Computer Science 351

Many-One Reductions

Instructor: Wayne Eberly

Department of Computer Science
University of Calgary

Lecture #17

Goals for Today

Goals for Today:

- Introduce ***many-one reductions***
- Explain how these can be used, both to prove that languages are ***undecidable*** and to prove that languages are ***unrecognizable***.

Many-One Reductions

Let Σ_1 and Σ_2 be two alphabets (possibly the same) and let $L_1 \subseteq \Sigma_1^\star$ and $L_2 \subseteq \Sigma_2^\star$ be two languages over these alphabets.

Definition: A ***many-one reduction*** from L_1 to L_2 is a ***total*** function

$$f : \Sigma_1^\star \to \Sigma_2^\star$$

such that the following properties are satisfied.

(a) For every string $\omega \in \Sigma_1^\star$, $\omega \in L_1$ *if and only if* $f(\omega) \in L_2$.

(b) The function f is computable.

We will say that L_1 is ***many-one reducible*** to L_2, and write

$$L_1 \preceq_{\mathrm{M}} L_2$$

if a many-one reduction from L_1 to L_2 exists.

Many-One Reductions

- A many-one reduction is like a ***signal converter***:

It is, effectively, converting an instance of *one* problem into an instance of *another* problem that has the same solution as the instance it was given.

Process Followed To Provide a Many-One Reduction

To prove that a language $L_1 \subseteq \Sigma_1^\star$ is many-one reducible to a language $L_2 \subseteq \Sigma_2^\star$,

1. Clearly and precisely describe the ***total*** function $f : \Sigma_1^\star \to \Sigma_2^\star$ that will be shown to be a "many-one reduction."
2. ***Prove*** that if $\omega \in L_1$ then $f(\omega) \in L_2$ for every string $\omega \in \Sigma_1^\star$.
3. ***Prove*** that if $\omega \notin L_1$ then $f(\omega) \notin L_2$ for every string $\omega \in \Sigma_1^\star$.

Process Followed To Provide a Many-One Reduction

4. ***Sketch a Proof*** that f is computable — including enough detail for it to be reasonably clear that you really *could* write a Python or Java program that computes this function from strings to strings.

   Go further, by describing a (multi-tape) implementation Turing machine that implements this algorithm, if you are not sure that it is easy to show that methods used by your program are computable.

Mistakes To Watch For and Avoid

- Giving a definition of f that is vague, ambiguous, or just-plain-unreadable.
- Defining a ***partial*** function from $\Sigma_1^\star$ to $\Sigma_2^\star$ (that is not defined for every string $\omega \in \Sigma_1^\star$) instead of a ***total function***.
- Forgetting about step 3, above — It is *not* sufficient just to show that if $\omega \in L_1$ then $f(\omega) \in L_2$.
- Failing to include enough detail at the end for it to be clear that your function f really ***is*** computable — sometimes because f has not been clearly defined and sometimes because it has, but f is not actually computable at all!

Example

Example Let $L, \widehat{L} \subsetneq \Sigma^\star$ for an alphabet Σ, such that $L \neq \Sigma^\star$, and suppose that $\widehat{L}$ is ***decidable***.

Claim: $L \cap \widehat{L} \preceq_{\mathrm{M}} L$.

Note: Since $L \neq \Sigma^\star$ there exists a string $\mu_{\mathrm{No}} \in \Sigma^\star$ such that $\mu_{\mathrm{No}} \notin L$.

Problem: Use the *process*, given above, to prove this claim.

Example

1. *Defining the function:*

   Let $f : \Sigma^\star \to \Sigma^\star$ such that, for all $\omega \in \Sigma^\star$,

$$f(\omega) = \begin{cases} \omega & \text{if } \omega \in \widehat{L}, \\ \mu_{\text{No}} & \text{if } \omega \notin \widehat{L}. \end{cases}$$

Example

2. *Proving that if* $\omega \in L \cap \widehat{L}$ *then* $f(\omega) \in L$*, for all* $\omega \in \Sigma^\star$*:*

   Let $\omega \in \Sigma^\star$ such that $\omega \in L \cap \widehat{L}$. Then

$$\begin{aligned} f(\omega) &= \omega && \text{(since } \omega \in L \cap \widehat{L}\text{, so that } \omega \in \widehat{L}\text{)} \\ &\in \widehat{L} \cap L && \text{(since it is given that } \omega \in \widehat{L} \cap L\text{)} \\ &\subseteq L. \end{aligned}$$

   Since ω was arbitrarily chosen from $\Sigma^\star$ such that $\omega \in L \cap \widehat{L}$ it follows that if $\omega \in L \cap \widehat{L}$ then $f(\omega) \in L$ *for all* $\omega \in \Sigma^\star$.

