Lecture #9: Many-One Reductions
Proofs of Undecidability — Examples II

About This Document

Sometimes, one example is not enough. This document provides another "many-one reduction", resembling those that you might be asked to give when solving a problem on an assignment. Indeed, it is somewhat more complicated than something that might be required for an assignment — the mappings you will need to describe will generally be *simpler* than the one that is used here.

Two New Problems

Consider the following decision problems.

The "Regular Language" Problem
Instance: A Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$.
Question: Is the language $L(M)$ of M a regular language?

Let us call the language of Yes-instances of this problem $\text{Regular}_{\text{TM}} \subseteq \Sigma^\star_{\text{TM}}$ — noting that (if we use the same encodings of Turing machines that we have used in previous lectures) $\text{Regular}_{\text{TM}}$ is a subset of the *decidable* language L_{TM} of encodings of Turing machines.

The "Nonregular Language" Problem
Instance: A Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$.
Question: Is the language $L(M)$ of M ***not*** a regular language?

Let us call the language of Yes-instances of this problem $\text{Nonregular}_{\text{TM}} \subseteq \Sigma^\star_{\text{TM}}$ — noting that $\text{Nonregular}_{\text{TM}} \subseteq L_{\text{TM}}$ as well.

Proving that the "Nonregular Language" Problem is Undecidable

Solution at a High Level

In order to prove that the "Nonegular Language" problem is undecidable, let us give a many-one reduction from the "Acceptance" problem to the "Nonegular Language" problem. In other words, let us show that $\mathsf{A}_{\mathsf{TM}} \preceq_{\mathsf{M}} \mathsf{Nonregular}_{\mathsf{TM}}$.

To begin, we must describe a mapping φ from Turing machines

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

and input strings $\omega \in \Sigma^\star$ — instances of the "Acceptance" problem — to Turing machines

$$\widehat{M} = (\widehat{Q}, \widehat{\Sigma}, \widehat{\Gamma}, \widehat{q}_0, \widehat{q}_{\mathsf{accept}}, \widehat{q}_{\mathsf{reject}})$$

— instances of the "Nonregular Language' problem — such that, for every Turing machine M and input string ω for M, if $\widehat{M} = \varphi((M, \omega))$, then

$$M \text{ accepts } \omega \iff L(\widehat{M}) \text{ is not a regular language.}$$

Now let $\Sigma_2 = \{\mathtt{a}, \mathtt{b}\}$ and consider the language

$$L_{\mathsf{Non}} = \{\mu \in \Sigma_2^\star \mid \text{the number of a's in } \mu \text{ is equal to the number of b's in } \mu\}.$$

Recall, from the lecture presentation for Lecture #5, that L_{Non} is not a regular language. Furthermore, a Turing machine M_{Non}, whose input alphabet is Σ_2 and whose language is L_{Non}, is designed on the preparatory material for Lecture #6.

Suppose, now, that the above Turing machine, $\widehat{M}$, is a Turing machine that implements the algorithm shown in Figure 1 on page 3. Note that even though ***the algorithm's input is a string $\zeta \in \Sigma_2^\star$, the first step uses the instance of the "Acceptance" problem that is used to define $\widehat{M}$, instead***: For a fixed instance $\widehat{M}$ and ω of the "Acceptance" problem, ***this algorithm does the same thing for every input string $\zeta \in \Sigma_2^\star$.***

Claim 1. *If M accepts ω then the language of $\widehat{M}$ is not a regular language.*

Proof. Suppose M accepts ω, and consider an execution of the algorithm in Figure 1 (which the Turing machine $\widehat{M}$ implements).

- Since M accepts ω, the simulation of M on input ω, at line 1 halts — so that the step at line 2 is reached and executed.

