1 Rice’s Theorem and Problems about Turing Machines

Section 5.1 of the textbook talks about several undecidable problems about TMs, and we could imagine many others:

- The “emptiness problem” $E_{TM}$: Given the code for a TM, does the TM accept any strings at all? (Formally, $E_{TM} = \{ \langle M \rangle \mid L(M) = \emptyset \}$)

- The “all strings problem” $ALL_{TM}$: Given the code for a TM, does the TM reject any string? (Formally, $ALL_{TM} = \{ \langle M \rangle \mid L(M) = \Sigma^* \}$)

- The “regex problem” $REG_{TM}$: Given the code for a TM, can the TM be replaced by a regular expression? (Formally, $REG_{TM} = \{ \langle M \rangle \mid L(M) \text{ is regular} \}$)

- The “password checker problem” $SINGLETON_{TM}$: Given the code for a TM, is there only one input that it will accept? (Formally, $SINGLETON_{TM} = \{ \langle M \rangle \mid |L(M)| = 1 \}$)

- The “Java parser problem” $JParser_{TM}$: Given the code for a TM, is it just checking whether its input is a syntactically legal Java program? (If you like, replace “Java” with “C#”, “Python”, “Ruby”…)

A common element among all of these problems is that you can answer them by knowing what set of strings a TM accepts, regardless of how it works. For example, if a TM accepts no strings, then it belongs to $E_{TM}$, and it doesn’t matter if this is because it goes into an infinite loop on all strings, or because it rejects right away on all strings, or because it tests whether its input string is the binary representation of a nontrivial integer solution to $a^n + b^n = c^n$, for $n > 2$.

If a problem about TMs has this feature, we call the problem a language property. Formally, we say that a set $T$ that consists of TM encodings\(^1\) is a language property if whenever two TMs $M_1$ and $M_2$ recognize the same language, either (the code for) both of them belong to $T$, or neither of them do, that is:

For all TMs $M_1, M_2$, If $L(M_1) = L(M_2)$, then $\langle M_1 \rangle \in T \iff \langle M_2 \rangle \in T$.

\(^1\)That is, strings that are the code for some TM
Another thing these problems have in common is that they’re nontrivial: there are some TMs where the answer is “yes”, and some where the answer is “no.” Formally, a set $T$ that consists of TM encodings is nontrivial if there are TMs whose code belongs to $T$ and TMs whose code does not, that is:

There exist TMs $M_{in}$ and $M_{out}$ such that $\langle M_{in} \rangle \in T$ and $\langle M_{out} \rangle \not\in T$.

Another thing these problems have in common is that we can use almost the same reduction from $A_{TM}$ to prove each of them are undecidable. The proof of Rice’s theorem, which we covered in class, generalizes this reduction to any problem that is a nontrivial language property:

**Rice’s Theorem.** Let $T$ be a set of TM encodings that is a nontrivial language property. Then $A_{TM} \leq_m T$. (Therefore, $T$ is undecidable.)

This is an extremely powerful theorem, because it lets us prove that problems (about TMs) are undecidable without having to write a reduction. We simply need to show that $T$ is nontrivial and is a language property, and then we can use Rice’s theorem to conclude that $T$ is undecidable.

**Exercise 1.** Why is it important that $T$ is nontrivial?

2 Examples

Here are a few rules of thumb to keep in mind when using Rice’s Theorem:

- The proof that $T$ is a language property usually consists of finding a way to rephrase $T$’s definition so that it only tests $L(M)$. (This might be very easy.)

- When designing $M_{in}$ and $M_{out}$, it is usually the case that $M_{rej}$ – the TM that transitions immediately from $q_0$ to $q_{rej}$ on any input symbol – or $M_{acc}$, the TM that transitions immediately from $q_0$ to $q_{acc}$ on any input symbol can be used for one such machine.

- For the other machine needed to show nontriviality, we can describe the TM algorithm informally in pseudocode.

2.1 The “All Strings” Problem

**Theorem 1.** $\text{ALL}_{TM}$ is undecidable.

*Proof.* We’ll apply Rice’s theorem to prove that $\text{ALL}_{TM}$ is undecidable.
First, we must show that $\text{ALL}_T M$ is a language property. Let $M_1$ and $M_2$ be any $TMs$ such that $L(M_1) = L(M_2)$. Then

$$\langle M_1 \rangle \in \text{ALL}_T M \iff L(M_1) = \Sigma^*$$

$$\iff L(M_2) = \Sigma^*$$

$$\iff \langle M_2 \rangle \in \text{ALL}_T M,$$

where lines (1) and (3) follow because of the definition of $\text{ALL}_T M$ and line (2) follows from the assumption $L(M_1) = L(M_2)$. Therefore $\text{ALL}_T M$ is a language property.

Second, we must show that $\text{ALL}_T M$ is nontrivial. Let

- $M_{in} = M_{accept}$, then $\langle M_{in} \rangle \in \text{ALL}_T M$, since $L(M_{in}) = \Sigma^*$;

- $M_{out} = M_{reject}$, then $\langle M_{out} \rangle \notin \text{ALL}_T M$, since $L(M_{out}) = \emptyset \neq \Sigma^*$.

Thus $\text{ALL}_T M$ is nontrivial.

\[\square\]

### 2.2 The Java Parser Problem

**Theorem 2.** $\text{JParser}_{TM}$ is undecidable.

**Proof.** We’ll apply Rice’s theorem to prove that $\text{JParser}_{TM}$ is undecidable.

First note that $\text{JParser}_{TM}$ is a language property. Let $J$ be the set of all syntactically legal Java programs, and let $M_1$ and $M_2$ be any $TMs$ such that $L(M_1) = L(M_2)$. Then

$$\langle M_1 \rangle \in \text{JParser}_{TM} \iff L(M_1) = J$$

$$\iff L(M_2) = J$$

$$\iff \langle M_2 \rangle \in \text{JParser}_{TM},$$

where lines (1) and (3) follow from the definition of $\text{JParser}_{TM}$ and line (2) follows from the assumption that $L(M_1) = L(M_2)$. Therefore $\text{JParser}_{TM}$ is a language property.