Example

3. *Proving that if* $\omega \notin L \cap \widehat{L}$ *then* $f(\omega) \notin L$*, for all* $\omega \in \Sigma^\star$*:*

   Let $\omega \in \Sigma^\star$ such that $\omega \notin L \cap \widehat{L}$. One of the following *cases* must arise:

   3(a) $\omega \in \widehat{L}$, but $\omega \notin L \cap \widehat{L}$ — so that $\omega \notin L$.

   3(b) $\omega \notin \widehat{L}$.

   These cases are considered separately, below.

Example

3(a) If $\omega \in \widehat{L}$, but $\omega \notin L \cap \widehat{L}$ (so that $\omega \notin L$) then

$$\begin{aligned} f(\omega) &= \omega && \text{(since } \omega \in \widehat{L}\text{)} \\ &\notin L && \text{(as noted above).} \end{aligned}$$

3(b) If $\omega \notin \widehat{L}$ then

$$\begin{aligned} f(\omega) &= \mu_{\text{No}} && \text{(since } \omega \notin \widehat{L}\text{)} \\ &\notin L && \text{(by the choice of } \mu_{\text{No}}\text{).} \end{aligned}$$

Thus $f(\omega) \notin L$ in every case. Since ω was arbitrarily chosen from $\Sigma^\star$ such that $\omega \notin L \cap \widehat{L}$ it follows that if $\omega \notin L \cap \widehat{L}$ then $f(\omega) \notin L$ *for all* $\omega \in \Sigma^\star$.

Example

4. *Sketch a proof that f is computable.*

   This is left as an ***exercise*** — which is taken up in a supplemental document for this lecture.

Decision Problems and Associated Languages

A ***decision problem*** is a computational problem that asks a question — so that it has a "Yes/No" answer.

Example:

Acceptance Problem

Instance: A Turing machine

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$$

and an input string $\omega \in \Sigma^\star$

Question: Does M accept ω?

Decision Problems and Associated Languages

When an *alphabet* Σ and an *encoding* — mapping instances of the problem to strings in $\Sigma^\star$ — is chosen, *three* languages can be associated with the decision problem:

- *Language of Instances:* The set of strings in $\Sigma^\star$ that are encodings of instances of the decision problem.
- *Language of Yes-Instances:* The set of strings in $\Sigma^\star$ that are encodings of instances of the decision problem for which the answer is “Yes”.
- *Language of No-Instances:* The set of strings in $\Sigma^\star$ that are encodings of instances of the decision problem for which the answer is “No”.

Decision Problems and Associated Languages

The Usual Situation:

- Every instance of the decision problem is encoded by some string in $\Sigma^\star$, and no string in $\Sigma^\star$ is an encoding for two or more instances of the decision problem.
- We are, generally, interested in decision problems whose languages of instances are ***decidable***.
- When we say that we are "reducing one decision problem to another" — *in this course* — we generally mean that we are giving a ***many-one reduction*** from the ***language of Yes-instances for the first decision problem*** to the ***language of Yes-instances of the second decision problem***.

Another Example

- An encoding scheme for the “Acceptance Problem”, using an alphabet Σ_{TM}, was introduced in the lecture material for *universal Turing machines*.
- For this encoding scheme, the *language of instances* of the Acceptance Problem is the language $L_{\mathsf{TM+I}} \subseteq \Sigma_{\mathsf{TM}}^{\star}$ — proved to be decidable in the same lecture material.
- For this encoding scheme, the *language of Yes-instances* of the Acceptance Problem is the language $A_{\mathsf{TM}} \subseteq \Sigma_{\mathsf{TM}}^{\star}$ — proved to be ***recognizable***, but also ***undecidable***, in the lecture material for universal Turing machines, once again.
- For this encoding scheme, the *language of No-instances* of the Acceptance Problem is the language $NA_{\mathsf{TM}} \subseteq \Sigma_{\mathsf{TM}}^{\star}$, — proved to be ***undecidable*** in the lecture material for *oracle reductions*.