On input $\zeta \in \Sigma_2^\star$ {

1. Use a universal Turing machine to simulate the execution of M on input ω — *noting that this simulation might never end, so that the next step might never be reached*.
2. `if` (the simulation indicated that M accepts ω) {
3. Execute the Turing machine M_{Non} on the input ζ
4. `if` (M_{Non} accepts ζ) {
5. `accept` ζ

`}` `else` `{` // Only reached if M_{Non} rejected ζ

6. `reject` ζ

`}`

`}` `else` `{` // Only reached if M rejected ω

7. `reject` ζ

`}`

`}`

Figure 1: Algorithm to be Implemented by Turing Machine $\widehat{M}$

- Furthermore, since M accepts ω the test at line 2 is passed, so that the step at line 3 is reached and executed.
- One can now see, by an examination of the steps at lines 4, 5 and 6, that $\widehat{M}$ accepts ζ if and only if M_{Non} accepts ζ — so the language of $\widehat{M}$ is the same as the language of M_{Non} in this case.

Since the language, L_{Non}, of M_{Non} is not a regular language, this establishes the claim. □

Claim 2. *If M does not accept ω then the language of $\widehat{M}$ is a regular language.*

Proof. Suppose that M does not accept ω. Then either M rejects ω or M loops on ω. Consider an execution of $\widehat{M}$ on an input string $\zeta \in \Sigma_2^\star$ in each case.

- If M rejects ω then, during an execution of the algorithm in Figure 1 on an input string $\zeta \in \Sigma_2^\star$, the simulation of $\widehat{M}$ on the string ω halts, and the test at line 2 is reached. Since $\widehat{M}$ rejects ω the test at line 2 fails, so the step at line 7 is reached and executed: $\widehat{M}$ rejects ζ (so $\widehat{M}$ does not accept ζ in this case).

- On the other hand, if M loops on ω then, during an execution of the algorithm in Figure 1 on an input string $\zeta \in \Sigma_2^\star$, the simulation of $\widehat{M}$ on the string ω never halts, so that $\widehat{M}$ loops on ζ (so $\widehat{M}$ does not accept ζ in this case, either).

Since ζ was arbitrarily chosen from $\Sigma_2^\star$, it follows that $\widehat{M}$ does not accept *any* string in $\Sigma_2^\star$ — so the language of $\widehat{M}$ is the empty language, $\emptyset$. This is certainly a regular language, as needed to establish the claim. □

While this is a start, it is not sufficient: We must define the mapping, φ, from the instances of the "Acceptance" Problem to instances of the "Nonregular Language" Problem (taking us from an encoding of M and ω to an encoding of a corresponding Turing machine $\widehat{M}$) more precisely, complete this to obtain a total function from $\Sigma_{\text{TM}}^\star$ to $\Sigma_{\text{TM}}^\star$, and then show that this function is ***computable***. These are considered next.

Adding Implementation-Level Details — Needed to Define the Mapping More Precisely

Consider the Turing machine

$$\widehat{M} = (\widehat{Q}, \widehat{\Sigma}, \widehat{\Gamma}, \widehat{q}_0, \widehat{q}_{\text{accept}}, \widehat{q}_{\text{reject}}),$$

that is to be computed from an instance of the "Acceptance" problem including a Turing machine

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

and an input string $\omega \in \Sigma^\star$, in more detail. As discussed above, $\widehat{M}$ is to implement the algorithm shown in Figure 1.

- As shown in Figure 1 the algorithm's inputs are strings in $\Sigma_2^\star$. Thus $\widehat{M}$'s input alphabet is $\widehat{\Sigma} = \Sigma_2 = \{\mathtt{a}, \mathtt{b}\}$. When considering encodings of transitions and input strings later on, we will consider "a" to be the symbol σ_1 and we will consider "b" to be the symbol σ_2; the blank, symbol, $\sqcup$, must be considered to be σ_0.

One can ideally see, from an examination of the algorithm in Figure 1, that it is necessary to carry out the first major step without using the input string, $\zeta \in \Sigma_2^\star$ at all — because this first step makes use of the Turing machine M and input string ω, for M, that are part of the instance of the "Acceptance" problem" that is being used to *define* $\widehat{M}$.

