Lecture #17: Many-One Reductions
A First Example of a Many-One Reduction

Overview

Let $L, \widehat{L} \subseteq \Sigma^\star$ for an alphabet Σ, such that $L \neq \Sigma^\star$, and suppose that $\widehat{L}$ is ***decidable***. The preparatory material for Lecture #17 included an outline of a proof that $L \cap \widehat{L} \preceq_{\mathsf{M}} L$. This document — which is for interest only — includes a more complete proof of this result.

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

The Many-One Reduction To Be Considered

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

$$f(\omega) = \begin{cases} \omega & \text{if } \omega \in \widehat{L}, \\ \mu_{\mathsf{No}} & \text{if } \omega \notin \widehat{L}. \end{cases} \tag{1}$$

Proof of a First Required Property

Claim 1. *For all $\omega \in \Sigma^\star$, if $\omega \in L \cap \widehat{L}$ then $f(\omega) \in L$.*

Proof. 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 $f(\omega) \in L$ for all $\omega \in \Sigma^\star$ such that $\omega \in L \cap \widehat{L}$, as claimed. □

Claim 2. *For all $\omega \in \Sigma^\star$, if $\omega \notin L \cap \widehat{L}$ then $f(\omega) \notin L$.*

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

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

These cases are considered separately, below.

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

$$\begin{aligned} f(\omega) &= \omega && \text{(since } \omega \in \widehat{L}\text{)} \\ &\notin L && \text{(by the choice of } \omega \text{ in this case).} \end{aligned}$$

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

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

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* $\omega \in \Sigma^\star$, as claimed. □

It follows by Claims 1 and 2 that $\omega \in L \cap \widehat{L}$ if and only if $f(\omega) \in L$ for all $\omega \in \Sigma^\star$. That is, the total function f satisfies the first property of a "many-one reduction" from $L \cap \widehat{L}$ to L.

Proof of a Second Required Property

Consider the algorithm shown in Figure 1 on page 3.

Claim 3. *The algorithm shown in Figure 1 is correct. That is, for every string $\omega \in \Sigma^\star$, if this algorithm is executed on input ω then this execution halts with $f(\omega)$ returned as output.*

Proof. Let $\omega \in \Sigma^\star$, and consider the execution of the algorithm in Figure 1. Either $\omega \in \widehat{L}$ or $\omega \notin \widehat{L}$; these cases are considered separately, below.

- If $\omega \in \widehat{L}$ then the test at line 1 is checked and passed, so that the step at line 2 is executed, and ω is returned as output. Now, since $f(\omega) = \omega$ in this case (as one can see by examining the definition of f at line (1)), one can see by an examination of this step that the execution of the algorithm halts, with $f(\omega)$ returned, in this case.

```
On input ω ∈ Σ* {
1.  if (ω ∈ L̂) {
2.    return ω
    } else {
3.    return μ_No
    }
}
```

Figure 1: A "High-Level" Algorithm to Compute the Function f

- If $\omega \notin \widehat{L}$ then the test at line 1 is checked and failed, so that the step at line 3 is executed, and μ_{No} is returned as output. Since $f(\omega) = \mu_{\text{No}}$ in this case (as one can see by examining the definition of f at line (1)), one can see by an examination of this step that the execution of the algorithm halts, with $f(\omega)$ returned, in this case as well.

Since the execution of the algorithm halts with $f(\omega)$ returned, in both cases, and since ω was arbitrarily chosen from $\Sigma^\star$, this establishes the claim. □

Now consider a two-tape Turing machine that implements the algorithm shown in Figure 2 on page 4 — supposing that $M_{\widehat{L}}$ is a (standard) Turing machine that decides the language $\widehat{L}$ and that cleans up after itself (as described in Tutorial Exercise #10): Since the language $\widehat{L}$ is decidable a Turing machine, with these properties, exists.

Claim 4. *The above function f is computable.*

Sketch of Proof. It follows by Claim 3, above, that the high-level algorithm, that is shown in Figure 1, computes the function f.

As noted above, since the language $\widehat{L}$ is decidable, there exists a (standard) Turing machine $\widehat{M}$, deciding the language $\widehat{L}$, that "cleans up after itself". This can be used a component in a two-tape Turing machine that implements the "implementation-level" algorithm shown in Figure 2.[1]

[1] Students who have been attending lecture presentations and completing tutorial exercises should be able to give formal descriptions for Turing machines that carry out the steps at lines #1 and #3–#5 of this implementation-level algorithm — and it should not be difficult to describe how to modify a formal description of $M_{\widehat{L}}$, in order to produce a formal description of a Turing machine that implements the step at line #2. A formal description of a Turing machine that implements the whole algorithm can then be described.

On input $\omega \in \Sigma^{\star}$ {

1. Copy the input string ω from Tape #1 to Tape #2 and move the tape heads back to the leftmost cells — so that both tapes store a copy of the input string.
2. Execute a modified copy of Turing machine $M_{\widehat{L}}$ that uses Tape #2 as its tape — without moving the location of the tape head for Tape #1 or changing the contents of this tape — and with transitions to $M_{\widehat{L}}$'s accepting state redirected to state that begins execution of the step at line #3, and with transitions to $M_{\widehat{L}}$'s rejecting state redirected to a state that begins execution of the step at line #4.
3. Copy the input string ω from Tape #1 to Tape #2 and move the tape heads back to their leftmost cells — replacing all symbols on Tape #1 with copies of "$\sqcup$" (during the movement of the tape heads back to the left). Enter the halting state, so that the string ω has been returned as output if this step has been reached and executed.
4. Move right over the non-blank symbols on Tape #1 and then move back to the left, erasing non-blank symbols with copies of "$\sqcup$" during a movement of the tape head back to the left — so that the tape head for Tape #1 is at the leftmost cells and no non-blank symbols are stored on this tape.
5. Write a copy of the string μ_{No} on the leftmost cells of Tape #2, move the tape head for this tape back to the leftmost cell, and enter the halting state, so that the string μ_{No} has been returned if the steps at lines #4 and #5 have been reached and executed.

}

Figure 2: Implementation-Level Description of Turing Machine to Compute the Function f

Note that the implementation-level algorithm in Figure 2 implements the high-level algorithm (which computes f) in Figure 1:

- The test at line $\#1$ of the algorithm in Figure 1 is carried out using the steps at lines #1 and #2 of the implementation-level algorithm in Figure 2.
- The step at line $\#2$ of the algorithm in Figure 1 is carried out using using the step at line #3 of the implementation-level algorithm in Figure 2.
- The step at line $\#3$ of the algorithm in Figure 1 is carried out using the step at lines #4–#5 of the implementation-level algorithm in Figure 2.

Thus a two-tape Turing machine, implementing the algorithm in Figure 2, also implements the high-level algorithm in Figure 1, so that it computes the function f.

It now follows by results (concerning the equivalence of multi-tape Turing machines and standard Turing machines) in the preparatory material and supplemental documents for Lecture #14 that the function f is computable, as claimed. □

Conclusion

It follows by Claims 1, 2 and 4 that the function f is a many-one reduction from $L \cap \widehat{L}$ to L. Thus

$$L \cap \widehat{L} \preceq_{\mathsf{M}} L.$$