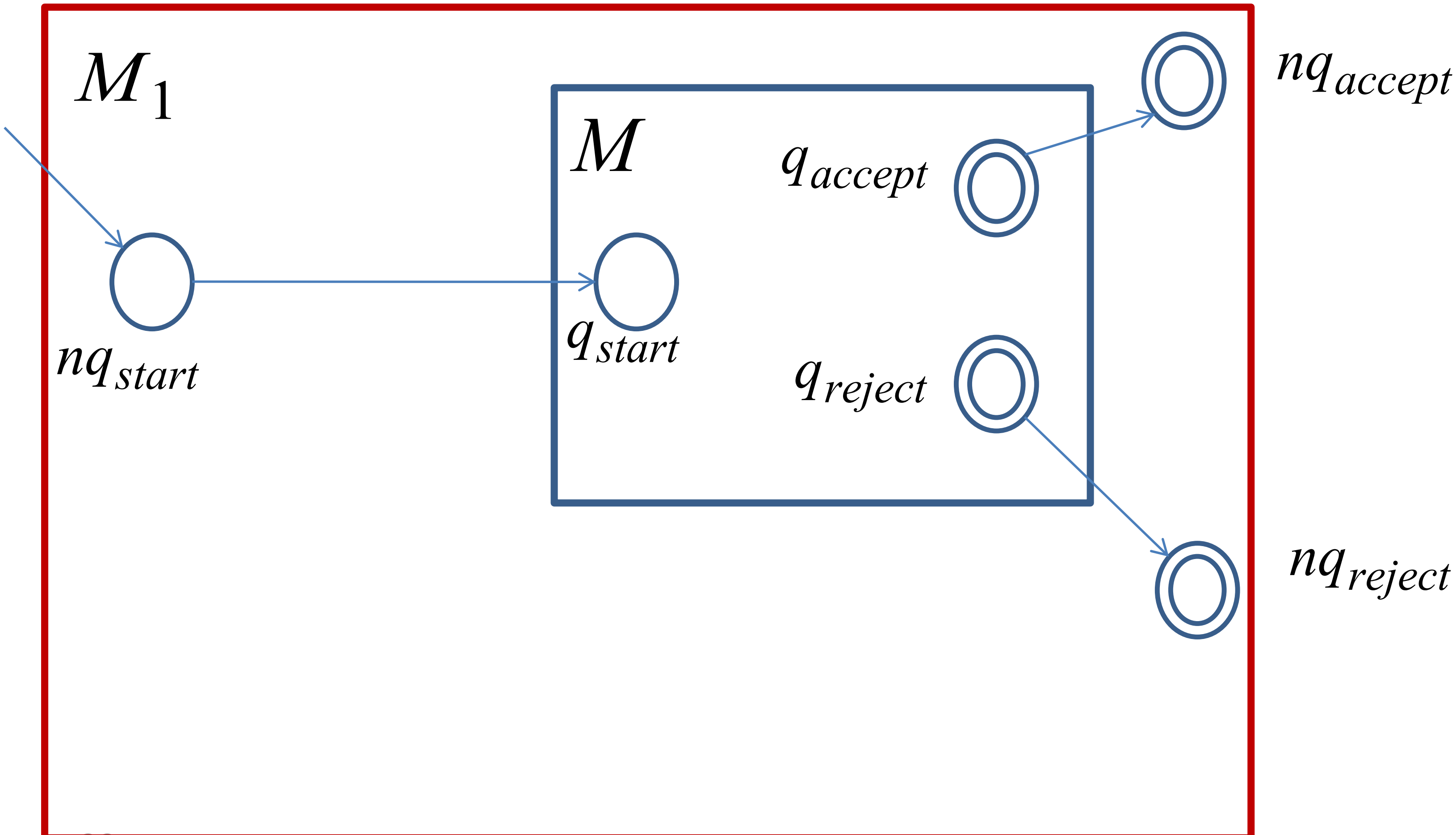
Proof

What happens if R rejects? The only conclusion we can draw is that $L(M) \neq \phi$. What we need to know though is whether $w \in L(M)$.