The first step of the algorithm in Figure 1 will be carried out using a Turing machine

$$M_{\text{UTM}} = (Q_{\text{UTM}}, \Sigma_{\text{TM}}, \Gamma_{\text{UTM}}, \delta_{\text{UTM}}, q_{0,\text{UTM}}, q_{\text{accept,UTM}}, q_{\text{reject,UTM}})$$

that is based on — but not identical to — the ***Universal Turing Machine*** that is introduced in Lecture #8.

The third step of the algorithm (reached if the initial configuration ends, with M accepting ω) is an execution of the Turing machine M_{Non} described in Lecture #6 — a Turing machine whose tape alphabet includes "$\sqcup$", the symbols $\mathtt{a}, \mathtt{b} \in \Sigma_2$, and another pair of tape symbols, "$\mathtt{x}$" and "$\mathtt{X}$".

Now, in order to implement the algorithm in Figure 1, let us consider a Turing machine $\widehat{M}$ whose tape alphabet, $\widehat{\Gamma}$, includes the following symbols:

- "$\sqcup$" (which will be denoted as σ_0), and the input symbols "$\mathtt{a}$" (which will be denoted as σ_1) and "$\mathtt{b}$" (which will be denoted as σ_2).
- A *new* symbol, "$", which will be used to mark the leftmost cell of $\widehat{M}$'s tape at various parts of $\widehat{M}$'s computation; this will denoted σ_3.
- The additional symbols, "$\mathtt{x}$" and "$\mathtt{X}$", that are in M_{Non}'s tape alphabet. These will be denoted as σ_4 and σ_5, respectively.
- Ordered pairs of symbols — that is, elements of $\Gamma_{\text{UTM}} \times \{\mathtt{a}, \mathtt{b}, \sqcup\}$ where, as above, Γ_{UTM} is the tape alphabet for the Turing machine M_{UTM}. In what follows, if $\alpha \in \Gamma_{\text{UTM}}$ and $\beta \in \{\mathtt{a}, \mathtt{b}, \sqcup\}$, then the ordered pair (α, β) will also be shown (in pictures) as

Note that — while its size depends on the size of the tape alphabet, Γ_{UTM} of the (incompletely described) universal Turing machine M_{UTM}, $\widehat{\Gamma}$ is a ***fixed*** tape alphabet that does not depend on the instance of the "Acceptance" problem that has been given here, at all: It is the same for all of them.

The Turing machine, $\widehat{M}$, can now be viewed as consisting of the following *components*.

1. A first component, M_{Init}, changes the tape contents from the given input string to a representation of both this string and the string $\mu \in \Sigma^{\star}_{\text{TM}}$ encoding the given instance of the "Acceptance" problem (that is, the instance including the above Turing machine M and input string ω). Suppose, in particular, that the input string is

   $$\zeta = \alpha_1 \alpha_2 \ldots \alpha_n \in \Sigma_2^{\star}$$

   — and that the string, encoding the instance of the "Acceptance" problem, is

   $$\mu = \beta_1, \beta_2, \ldots, \beta_m \in \Sigma^{\star}_{\text{TM}}$$

Let $k = \max(n, m)$ and let $\alpha_i = \sqcup$ for $n + 1 \leq i \leq k$ (if $m > n$) and $\beta_i = \sqcup$ for $m + 1 \leq i \leq k$ (if $m < n$) — so that $\alpha_i \in \Sigma_{\mathsf{TM}}$ and $\beta_i \in \{\mathtt{a}, \mathtt{b}, \sqcup\}$ for $1 \leq i \leq n$. Then the goal of M_{init} is to change tape's contents from initial contents for input ζ,

α_1	α_2	$\cdots$	α_n	$\sqcup$	$\sqcup$	$\cdots$

(1)

to contents that have the leftmost cell marked (by storing "\$") and that use symbols in $\Gamma_{\mathsf{UTM}} \times \{\mathtt{a}, \mathtt{b}, \sqcup\}$ to store ω on a top track and ζ on a bottom track of what is a "multi-track" part of a tape:

\$	β_1 / α_1	β_2 / α_2	$\cdots$	β_k / α_k	$\sqcup$	$\sqcup$	$\cdots$

(2)

This Turing machine both starts and ends the computation with the tape head at the leftmost cell of the tape.