Another Example

Consider another decision problem:

Halting Problem

Instance: A Turing machine

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$$

and an input string $\omega \in \Sigma^\star$

Question: Does M halt, when executed on input ω?

Another Example

- The same encoding scheme — again, using alphabet Σ_{TM} — can be used for the "Halting Problem" as for the "Acceptance Problem".
- Since the same set of instances is used, the *language of instances* of the Halting Problem is (once again) the language $L_{\mathsf{TM+I}} \subseteq \Sigma_{\mathsf{TM}}^{\star}$ that was proved to be decidable in the lecture material for universal Turing machines.
- The *language of Yes-instances* of the Halting Problem is the set $\mathit{Halt}_{\mathsf{TM}} \subseteq \Sigma_{\mathsf{TM}}^{\star}$ of encodings of Turing machines M, and input strings ω for M, such that M halts when executed on input ω.
- The *language of No-instances* of the Halting Problem is the set $\mathit{Loop}_{\mathsf{TM}} \subseteq \Sigma_{\mathsf{TM}}^{\star}$ of encodings of Turing machines M, and input strings ω for M, such that M loops when executed on input ω.

Another Example

Suppose we wish to wish to show that the "Halting Problem" is reducible to the "Acceptance Problem".

- In other words, we wish to show that

$$\mathit{Halt}_{\mathrm{TM}} \preceq_{\mathrm{M}} \mathit{A}_{\mathrm{TM}}.$$

- A revised process — suitable for (some) reductions between decision problems, will be given as this example is completed.
- ***The main new idea:*** When you can, start by working at the high level (working with decision problems instead of languages of instances); get things right at that level; and then add detail, as needed to prove a corresponding reduction from one *language* to another.

Another Example — High-Level Details

1. Start by describing a mapping, φ, from instances of the first decision problem to instances of the second decision problem — which maps Yes-instances of the first decision problem to Yes-instances of the second decision problem, and which maps No-instances of the first decision problem to No-instances of the second decision problem.

   This will —- eventually — be used to produce the function f from strings to strings that is used in the first version of the process, that was given before this.

Another Example — High-Level Details

- For our example, we need to find a mapping φ from Turing machines M and input strings ω (instances of the "Halting Problem") to Turing machines $\widehat{M}$ and input strings $\widehat{\omega}$ (instances of the "Acceptance Problem") such that

$$M \text{ halts, when executed on } \omega \iff \widehat{M} \text{ accepts } \widehat{\omega}.$$

Another Example — High-Level Details

- Continuing with this example, consider an instance of the "Halting Problem" that includes a Turing machine

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$$

and input string $\omega \in \Sigma^\star$. Suppose φ maps this to a Turing machine

$$\widehat{M} = (Q, \Sigma, \Gamma, \widehat{\delta}, q_0, q_{\text{accept}}, q_{\text{reject}})$$

— with the same set of states, start state, accepting state, rejecting state, input alphabet and tape alphabet, but with a slightly different transition function $\widehat{\delta}$ — and the same input string, so that $\widehat{\omega} = \omega$.