In order to use our decider R for E_{TM} , we once again modify the input machine M to obtain TM M_1 :

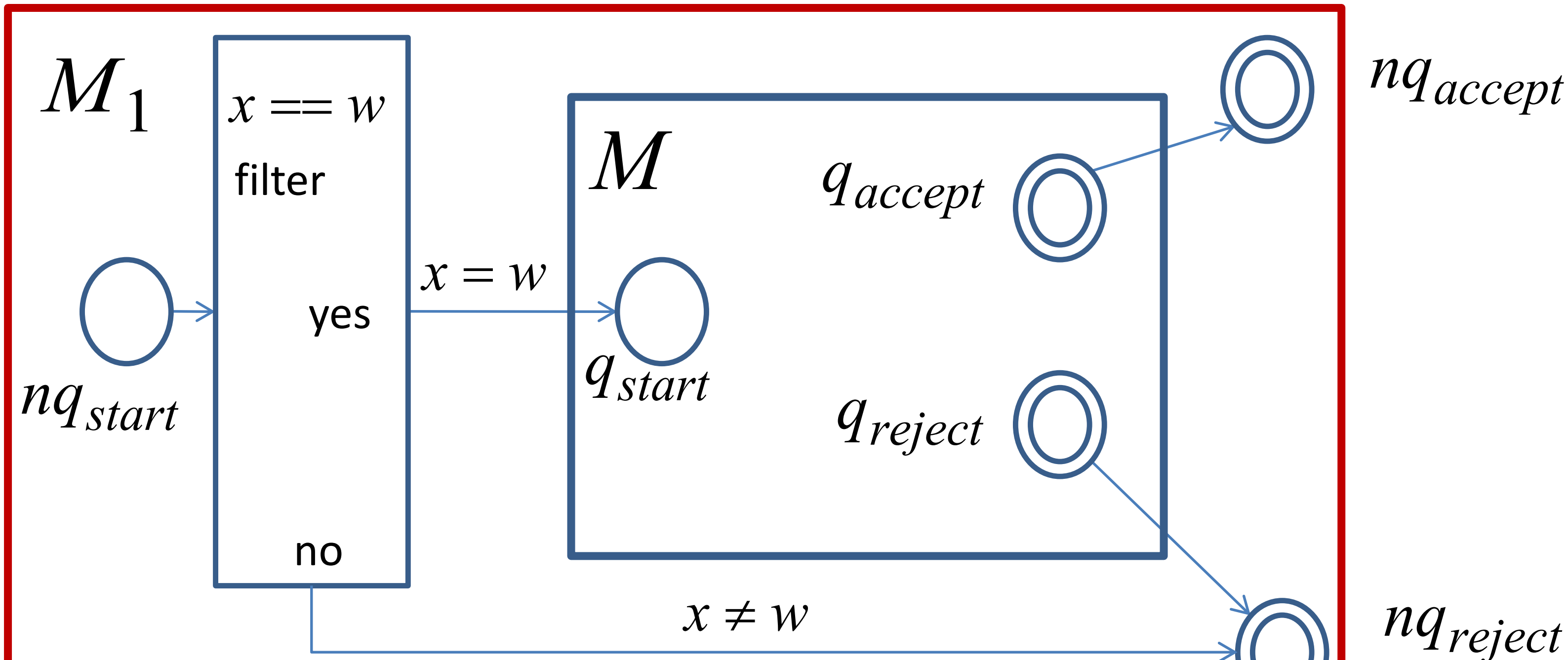
Description of M_1

We start with a TM satisfying $L(M_1) = L(M)$.



Description of M_1

Now we add a ***filter*** to divert all inputs but w.



$$L(M_1) = \begin{cases} \{w\} & \text{if } M \text{ accepts } w \\ \phi & \text{if } M \text{ rejects } w \end{cases}$$

Proof

TM M_1 has a ***filter*** that ***rejects*** all inputs ***excepts*** w, so the only input reaching M, is w.

Therefore, M_1 satisfies:

$$L(M_1) = \begin{cases} \{w\} & \text{if } M \text{ accepts } w \\ \phi & \text{if } M \text{ rejects } w \end{cases}$$

Proof

Here is a formal description of M_1 :

$M_1 =$ "On input x :

1. If $x \neq w$ - *reject* .
2. If $x = w$ - run M on w and *accept* if M accepts. "

Note: M accepts w if and only if $L(M_1) \neq \phi$.

Proof

This way, if R accepts, S “can be sure” that $w \in L(M)$ and accept. Note that S gets the pair $\langle M, w \rangle$ as input, thus before S runs R, it should compute an encoding $\langle M_1 \rangle$ of M_1. This encoding is not too hard to compute using S’s input $\langle M, w \rangle$.

Proof

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

1. Compute an encoding $\langle M_1 \rangle$ of TM M_1.
2. Run R on input $\langle M_1 \rangle$.
3. If R rejects - *accept*, otherwise - *reject*.

Proof

Recall that R is a decider for E_{TM}. If R rejects the modified machine M_1, $L(M_1) \neq \phi$, hence by the specification of M_1, $w \in L(M)$, and a decider for A_{TM} must accept $\langle M, w \rangle$.

If however R accepts, it means that $L(M_1) = \phi$, hence $w \notin L(M)$, and S must reject $\langle M, w \rangle$.

QED