Exercise: Design the component M_{init} — noting, here, that μ should be considered to be a *fixed string* (that you know, and that can be "hard-wired" into the transition function for this component) — while ζ should be considered to be an input string (that is, as usual, not known when the Turing machine is being designed). You should find that your component has approximately (but, probably, slightly more than) $3m$ states.

Then confirm that an *encoding* of M_{Init} (encoded as a string in $\Sigma^{\star}_{\mathsf{TM}}$ in the usual way) can be computed using an algorithm that has the above string $\mu \in \Sigma^{\star}_{\mathsf{TM}}$ as its input.

2. The next component is based on the ***universal Turing machine*** introduced in Lecture #8. However, this is modified so that it works with the "multi-track" representation described above — so that it can be used when "\$" marks the leftmost cell and only uses the bottom track of its tape, extending the non-blank part of the tape by placing a copy of "$\sqcup$" with

$\sqcup$
$\sqcup$

when it needs to. Furthermore, this component — which we will call M_{UTM} ***cleans up after itself***, essentially as described in Tutorial Exercise #7: If the execution of this component that starts (in its initial state, with the tape head at the leftmost cell) with the tape contents as shown at line (2), above — and this execution eventually halts — then, when the execution halts, then the contents of the tape are as follows.

\$	$\sqcup$ / α_1	$\sqcup$ / α_2	$\cdots$	$\sqcup$ / α_n	$\sqcup$	$\sqcup$	$\cdots$

(3)

Exercise: Reviewing course information, as needed, confirm that a component M_{UTM}, that carries out this computation, exists. Note that this is *fixed* Turing machine that does not depend on the string, μ, that encodes the given instance of the "Acceptance" problem, at all.

3. A third component, M_{Restore}, is used to restore the Turing machine's initial tape contents. That is, it begins with with the tape as shown at line (3) and ends with the contents as shown at line (1), once again (with the tape head at the leftmost cell, in each case).

   Exercise: Confirm that this component also exists (and is easily described). Once again, it is a *fixed* Turing machine (with a very small number of states) that does not depend on the string μ.

4. The fourth (and final) component is the Turing machine M_{Non}, described above (with transitions added for symbols in $\widehat{\Gamma}$ that are not in M_{Non}'s tape alphabet. Once again, this is a *fixed* Turing machine that does not depend on μ.

Exercise: After confirming that encodings of each of the above components M_{Init}, M_{UTM}, M_{Restore} and M_{Non} are each computable from μ (indeed, the last three are fixed strings that do not depend on μ, at all), prove that there exists a ***computable*** partial function

$$g : L_{\text{TM+I}} \to L_{\text{TM}} \tag{4}$$

which maps strings in $L_{\text{TM+I}}$, encoding instances of the "Acceptance" problem including a Turing machine M and input string ω, to encodings of the corresponding Turing machine $\widehat{M}$ that implements the algorithm in Figure 1.

Note that the set, $\widehat{Q}$ of $\widehat{M}$ should include the non-halting states of each of the four components, along with an accepting and a rejecting state. An encoding of the transition function of $\widehat{M}$ can be obtained as the concatenation of the encodings of the transition functions of the components (with leading brackets, and separating commas, inserted or removed as needed) — with indices of states adjusted as needed, so that the set of states of $\widehat{M}$ is a set with the usual form

$$\widehat{Q} = \{q_0, q_1, \ldots, q_s, q_{\text{accept}}, q_{\text{reject}}\}$$

for a positive integer s.