Second, we show that $\text{JParser}_{TM}$ is nontrivial.

- The set of syntactically legal java programs is defined by a context free grammar, and we know that any CFG can be decided by a $TM$ from Theorem 4.7 in the textbook. So we know there exists a $TM$ that recognizes the set of syntactically legal java programs; let’s call this $TM$ $M_{in} \in \text{JParser}_{TM}$.

- Let $M_{out} = M_{reject}$; then $L(M_{out}) = \emptyset \neq J$, so $\langle M_{out} \rangle \notin \text{JParser}_{TM}$.

Therefore $\text{JParser}_{TM}$ is nontrivial. \[\square\]
2.3 Accepting a specific string

Consider languages like the following:

- $A_{10101} = \{ \langle M \rangle \mid M \text{ accepts } 10101 \}$
- $A_{00000000000000000} = \{ \langle M \rangle \mid M \text{ accepts } 00000000000000000 \}$
- $A_{\text{magic}} = \{ \langle M \rangle \mid M \text{ accepts } \text{magic} \}$

In each of these cases, we are testing whether a certain string belongs to the language of the TM. We’ll give the proof for $A_{\text{magic}}$:

**Theorem 3.** $A_{\text{magic}}$ is undecidable.

**Proof.** We’ll apply Rice’s theorem.

First, we prove that $A_{\text{magic}}$ is a language property. Let $M_1$ and $M_2$ be TMs such that $L(M_1) = L(M_2)$. Then

\[
\langle M_1 \rangle \in A_{\text{magic}} \iff \text{magic} \in L(M_1) \quad (1)
\]
\[
\iff \text{magic} \in L(M_2) \quad (2)
\]
\[
\iff \langle M_2 \rangle \in A_{\text{magic}} \quad (3)
\]

where lines (1) and (3) follow because $\langle M \rangle \in A_{\text{magic}} \iff \text{magic} \in L(M)$ and line (2) follows by the assumption that $L(M_1) = L(M_2)$.

Next we prove that $A_{\text{magic}}$ is nontrivial:

- Let $M_{\text{IN}} = M_{\text{accept}}$. Then $\text{magic} \in L(M_{\text{IN}})$, since $M_{\text{IN}}$ accepts every string, so $M_{\text{IN}} \in A_{\text{magic}}$.
- Let $M_{\text{OUT}} = M_{\text{reject}}$, then $\text{magic} \notin L(M_{\text{OUT}})$, since $M_{\text{OUT}}$ rejects every string, so $M_{\text{OUT}} \notin A_{\text{magic}}$.

Thus $A_{\text{magic}}$ is a nontrivial language property and satisfies Rice’s Theorem.

**Exercise 2.** Complete the proofs that $A_{10101}$ and $A_{00000000000000000}$ are undecidable.

3 Rice’s Theorem and other programming languages

Programs in other languages than TMs have more complicated behaviors than simply “accepting” or “rejecting” and can have multiple input sources; this makes the concept of a “language property” harder to define. However, as long as we pick a definition for what it means for, say, a Java program to accept a string in the context of a given problem, we can adapt Rice’s Theorem to this context. This tells us that any of the above problems are still undecidable if instead of asking about TMs whose languages have some property, they ask about Java, Python, or C programs that accept languages with this property. So for example, Rice’s theorem tells us that it is undecidable to determine whether an arbitrary Python program is a Java parser.
4 When can I not use Rice’s theorem?

Rice’s theorem has three requirements:

1. The problem instances must be only programs. So any problem that has input that includes more than just a single program can’t directly use Rice’s Theorem. For example, the language $\text{EQ}_{\text{Python, TM}} = \{ \langle P, T \rangle \mid P \text{ is a Python program and } T \text{ is a TM and } L(P) = L(T) \}$ cannot be proven undecidable using Rice’s Theorem alone. But there’s a simple reduction from $E_{\text{TM}}$ to $\text{EQ}_{\text{Python, TM}}$: let $f(\langle m \rangle) = \langle M, \text{p_empty} \rangle$, where $\text{p_empty}$ is the python program:

   ```python
def p_empty(s):
    return False
```

And since we know from Rice’s Theorem that $E_{\text{TM}}$ is undecidable, so is $\text{EQ}_{\text{Python, TM}}$.

2. The problem must be nontrivial. Any “property” that includes all programs or no programs is decidable: just answer “yes” (or “no”). The trick is recognizing that the “property” is always true or false. For example, the problem $\text{COUNTABLE}_{\text{TM}} = \{ \langle M \rangle \mid L(M) \text{ is countable.} \}$ is trivial because every language is a subset of $\Sigma^*$, which is countable by the bijection that lists strings in lexicographic order.

3. The problem must be a language property. If it is possible to have two programs that accept all the same strings, but have different answers for the problem, then Rice’s theorem can’t be applied. For example, the language $\text{SHORTEST}_{\text{java}} = \{ \langle J \rangle \mid J \text{ is the shortest java program that recognizes } L(J) \}$ is not a language property, because for every program $J \in \text{SHORTEST}_{\text{java}}$, we can make a longer program $J’$ that recognizes the same language by adding “no-op” instructions to the beginning of $J$, and by definition we’ll have $L(J) = L(J’)$ but $J’ \notin \text{SHORTEST}_{\text{java}}$. 