Another Example — High-Level Details

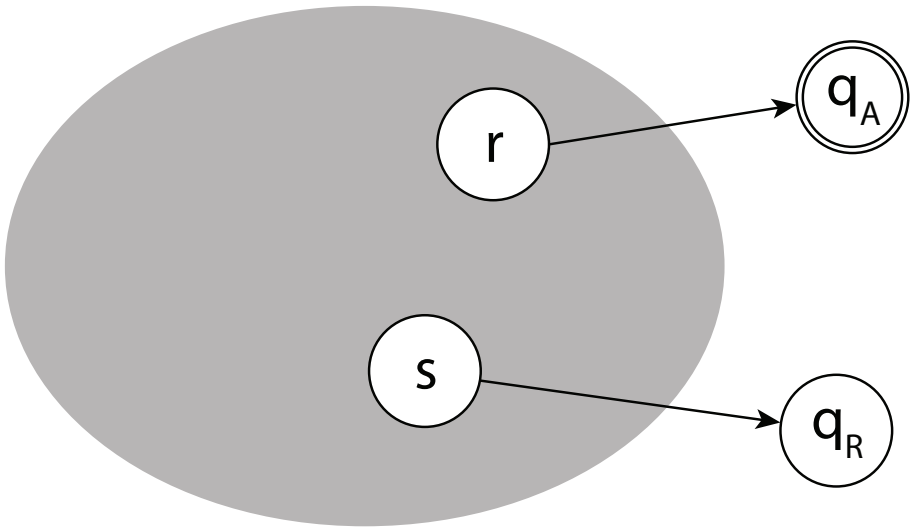
- In particular suppose that, for $q \in Q \setminus \{q_{\text{accept}}, q_{\text{reject}}\}$ and $\sigma \in \Gamma$,

$$\widehat{\delta}(q,\sigma) = \begin{cases} \delta(q,\sigma) & \text{if } \delta(q,\sigma) = (r,\tau,m) \\ & \text{where } r \neq q_{\text{reject}}, \\ (q_{\text{accept}},\tau,m) & \text{if } \delta(q,\sigma) = (q_{\text{reject}},\tau,m) \end{cases}$$

  where $r \in Q$, $\tau \in \Gamma$, and $m \in \{\mathtt{L},\mathtt{R}\}$.
- Thus transitions to the ***rejecting*** state are replaced with similar transitions to the ***accepting*** state in M_1, and everything else is the same.

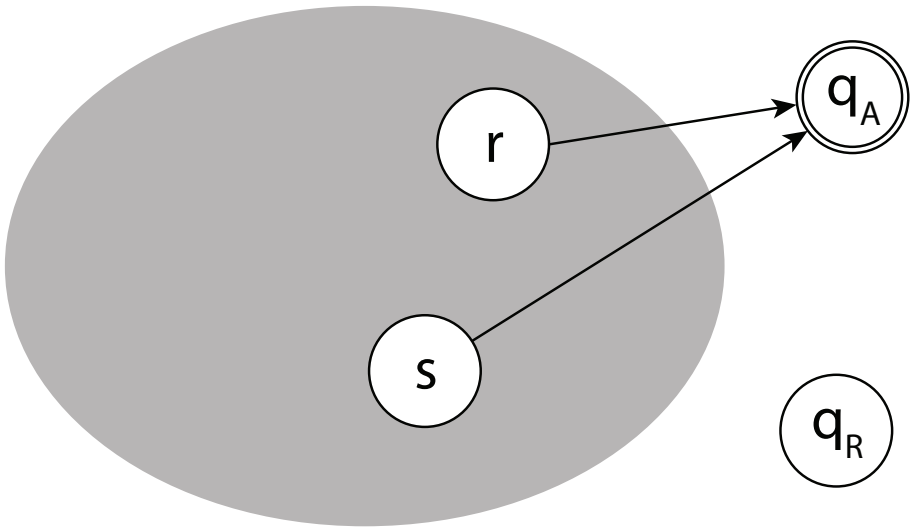
Another Example — High-Level Details

That is, if M looks like this...



Another Example — High-Level Details

...then $\widehat{M}$ looks like this, instead:



Another Example — High Level Details

2. ***Prove*** that the mapping φ maps each Yes-instance of the first problem to a Yes-instance of the second problem, and that φ maps each No-instance of the first problem to a No-instances of the second problem.

Another Example — High Level Details

- For this example it is necessary to show that if M, ω and $\widehat{M}$ are as above then if M halts when executed on ω then $\widehat{M}$ accepts ω — and if M loops when executed on ω then $\widehat{M}$ *does not* accept ω.
- The following is easily proved by induction on k: If M's execution on ω uses at least $k+1$ steps, for $k \in \mathbb{N}$, then so does $\widehat{M}$'s execution on ω — and M and $\widehat{M}$ are in the same configuration after the first k of these steps have been taken.

  Exercise: Prove the above claim.

Another Example — High Level Details

- Suppose that M halts when executed on input ω. Then either M accepts ω or M rejects ω after $k + 1$ steps for some $k \in \mathbb{N}$.