Completion of the Proof that the "Nonregular Language" Problem is Undecidable

Let $x_{\text{Junk}} = \lambda$ — then $x_{\text{Junk}} \notin L_{\text{TM}}$ and, since $\text{Nonregular}_{\text{TM}} \subseteq L_{\text{TM}}$, this ensures that $x_{\text{Junk}} \notin \text{Nonregular}_{\text{TM}}$ as well. Let $f : \Sigma^\star_{\text{TM}} \to \Sigma^\star_{\text{TM}}$ such that, for all $\mu \in \Sigma^\star_{\text{TM}}$,

$$f(\mu) = \begin{cases} g(\mu) & \text{if } \mu \in L_{\text{TM+I}}, \\ x_{\text{Junk}} & \text{if } \mu \in L_{\text{TM+I}} \end{cases} \tag{5}$$

where $g : L_{\text{TM+I}} \to L_{\text{TM+I}}$ is as defined at line (4), above. Then f is a total function from $\Sigma^\star_{\text{TM}}$ to $\Sigma^\star_{\text{TM}}$.

Claim 3. *Let $\mu \in \Sigma^\star_{\mathsf{TM}}$. If $\mu \in \mathsf{A}_{\mathsf{TM}}$ then $f(\mu) \in \mathsf{Nonregular}_{\mathsf{TM}}$.*

Proof. Let $\mu \in \Sigma^\star_{\mathsf{TM}}$ such that $\mu \in \mathsf{A}_{\mathsf{TM}}$. Then μ encodes a Turing machine

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

and a string $\omega \in \Sigma^\star$ such that M accepts ω. Now, since $\mu \in \mathsf{A}_{\mathsf{TM}} \subseteq L_{\mathsf{TM+I}}$ it follows by the definition of f at line (5), above, that $f(\mu) = g(\mu)$ for the partial function, g, described above — so that $g(\mu)$ is the encoding of a Turing machine $\widehat{M}$ implements the algorithm shown in Figure 1. Now, since M accepts ω, it follows by Claim 1 that the language of $\widehat{M}$ is not a regular language — so that its encoding, $g(\mu)$, is in $\mathsf{Nonregular}_{\mathsf{TM}}$.

Thus $f(\mu) = g(\mu) \in \mathsf{Nonregular}_{\mathsf{TM}}$ and, since μ was arbitrarily chosen from A_{TM}, this establishes the claim. □

Claim 4. *Let $\mu \in \Sigma^\star_{\mathsf{TM}}$. If $\mu \notin \mathsf{A}_{\mathsf{TM}}$ then $f(\mu) \notin \mathsf{Nonregular}_{\mathsf{TM}}$.*

Proof. Let $\mu \in \Sigma^\star_{\mathsf{TM}}$ such that $\mu \notin \mathsf{A}_{\mathsf{TM}}$. Then either $\mu \in L_{\mathsf{TM+I}}$ but $\mu \notin \mathsf{A}_{\mathsf{TM}}$, or $\mu \notin L_{\mathsf{TM+I}}$. These cases are considered separately, below.

- If $\mu \in L_{\mathsf{TM+I}}$ but $\mu \notin \mathsf{A}_{\mathsf{TM}}$ then μ encodes a Turing machine

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

  and a string $\omega \in \Sigma^\star$ such that M *does not* accept μ. Now, since $\mu \in L_{\mathsf{TM+I}}$, it follows by the definition of f at line (5), above, that $f(\mu) = g(\mu)$ for the partial function, g, described above — so that $g(\mu)$ is the encoding of a Turing machine $\widehat{M}$ that implements the algorithm shown in Figure 1. Now, since M does not accept ω, it follows by Claim 2 that the language of $\widehat{M}$ is a regular language – so that its encoding, $g(\mu)$ is not in $\mathsf{Nonregular}_{\mathsf{TM}}$. Thus $f(\mu) \notin \mathsf{Nonregular}_{\mathsf{TM}}$ in this case.
- On the other hand, if $\mu \notin L_{\mathsf{TM+I}}$, then it follows by the definition of f at line (5), above, that $f(\mu) = x_{\mathsf{Junk}}$ — so that $f(\mu) \notin L_{\mathsf{TM}}$. Since $\mathsf{Nonregular}_{\mathsf{TM}} \subseteq L_{\mathsf{TM}}$ it follows that $f(\mu) \notin \mathsf{Nonregular}_{\mathsf{TM}}$ in this case as well.

