

Computational Models Lecture 9, Spring 2009

- Reducibility among languages
 - Mapping reductions
 - More undecidable languages
 - Undecidability by Rice Theorem
 - Reductions using controlled executions (steppers)
 - \mathcal{RE} -Completeness
- Sipser's book, Chapter 5, Sections 5.1, 5.3

So Far

- We have established Turing Machines as the **gold standard** of computers and computability ...
- seen examples of solvable problems ...
- defined the **universal** TM
- saw two problem, A_{TM} & H_{TM} , that are computationally unsolvable (by a direct diagonalization proof)
- and saw an incomputable function – the busy beaver.

In this lecture, we look at other computationally unsolvable problems, and establish the technique of **mapping reducibilities** for proving that languages are either undecidable or non-enumerable.

Reducibility

Example:

- Finding your way around a new city
- reduces to ...
- obtaining a city map.

Reducibility, In Our Context

Always involves two problems, A and B .

Desired Property: If A reduces to B , then any solution of B can be used to find a solution of A .

Remark: This property says nothing about solving A by itself or B by itself.

Examples

Reductions:

- Traveling from Boston to Paris . . .
- reduces to buying plane ticket . . .
- which reduces to earning the money for that ticket . . .
- which reduces to finding a job
(or getting the \$s from mom and dad. . .)

Examples

Reductions:

- Measuring area of rectangle ...
- reduces to measuring lengths of sides.

Also:

- Solving a system of linear equations ...
- reduces to inverting a matrix.

Reducibility

If A is reducible to B , then

- A cannot be harder than B
- if B is decidable, so is A .
- if A is **undecidable** and reducible to B , then B is **undecidable**.

Additional Undecidable Problems

We have already established that A_{TM} is undecidable.

We also saw the **original** halting problem (of Shoshana and Uri :-), and it was shown (on board) this language is also undecidable.

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

Short reminder: How does H_{TM} differ from A_{TM} ?

Undecidable Problems

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

Theorem: H_{TM} is undecidable.

An alternative proof, by reduction from A_{TM} :

- By contradiction.
- Assume H_{TM} is decidable.
- Let R be a TM that decides H_{TM} .
- Use R to construct S , a TM that decides A_{TM} .
- So A_{TM} is reduced to H_{TM} .
- Since A_{TM} is undecidable, so is H_{TM} .

Undecidable Problems

Theorem: H_{TM} is undecidable.

Proof: Assume, by way of contradiction, that TM R decides H_{TM} . Define a new TM, S , as follows:

- On input $\langle M, w \rangle$,
 - run R on $\langle M, w \rangle$.
 - If R rejects, **reject**.
 - If R accepts (meaning M halts on w), simulate M on w until it halts (namely run U on $\langle M, w \rangle$).
 - If M accepted, **accept**; otherwise **reject**.
- TM S decides A_{TM} , **a contradiction**



Undecidable Problems

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

Theorem: H_{TM} is undecidable.

What we actually did was a reduction
from A_{TM} to H_{TM} .

This will be formalized shortly.

Undecidable Problems (2)

Does a TM accept any string at all?

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

Theorem: EMPTY_{TM} is undecidable.

Proof structure:

- By contradiction.
- Assume EMPTY_{TM} is decidable.
- Let R be a TM that decides EMPTY_{TM} .
- Use R to construct S , a TM that decides A_{TM} .

Undecidable Problems (2)

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

First attempt: When S receives input $\langle M, w \rangle$, it calls R with input $\langle M \rangle$.

- If R accepts, then **reject**, because M does not accept any string, let alone w .
- But what if R **rejects**?

Second attempt: Let's modify M .

Undecidable Problems (2)

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

Define M_1 : on input x ,

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

M_1 either

- accepts **just** w , or
- accepts **nothing**.

Undecidable Problems (2)

Machine M_1 : on input x ,

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

Question:

Can a **TM** construct M_1 from M ?

Answer:

Easily, because we need only hardwire w , and add a few extra states to perform the “ $x = w$?” test.

Undecidable Problems (2)

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

Theorem: EMPTY_{TM} is undecidable.

Define S as follows:

On input $\langle M, w \rangle$, where M is a TM and w a string,

- Construct M_1 from M and w .
- Run R on input $\langle M_1 \rangle$,
- if R accepts, **reject**; if R rejects, **accept**.
- TM S decides A_{TM} , **a contradiction**



Undecidable Problems (3)

Does a TM accept a regular language?

$$\text{REG}_{\text{TM}} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is regular}\}$$

Theorem:

REG_{TM} is undecidable.