  The result on the previous slide — and a consideration of $\widehat{M}$'s transition function, can be used to show that $\widehat{M}$ *accepts* ω after $k + 1$ steps.

- Suppose, instead, that M loops when executed on input ω. Then M takes at least $k + 1$ steps, when executed on input ω, for all $k \in \mathbb{N}$.

  The result on the previous slide implies that $\widehat{M}$ takes at least $k + 1$ steps, when executed on ω, for all $k \in \mathbb{N}$ as well — so that $\widehat{M}$ loops on ω too.

  Thus $\widehat{M}$ *does not* accept ω in this case.

Another Example — Introducing Encodings, and Handling a Special Case

3. Now consider encoding schemes: One, e_1, maps instances of the first problem to strings over an alphabet Σ_1, and another, e_2, maps instances of the second problem to strings over an alphabet Σ_2.

   For a tutorial exercise, or problem on an assignment or test, these encoding schemes will be given for you. Under other circumstances you may need to choose and describe them yourself.

Another Example — Introducing Encodings, and Handling a Special Case

- For this example, e_1 and e_2 are both the encoding scheme for Turing machines and input strings described in the lecture material for universal Turing machines — and using alphabet $\Sigma_1 = \Sigma_2 = \Sigma_{\text{TM}}$.

Another Example — Introducing Encodings, and Handling a Special Case

4. Let $L_{I_1} \subseteq \Sigma_1^\star$ be the language of encodings of instances of the first problem, and let $L_{I_2} \subseteq \Sigma_2^\star$ be the language of encodings of instances of the second problem.

   Check (and prove, if it is not obvious) that the following ***Usual Situation*** holds: L_{I_1} is decidable, and $L_{I_2} \neq \Sigma_2^\star$ — so that there exists a string $\mu_{\text{Junk}} \in \Sigma_2^\star$ such that $\mu_{\text{Junk}} \notin L_{I_2}$.

   If this "usual situation" holds then it will be necessary to choose the string μ_{Junk} in order to continue.

   Unfortunately, if this "usual situation" then this complicates things and you cannot apply the rest of this process to finish.

Another Example — Introducing Encodings, and Handling a Special Case

- For this example, $L_{I_1} = L_{I_2} = L_{\text{TM+I}}$ — a language that was shown to be decidable (as desired) in the lecture material for universal Turing machines.
- $L_{\text{TM+I}} \neq \Sigma^{\star}_{\text{TM}}$, so that the "usual situation" holds — and we can set μ_{Junk} to be the empty string, λ, since $\lambda \notin L_{\text{TM+I}}$.

Another Example — Introducing Encodings, and Handling a Special Case

5. Let $f : \Sigma_1^\star \to \Sigma_2^\star$ be defined as follows for all $\mu \in \Sigma_1^\star$:
   - If $\mu \in L_{I_1}$ — so that μ is the encoding of some instance, α, of the first problem, then set $f(\mu)$ to be the encoding of the correspondence instance $\varphi(\alpha)$ of the second problem — where φ is the mapping from instances of the first problem to instances of the second problem, chosen at the beginning of this process.
   - If $\mu \notin L_{I_2}$ then set $f(\mu)$ to be μ_{Junk} — for the string $\mu_{\text{Junk}} \in \Sigma_2^\star$ chosen during step 4, above.

Another Example — Introducing Encodings, and Handling a Special Case

Exercise:

- Let $L_{\text{Yes},1} \subseteq \Sigma_1^\star$ the language of Yes-instances of the first problem and let $L_{\text{Yes},2} \subseteq \Sigma_2^\star$ be the language of Yes-instances of the second problem.

  Prove that

  $$\mu \in L_{\text{Yes},1} \iff f(\mu) \in L_{\text{Yes},2}$$

  for every string $\mu \in \Sigma_1^\star$.