Thus $f(\mu) \notin M_{\mathsf{Nonregular}}$ in both of the cases that might arise. Since μ was arbitrarily chosen from $\Sigma^\star_{\mathsf{TM}}$ such that $\mu \notin \mathsf{A}_{\mathsf{TM}}$, this holds for all such strings, as needed to establish the claim. □

Claim 5. *The above function, f, is a computable total function from $\Sigma^\star_{\mathsf{TM}}$ to $\Sigma^\star_{\mathsf{TM}}$.*

Proof. It suffices to describe a multi-tape Turing machine that computes the function f in order to prove this claim. With that noted, consider a two-tape Turing machine that has the following components:

- A one-tape Turing machine — which will work with the second tape for the Turing machine (computing f) now being described — that decides membership in $L_{\text{TM+I}}$. Furthermore, we may assume that this is a Turing machine that cleans up after itself — so that second tape is filled with copies of "$\sqcup$" and the tape head for the second tape is at the leftmost cell of the tape when an execution of this component ends.
- A one-tape Turing machine — which will also work with the second tape for the Turing machine (computing f) now being described — that computes the partial function g described above.

Suppose that the Turing machine carries out the following steps.

1. Copy the input string, μ, onto the second tape (without changing the contents of the first tape), moving both tape heads back to the leftmost cells of the tape.
2. Apply the component described above, which decides $L_{\text{TM+I}}$, to decide whether $\mu \in L_{\text{TM+I}}$. When this step ends the first tape will still store the input string, the second tape will be filled with copies of "$\sqcup$", and both tape heads will be at the leftmost cells of the tape.
3. The details of the final step depend on whether it was decided that $\mu \in L_{\text{TM+I}}$ (or not).
    - If it was decided that $\mu \in L_{\text{TM+I}}$ then the input string should be copied onto the second tape, once again, with the tape heads moved back to the leftmost cells — but, this time, the copy of the input on the first tape should be erased (replacing every non-blank symbol with "$\sqcup$" when sweeping back to the left). The component that computes the partial function g should be applied — so that the desired string, $f(\mu) = g(\mu)$, is on the second tape (with the tape head back at the leftmost cell) when the computation ends, in this case.
    - On the other hand, if it was decided that $\mu \notin L_{\text{TM+I}}$, then the input string should be erased on the first tape, with the tape head for the first tape moved back to the leftmost cell. The fixed string x_{Junk} should then be written onto the second tape, with the tape head for the second tape moved back to the leftmost cell — so that $f(\mu)$ has been written onto the second tape in this case as well.

Each of the above steps is simple enough for a Turing machine to be able to carry out — and this computation ends (in every case) with the first tape filled with copies of "$\sqcup$", the second tape storing $f(\mu)$, with copies of "$\sqcup$" to the right, and with both tape heads at the leftmost cell of the tape. Thus this Turing machine computes the function f, as needed to establish claim. □

```
On input μ ∈ Σ*_TM {
1.  if (μ ∈ L_TM) {
2.    if (μ ∈ Regular_TM) {
3.      reject μ
      } else {
4.      accept μ
      }
    } else {
5.    reject μ
    }
}
```

Figure 2: Oracle Reduction from Nonregular$_{\mathsf{TM}}$ to Regular$_{\mathsf{TM}}$

Theorem 6. *The "Regular Language" problem is undecidable.*

Proof. It follows by Claims 3, 4 and 5, above, that the above function f is a ***many-one reduction*** from A$_{\mathsf{TM}}$ to Nonregular$_{\mathsf{TM}}$ – so that A$_{\mathsf{TM}}$ $\preceq_{\mathsf{M}}$ Nonregular$_{\mathsf{TM}}$. Since A$_{\mathsf{TM}}$ is undecidable it follows that the language Nonregular$_{\mathsf{TM}}$ is undecidable, as well. Since this language is the language of Yes-instances of the "Regular Language" problem, this establishes the claim. □