Skeleton of Proof:

- By contradiction.
- Assume REG_{TM} is decidable.
- Let R be a TM that decides REG_{TM} .
- Use R to construct S , a TM that decides A_{TM} .

But how?

Undecidable Problems (3)

$$\text{REG}_{\text{TM}} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is regular}\}$$

Intuition: Modify M so that the resulting TM accepts a regular language if and only if M accepts w .

Design M_2 so that

- if M does not accept w , then M_2 accepts the (non-regular) language $\{0^n 1^n \mid n \geq 0\}$
- if M accepts w , then M_2 accepts Σ^* (regular).

Undecidable Problems (3)

Given M and w , construct M_2 :

On input x ,

1. If x has the form $0^n 1^n$, **accept** it.
2. Otherwise, run M on input w and **accept** x if M accepts w .

Claim:

- If M does not accept w , then M_2 accepts $\{0^n 1^n \mid n \geq 0\}$.
- If M accepts w , then M_2 accepts Σ^* .
- The function: On input $\langle M, w \rangle$, output $\langle M_2 \rangle$, is **computable**.

Undecidable Problems (3)

$$\text{REG}_{\text{TM}} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is regular}\}$$

Theorem: REG_{TM} is undecidable.

Define S :

On input $\langle M, w \rangle$,

1. Construct M_2 from M and w .
2. Run R on input $\langle M_2 \rangle$.
3. If R accepts, **accept**; if R rejects, **reject**.

● TM S decides A_{TM} , **a contradiction**



Undecidable Problems (4)

Are two TMs equivalent?

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

Theorem: EQ_{TM} is undecidable.

We are getting tired of reducing A_{TM} to **everything**.

Let's try instead a reduction from EMPTY_{TM} to EQ_{TM} .

Undecidable Problems (4)

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

Theorem: EQ_{TM} is undecidable.

Idea:

- EMPTY_{TM} is the problem of testing whether a TM language is empty.
- EQ_{TM} is the problem of testing whether two TM languages are the same.
- If one of these two TM languages happens to be empty, then we are back to EMPTY_{TM} .
- So EMPTY_{TM} is a special case of EQ_{TM} .

The rest is easy.

Undecidable Problems (4)

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

Theorem: EQ_{TM} is undecidable.

Let M_{NO} be this TM: On input x , **reject**.

Let R decide EQ_{TM} .

Let S be: On input $\langle M \rangle$:

1. Run R on input $\langle M, M_{\text{NO}} \rangle$.
2. If R accepts, **accept**; if R rejects, **reject**.

If R decides EQ_{TM} , then S decides EMPTY_{TM} . ♣

Bucket of Undecidable Problems

Same techniques prove undecidability of

- Does a TM accept a **decidable** language?
- Does a TM accept a **context-free** language?
- Does a TM accept a **finite** language?
- Does a TM halt on **all inputs**?
- Is there an input string that causes a TM to **traverse all its states**?
- Does a TM accept a language that contains **all prime numbers**?

And ...

- Does a TM accept an **enumerable** language? (**really?**)

Reducibility

So far, we have seen many examples of **reductions** from one language to another, but the notion was implicit – it was neither defined nor treated formally.

Reductions play an important role in

- decidability theory (here and now)
- complexity theory (to come)

Time to **get formal**.

Mapping Reductions

Definition: Let A and B be two languages. We say that there is a **mapping reduction** from A to B , and denote

$$A \leq_m B$$

if there is a **computable function**

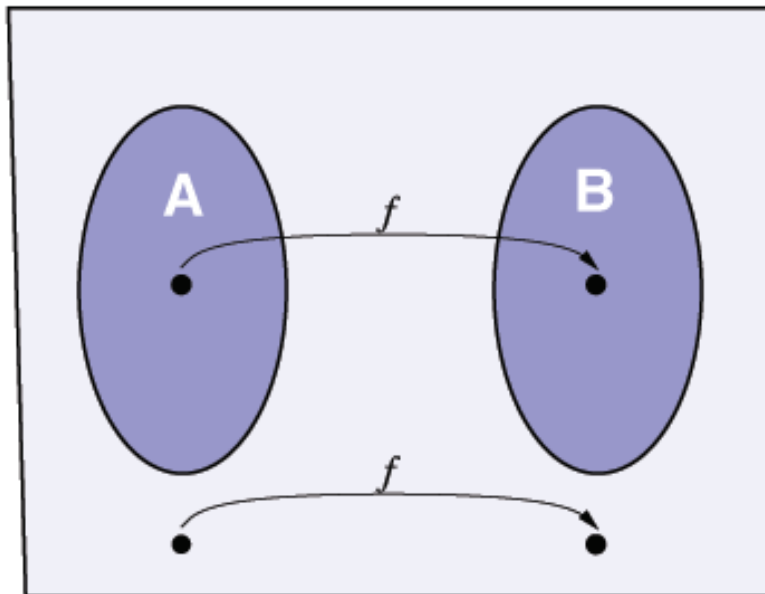
$$f : \Sigma^* \longrightarrow \Sigma^*$$