Another Example — Introducing Encodings, and Handling a Special Case

- For the example, $L_{I_1} = L_{I_2} = L_{\text{TM+I}}$ and $\mu_{\text{Junk}} = \lambda$, so $f : \Sigma^{\star}_{\text{TM}} \to \Sigma^{\star}_{\text{TM}}$ is a function such that the following properties are satisfied for every string $\mu \in \Sigma^{\star}_{\text{TM}}$:
  - If $\mu \in L_{\text{TM+I}}$, so that μ is the encoding of a Turing machine

    $$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$$

    and string $\omega \in \Sigma^{\star}$, then $f(\mu)$ is the encoding of the corresponding Turing machine

    $$\widehat{M} = (Q, \Sigma, \Gamma, \widehat{\delta}, q_0, q_{\text{accept}}, q_{\text{reject}})$$

    and the same string $\omega \in \Sigma^{\star}$ as described in Step #1.
  - If $\mu \in L_{\text{TM+I}}$ then (since μ_{Junk} was chosen to be λ, for this example) $f(\mu) = \lambda$.

Another Example — Completion of the Process

6. Give an algorithm to compute the function f and prove that it is correct.

   It is often advisable to start at the high level, using the mapping φ and, and the encoding schemes for problems, to give an algorithm to compute f that can be described using pseudocode.

   Details can be added, as needed, to confirm that there is a multi-tape Turing that computes the function f, as well.

- This is carried out, in order to complete this example, in supplemental material for this lecture.

Closure Properties

Proofs of the following claims are given in a supplemental document for this lecture.

Claim: #1 Suppose that $L_1 \subseteq \Sigma_1^\star$ and $L_2 \subseteq \Sigma_2^\star$ (for alphabets Σ_1 and Σ_2) are languages such that $L_1 \preceq_{\mathrm{M}} L_2$. If L_2 is decidable then L_1 is decidable too.

Claim #2: Suppose that $L_1 \subseteq \Sigma_1^\star$ and $L_2 \subseteq \Sigma_2^\star$ (for alphabets Σ_1 and Σ_2) are languages such that $L_1 \preceq_{\mathrm{M}} L_2$. If L_2 is recognizable then L_1 is recognizable too.

Closure Properties

The following are “corollaries” of Claim #3 and of Claim #4, respectively.

Corollary #3: Suppose that $L_1 \subseteq \Sigma_1^\star$ and $L_2 \subseteq \Sigma_2^\star$ (for alphabets Σ_1 and Σ_2) are languages such that $L_1 \preceq_{\mathrm{M}} L_2$. If L_1 is undecidable then L_2 is undecidable too.

Corollary #4: Suppose that $L_1 \subseteq \Sigma_1^\star$ and $L_2 \subseteq \Sigma_2^\star$ (for alphabets Σ_1 and Σ_2) are languages such that $L_1 \preceq_{\mathrm{M}} L_2$. If L_1 is unrecognizable then L_2 is unrecognizable too.

Another Way to Prove Undecidability

Another process to prove that a language $L \subseteq \Sigma^\star$ is undecidable:

- Choose another language $\widehat{L} \subseteq \widehat{\Sigma}^\star$ (over some alphabet $\widehat{\Sigma}$) such that $\widehat{L}$ is undecidable.
- Prove that $\widehat{L} \preceq_{\mathrm{M}} L$.
- Conclude, by Corollary #3, above, that L must be undecidable too.

A Way to Prove Unrecognizability

A process to prove that a language $L \subseteq \Sigma^\star$ is unrecognizable:

- Choose another language $\widehat{L} \subseteq \widehat{\Sigma}^\star$ (over some alphabet $\widehat{\Sigma}$) such that $\widehat{L}$ is unrecognizable.
- Prove that $\widehat{L} \preceq_{\mathrm{M}} L$.
- Conclude, by Corollary #4, above, that L must be unrecognizable too.

A Relationship Between Reducibilities

Proofs of the following claims are given in a supplemental document for this lecture.

Claim #5: The set of many-one reductions forms a reducibility.

Claim #6: Let $L_1 \subseteq \Sigma_1^\star$ and let $L_2 \subseteq \Sigma_2^\star$. If $L_1 \preceq_{\mathrm{M}} L_2$ then $L_1 \preceq_{\mathrm{O}} L_2$.

Claim #7: There exist languages $L_1 \subseteq \Sigma_1^\star$ and $L_2 \subseteq \Sigma_2^\star$ (for alphabets Σ_1 and Σ_2) such that $L_1 \preceq_{\mathrm{O}} L_2$ but $L_1 \not\preceq_{\mathrm{M}} L_2$.