Proving that the "Regular Language" Problem is Undecidable

In order to prove that the "Regular Language" problem (that is, its language Regular$_{\mathsf{TM}}$ of Yes-instances) is undecidable, it is sufficient to give an ***oracle reduction*** from Nonregular$_{\mathsf{TM}}$ to Regular$_{\mathsf{TM}}$. With that noted, consider the algorithm shown in Figure 2.

Claim 7. *Let $\mu \in \Sigma^{\star}_{\mathsf{TM}}$. Then — assuming the existence, and use, of an algorithm to perform the test for membership in* Regular$_{\mathsf{TM}}$ *at line 2, an execution of the algorithm in Figure 2 always halts. Furthermore, this execution ends by accepting μ if $\mu \in$* Nonregular$_{\mathsf{TM}}$*, and this execution ends by rejecting μ if $\mu \notin$* Nonregular$_{\mathsf{TM}}$.

Proof. Let $\mu \in \Sigma^{\star}_{\mathsf{TM}}$ and consider an execution of the algorithm, shown in Figure 2 on input μ — assuming the existence, and use, of an algorithm to perform the test for membership in Regular$_{\mathsf{TM}}$ at line 2. Either $\mu \in$ Nonregular$_{\mathsf{TM}}$, $\mu \in L_{\mathsf{TM}}$ but not in Nonregular$_{\mathsf{TM}}$, or $\mu \notin L_{\mathsf{TM}}$. These cases are considered separately, below.

- Suppose that $\mu \in \text{Nonregular}_{\text{TM}}$. Then it is necessary and sufficient to show that the execution of the algorithm on input μ halts (under the above assumption), accepting μ.

  Now, since $\mu \in \text{Nonregular}_{\text{TM}}$ and $\text{Nonregular}_{\text{TM}} \subseteq L_{\text{TM}}$, $\mu \in L_{\text{TM}}$ and the test at line 1 of the algorithm is passed, so that the test at line 2 is reached. Since $\mu \in \text{Nonregular}_{\text{TM}}$ and $\text{Regular}_{\text{TM}} \cap \text{Nonregular}_{\text{TM}} = \emptyset$, so $\mu \notin \text{Regular}_{\text{TM}}$ and (under the above assumption) the test at line 2 is completed and failed. Thus the step at line 4 is reached and the execution of the algorithm ends, accepting μ, as required for this case.

- Suppose, next, that $\mu \in L_{\text{TM}}$ but $\mu \notin \text{Nonregular}_{\text{TM}}$. Then it is necessary and sufficient to show that the execution of the algorithm on input μ halts (under the above assumption), rejecting μ.

  Now, since $\mu \in L_{\text{TM}}$, the test at line 1 is passed, so that the test at line 2 is reached. Since $L_{\text{TM}} = \text{Regular}_{\text{TM}} \cup \text{Nonregular}_{\text{TM}}$, $\mu \in L_{\text{TM}}$ and $\mu \notin \text{Nonregular}_{\text{TM}}$, $\mu \in \text{Regular}_{\text{TM}}$ so that (under the above assumption) the test at line 2 is completed and passed. Thus the step at line 3 is reached and the execution of the algorithm ends, rejecting μ, as required for this case.

- Finally, suppose that $\mu \notin L_{\text{TM}}$. Then it is necessary and sufficient to show that the execution of the algorithm on input μ halts (under the above assumption), rejecting μ.

  Since $\mu \notin L_{\text{TM}}$ the test at line 1 is failed and the test at line 5 is reached and the execution of the algorithm ends, rejecting μ, as required for this case well.

Thus the condition in the claim is satisfied in every possible case, as required. □

Claim 8. $\text{Nonregular}_{\text{TM}} \preceq_O \text{Regular}_{\text{TM}}$.