such that, for every w ,

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

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

Mapping Reductions



A mapping reduction converts questions about membership in A to membership in B

Notice that $A \leq_m B$ implies $\overline{A} \leq_m \overline{B}$.

Mapping Reductions: Reminders

Theorem 1:

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

Theorem 2 :

If $A \leq_m B$ and B is recursively enumerable, then A is recursively enumerable.

Mapping Reductions: Corollaries

Corollary 1: If $A \leq_m B$ and A is undecidable, then B is undecidable.

Corollary 2: If $A \leq_m B$ and A is not in \mathcal{RE} , then B is not in \mathcal{RE} .

Corollary 3: If $A \leq_m B$ and A is not in $co\mathcal{RE}$, then B is not in $co\mathcal{RE}$.

Mapping Reductions in General

Mapping reductions are applicable to wide areas in mathematics, not only computing. For example, consider the following two sets:

1. The set of **equations** of the form $Ax^2 + By + C$ where the coefficients are integers, that have a root consisting of **positive integers**.
2. The set of **knots** that can be untied (without tearing or breaking the rope) leaving at most ℓ **loops**.

Even though these two sets have very different nature, they are **reducible to each other** (by mapping reductions).

In this course we concentrate mainly on computing related problems, but reductions are relevant in much wider scopes.

Example: Halting

Recall that

$$A_{\text{TM}} = \{ \langle M, w \rangle \mid \text{TM } M \text{ accepts input } w \}$$

$$H_{\text{TM}} = \{ \langle M, w \rangle \mid \text{TM } M \text{ halts on input } w \}$$

Earlier today we proved that

- H_{TM} undecidable
- by (de facto) reduction from A_{TM} .

Let's reformulate this.

Example: Halting

Define a **computable function**, f :

- input of form $\langle M, w \rangle$
- output of form $\langle M', w' \rangle$
- where $\langle M, w \rangle \in A_{\text{TM}} \iff \langle M', w' \rangle \in H_{\text{TM}}$.

Example: Halting

The following machine computes this function f .

$F =$ on input $\langle M, w \rangle$:

- Construct the following machine M' .

M' : on input x

- run M on x
- If M accepts, *accept*.
- if M rejects, **enter a loop**.

- output $\langle M', w \rangle$

The function f is clearly computable. Let us convince ourselves (on board) it is also a reduction **from** A_{TM} **to**

H_{TM} .



TM Equality

Theorem: Both EQ_{TM} and its complement, $\overline{EQ_{TM}}$, are not enumerable. Stated differently, EQ_{TM} is neither enumerable nor co-enumerable.

- We show that A_{TM} is reducible to EQ_{TM} . The **same function** is also a mapping reduction from $\overline{A_{TM}}$ to $\overline{EQ_{TM}}$, and thus $\overline{EQ_{TM}}$ is **not enumerable**.
- We then show that A_{TM} is reducible to $\overline{EQ_{TM}}$. The **new function** is also a mapping reduction from $\overline{A_{TM}}$ to EQ_{TM} , and thus EQ_{TM} is **not enumerable**.

TM Equality

Claim: A_{TM} is reducible to $\overline{\text{EQ}_{\text{TM}}}$.

$f_1 : A_{\text{TM}} \longrightarrow \overline{\text{EQ}_{\text{TM}}}$ works as follows:

F_1 : On input $\langle M, w \rangle$

- Construct machine M_1 : on any input, **reject**.
- Construct machine M_2 : on input x , run M on w . If it accepts, **accept**.
- Output $\langle M_1, M_2 \rangle$.

TM Equality

F_1 : On input $\langle M, w \rangle$

- Construct machine M_1 : on any input, **reject**.
- Construct machine M_2 : on any input x , run M on w .
If it accepts, **accept** x .
- Output $\langle M_1, M_2 \rangle$.

Note

- M_1 accepts **nothing**
- if M accepts w then M_2 accepts **everything**,
and otherwise **nothing**.
- so $\langle M, w \rangle \in A_{\text{TM}} \iff \langle M_1, M_2 \rangle \in \overline{\text{EQ}_{\text{TM}}}$
- f_1 is clearly computable. Thus it is a reduction **from**
 A_{TM} **to** $\overline{\text{EQ}_{\text{TM}}}$. ♠

TM Equality

Claim: A_{TM} is reducible to EQ_{TM} .

$f_2 : A_{\text{TM}} \longrightarrow \text{EQ}_{\text{TM}}$ works as follows:

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

- Construct machine M_1 : on any input, *accept*.
- Construct machine M_2 : on any input x , run M on w .
If it accepts, *accept*.
- Output $\langle M_1, M_2 \rangle$.

TM Equality

F_2 : On input $\langle M, w \rangle$

- Construct machine M_1 : on any input, **accept**.
- Construct machine M_2 : on any input x , run M on w .
If it accepts, **accept**.
- Output $\langle M_1, M_2 \rangle$.

Note

- M_1 accepts **everything**
- if M accepts w , then M_2 accepts **everything**,
and otherwise **nothing**.
- $\langle M, w \rangle \in A_{\text{TM}} \iff \langle M_1, M_2 \rangle \in \text{EQ}_{\text{TM}}$.
- f_2 is clearly computable. Thus it is a reduction **from**
 A_{TM} **to** EQ_{TM} . ♣

Non Trivial Properties of \mathcal{RE} Languages

A few examples

- L is finite.
- L is infinite.
- L contains the empty string.
- L contains no prime number.
- L is co-finite.
- . . .

All these are **non-trivial** properties of enumerable languages, since for each of them there is $L_1, L_2 \in \mathcal{RE}$ such that L_1 satisfies the property but L_2 does not.

Are there any **trivial** properties of \mathcal{RE} languages?

Rice's Theorem

Theorem Let \mathcal{C} be a proper non-empty subset of the set of enumerable languages. Denote by $L_{\mathcal{C}}$ the set of all TMs encodings, $\langle M \rangle$, such that $L(M)$ is in \mathcal{C} . Then $L_{\mathcal{C}}$ is undecidable.

(See problem 5.22 in Sipser's book)

Proof by reduction from A_{TM} .

Given M and w , we will construct M_0 such that:

- If M accepts w , then $\langle M_0 \rangle \in L_{\mathcal{C}}$.
- If M does not accept w , then $\langle M_0 \rangle \notin L_{\mathcal{C}}$.

Rice's Theorem

- Without loss of generality, $\emptyset \notin \mathcal{C}$.
- (Otherwise, look at $\overline{\mathcal{C}}$, also proper and non-empty.)
- Since \mathcal{C} is not empty, there exists some language $L \in \mathcal{C}$. Let M_L be a TM accepting this language (recall \mathcal{C} contains only **recursively enumerable** languages).
- continued ...

Rice's Theorem

Given $\langle M, w \rangle$, construct M_0 such that:

- If M accepts w , then $L(M_0) = L \in \mathcal{C}$.
- If M does not accept w , then $L(M_0) = \emptyset \notin \mathcal{C}$.

M_0 on input y :

1. Run M on w .
2. If M accepts w , run M_L on y .
 - a. if M_L accepts, **accept**, and
 - b. if M_L rejects, **reject**.

Claim: The transformation $\langle M, w \rangle \rightarrow \langle M_0 \rangle$ is a mapping reduction from A_{TM} to $L_{\mathcal{C}}$.

Rice's Theorem

Proof: M_0 on input y :

1. Run M on w .
 2. If M accepts, run M_L on y .
 - a. if M_L accepts, **accept**, and
 - b. if M_L rejects, **reject**.
- The machine M_0 is simply a concatenation of two known TMs – the **universal machine**, and M_L .
 - Therefore the transformation $\langle M, w \rangle \rightarrow \langle M_0 \rangle$ is a computable function, defined for all strings in Σ^* .
 - (**But what do we actually do with strings not of the form $\langle M, w \rangle$?**)

Rice's Proof (Concluded)

- If $\langle M, w \rangle \in A_{\text{TM}}$ then M_0 gets to step 2, and runs M_L on y .
- In this case, $L(M_0) = L$, so $L(M_0) \in \mathcal{C}$.
- On the other hand, if $\langle M, w \rangle \notin A_{\text{TM}}$ then M_0 never gets to step 2.
- In this case, $L(M_0) = \emptyset$, so $L(M_0) \notin \mathcal{C}$.
- This establishes the fact that $\langle M, w \rangle \in A_{\text{TM}}$ iff $\langle M_0 \rangle \in L_{\mathcal{C}}$. So we have $A_{\text{TM}} \leq_m L_{\mathcal{C}}$, thus $L_{\mathcal{C}}$ is undecidable.