Proof. It is sufficient to show that — assuming the existence of a (standard, one-tape) "Turing machine that cleans up after itself", M_{NR}, there exists a multi-tape Turing machine that implements the algorithm shown in Figure 2 — to decide the language $\text{Nonregular}_{\text{TM}}$.

Consider, now, a two-tape Turing machine that also uses, as a component a (standard, one Tape) Turing machine that cleans up after itself, that decides the language L_{TM}, and which is modified so that it uses the second tape while leaving the contents and position of the tape head for the first tape unchanged: This exists, since the language L_{TM} is decidable.

Since an oracle reduction to $\text{Regular}_{\text{TM}}$ is being established, we may also assume the existence of a (standard, one-tape) "Turing machine that cleans up after itself" that decides the language $\text{Regular}_{\text{TM}}$. This can also be modified to be used as a component — so that it uses the *first* tape while leaving the contents and position of the tape head for the *second* tape unchanged.

It is sufficient, now, for the multi-tape Turing machine to carry out the following process when executed on an input string $\mu \in \Sigma_{\text{TM}}^{\star}$:

1. Copy the input string, μ, onto the second tape, without changing the contents of the first tape, and moving each tape head back to the leftmost cells of its tape.
2. Apply the component that decides membership in L_{TM} — rejecting μ if it is decided that $\mu \notin L_{\text{TM}}$ and proceeding to the next step, otherwise.
3. Apply the component that decides membership in $\text{Regular}_{\text{TM}}$ (noting that a copy of μ is still on the first tape). If it is decided that $\mu \in \text{Regular}_{\text{TM}}$ then *reject* μ. Otherwise, *accept* μ.

Each step in this computation can be carried out using a Turing machine (under the assumption in the statement of the claim), and it correctly implements the algorithm in Figure 2, so that it decides the language $\text{Nonregular}_{\text{TM}}$. Thus $\text{Nonregular}_{\text{TM}} \preceq_{\text{O}} \text{Regular}_{\text{TM}}$, as claimed. □

Theorem 9. *The "Regular Language" problem is undecidable.*

Proof. It follows by Claim 8 that $\text{Nonregular}_{\text{TM}} \preceq_{\text{O}} \text{Regular}_{\text{TM}}$ and, since $\text{Nonregular}_{\text{TM}}$ is undecidable (as observed when proving Theorem 6), it follows that the language $\text{Regular}_{\text{TM}}$ is also undecidable. Since this is the language of Yes-instances of the "Regular Language" problem, this establishes the claim. □

Rice's Theorem

The final result, mentioned here, is a generalization of the above ones.

Theorem 10 (Rice's Theorem)**.** *Suppose P is a property of Turing machines that satisfies the following conditions:*

- *This property is "nontrivial:" There exists at least one Turing machine M_{Yes} that satisfies this property, and at least one Turing machine M_{No} that* does not *satisfy this property.*
- *This is actually a property of the* **languages** *of these machines: That is, if M_1 and M_2 are Turing machines (with the same input alphabet) such that $L(M_1) = L(M_2)$ then M_1 satisfies this property if and only if M_2 does.*
- *Either the empty set* always *satisfies this property (regardless of the input alphabet, Σ, being used to define this as a subset of $\Sigma^\star$) or it* never *satisfies this property.*[1]

Then the language $L_P \subseteq L_{\text{TM}}$, consisting of the set of encodings of Turing machines that satisfy property P, is undecidable.

[1] While one does not generally see this included in a statement of Rice's Theorem it seems to be a necessary one.

How This is Proved. Since the property P can be replaced by its complementary property ("the original property is *not* satisfied") we may assume, without loss of generality, that, the empty languages *do not* satisfy this property.

Rice's Theorem can then be established by modifying the proof given, above, that the "Non-regular Language" problem is undecidable — with the Turing machine M_{Yes}, mentioned in the statement of the claim, replacing the Turing machine "M_{Non}" in the above argument. □