Rice's Theorem (Reflections)

- Rice's theorem can be used to show **undecidability** of properties like
 - “does $L(M)$ contain infinitely many primes”
 - “does $L(M)$ contain an arithmetic progression of length 15”
 - “is $L(M)$ empty”
- Decidability of properties related to the encoding itself cannot be inferred from Rice. For example “does $\langle M \rangle$ has an even number of states” is decidable.
- Properties like “does M reaches state q_6 on the empty input string” are undecidable, but this **does not** follow from Rice's theorem. Likewise, Rice does not say anything on membership in $\mathcal{P}\mathcal{S}$ of problems like “is $L(M)$ finite”

Reductions via Controlled Executions

Consider the language $L_{\text{infinite}} = \{\langle M \rangle \mid L(M) \text{ is infinite}\}$.

By Rice Theorem, this language is not in \mathcal{R} .

We want to show that $L_{\text{infinite}} \notin RE$.

Idea: Reduction from $\overline{H_{\text{TM}}}$.

So we are after a reduction $f : \langle M, w \rangle \mapsto \langle M_0 \rangle$ such that

- If M halts on w then $L(M_0)$ is **finite**.
- If M does not halts on w then $L(M_0)$ is **infinite**.

This looks a bit tricky...

Reductions via Controlled Executions (2)

We are after a reduction $f : \langle M, w \rangle \mapsto \langle M_0 \rangle$ such that

- If M halts on w then $L(M_0)$ is **finite**.
- If M does not halt on w then $L(M_0)$ is **infinite**.

Given $\langle M, w \rangle$, construct the TM M_0 as following:

- M_0 on input y
- Runs the universal machine U on $\langle M, w \rangle$ for $|y|$ steps.
- If U did **not** halt in that many steps, M_0 **accepts** y .
- If U **did halt** in that many steps, M_0 **rejects** y .

$f(\langle M, w \rangle) = M_0$. Let us examine $L(M_0)$.

(Remark: M_0 halts on all inputs.)

Reductions via Controlled Executions (3)

$f(\langle M, w \rangle) = M_0$. Let us examine $L(M_0)$.

- If M does **not** halt on w , then M_0 **accepts all** y , so $L(M_0) = \Sigma^*$, and thus $\langle M_0 \rangle \in \mathbf{L}_{\text{infinite}}$.
- If M does halt on w after k simulation steps, then M_0 **accepts only** y s of length smaller than k , so $L(M_0)$ is **finite**, and thus $\langle M_0 \rangle \notin \mathbf{L}_{\text{infinite}}$.

We have shown that $\overline{H_{\text{TM}}} \leq_m \mathbf{L}_{\text{infinite}}$.

Since $\overline{H_{\text{TM}}} \notin \mathcal{RE}$, this implies $\mathbf{L}_{\text{infinite}} \notin \mathcal{RE}$. ♠

\mathcal{RE} -Completeness

Question: Is there a language L that is **hardest** in the class \mathcal{RE} of enumerable languages (languages accepted by some TM)?

Answer: Well, you have to **define** what you mean by “hardest language”.

Definition: A language $L_0 \subseteq \Sigma^*$ is called **\mathcal{RE} -complete** if the following holds

- $L_0 \in \mathcal{RE}$ (membership).
- For **every** $L \in \mathcal{RE}$, $L \leq_m L_0$ (hardness).

\mathcal{RE} -Completeness

Definition A language $L_0 \subseteq \Sigma^*$ is called \mathcal{RE} -complete if the following holds

- $L_0 \in \mathcal{RE}$ (membership).
- For **every** $L \in \mathcal{RE}$, $L \leq_m L_0$ (hardness).

The second item means that for every enumerable L there is a mapping reduction f_L **from** L **to** L_0 . The reduction f_L depends on L and will typically differ from one language to another.

\mathcal{RE} -Completeness

Question: Having defined a reasonable notion, we should make sure it is not vacuous, namely verify there is at least one language satisfying it.

Theorem The language A_{TM} is \mathcal{RE} -Complete.

Proof:

- The universal machine U accepts the language A_{TM} , so $A_{\text{TM}} \in \mathcal{RE}$.
- Suppose L is in \mathcal{RE} , and let M_L be a TM accepting it. Then $f_L(w) = \langle M_L, w \rangle$ is a mapping reduction **from L to A_{TM}** (why?). ♣