Undecidability & Rice's Theorem - Supplement to Section 12.2

For the following, assume we are using the "Universal" Turing encoding described in Section 12.4 of the text, in which all machines are described using 0's and 1's. In particular, assume an input alphabet of $\Sigma = \{0,1\}$.

FUNCTIONAL PROPERTIES

First, we define the notation P_f: For any function f, let P_f denote the set of all Turing machines that compute function f.

Now suppose L is a set of Turing machines such that, for any given function f, either P_f is a subset of L or P_f is disjoint from L.
(In other words, for any function f, either every Turing machine that computes f is in L, or no Turing machine that computes f is in L.)
We also require that L be "non-trivial," which means that L and $\bar{L}$ must both be nonempty (that is, at least one machine must be in L, but also there must be at least one machine not in L.) In this case, we refer to L as a "**functional property**" of Turing machines.

Note: Another way this is sometimes expressed is that the condition for membership in L, rather than L itself, is described as being the "functional property."

For example: If L is defined to be the set of all Turing machines that halt on all strings of even length, then L is a "functional property" of Turing machines. This is because a machine is in L if and only if the domain of the function it computes includes all strings of even length. (The set of strings on which a machine halts can be thought of as the domain of the function that it computes, since output is generated iff the machine halts.)

We will discuss examples of properties that are functional, and also some examples that are not functional. (To show a property is not functional, you just have to find two Turing machines that both have the same property but compute different functions.)

EXAMPLES

Here are a few examples of "*functional* properties." In each of the following, f_M is shorthand for "the function computed by M)

- $L_1 = \{M: M$ eventually halts when given a blank input tape$\}$ …or, equivalently: $L_1 = \{M: \lambda \in dom(f_M)\}$
- $L_2 = \{M: M$ halts on only a finite number of strings$\}$ …equivalently: $L_2 = \{M: dom(f_M)$ is finite$\}$
- $L_3 = \{M: M$ outputs a blank tape for all inputs of even length$\}$
  …equivalently: $\{M: |w|$ even $\rightarrow w \in dom(f_M)$ and $f_M(w) = \lambda\}$
- $L_4 = \{M: M$ halts on all strings containing at most three $0's\}$

By contrast, here are a few examples of *non-functional* properties (also known as "*structural*" properties):

- $N_1 = \{M: M$ halts in at most three steps when given a blank input tape$\}$
- $N_2 = \{M: M$ halts after exactly five steps on all strings of length 5 or less$\}$
- $N_3 = \{M:$ the transition function for M includes the transition $\delta(11, 111) = (111, 1111, 1)\}$

For each of these, we could find two Turing machines, both computing the same function, such that one has the stated property but the other does not. As a simple example of this, consider N_1 above. Let M_1 and M_2 be as follows:

- M_1: a Turing machine that immediately halts regardless of input
- M_2: A Turing machine that moves to the right five times and then halts, again regardless of input

Both of these machines compute the same language – namely, $f(w) = w$ – since neither makes any changes to the input tape, and both halt on all inputs, including a blank input tape. However, M_1 halts in at most three steps, but M_2 requires more than three steps; therefore, even though both machines compute the same function, M_1 is in N_1, but M_2 is not in N_1. Thus, N_1 is not a "functional property" of Turing machines. Similarly we can show that N_2 and N_3 are both non-functional (try for yourself).

Exercises: (answers are included below; try these before reading the answers!)
Determine which of the following describe functional properties of Turing machines:
1. M halts on all strings of length at most 10
2. M halts on all strings in at most 10 steps
3. M has at most 10 states and halts on the input 1
4. M halts on all strings that contain a 0

Answers to exercises:
1. Let M be any Turing machine, and let f be the function that it computes. Then, M will have the stated property if, and only if, $dom(f)$ contains all strings of length 10 or less. Also, it's clear that this is a nontrivial property – at least one machine has this property, but not all of them do. Therefore, **this is a functional property** of Turing machines.

2. **This is not a functional property**. We can show this by designing two machines that both compute the same language such that one has the property and one does not.

- M_1: given any input, immediately halts after 1 step without altering the output. This machine computes $f(w) = w$, and has the stated property of halting on all strings in at most 10 steps.
- M_2: given any input, move the read/write head back and forth 20 times, then halt without altering the output. This machine also computes $f(w) = w$, but does not have the stated property. (It does halt on all strings, but it violates the "at most 10 steps" part.)

Since we can exhibit an example of two Turing machines, both computing the same function, such that one has the stated property and the other does not, we conclude that this is not a functional property.

3. **This is not a functional property**. This is due to the fact that the property references the number of internal states of the machine, which is structural rather than functional. (The "halts on the input 1" part is functional by itself, but as a whole the requirement on states makes this a non-functional property.)

We won't give the full construction here, but it's not hard to see that we can design two machines, both computing the same language (the language $f(w) = w$ would work for this example; as long as it's a function with 1 in the domain, it works) but such that one machine has at most 10 states and the other has more than 10 states.)

For example, M_1 could simply halt immeidately on all inputs; this would only take two states (an initial state and a final state); while M_2 could be designed to run through, say, twenty different states as it moves to the right twenty times, before halting without modifying the input. In this way, we'd describe two machines, both computing $f(w) = w$, such that one has the stated property and one does not.

4. Let M be any Turing machine, and let f be the function that it computes. Then, M will have the stated property if, and only if, the domain of f includes all strings containing a 0. (Equivalently: M does not have the stated property if, and only if, its domain consists entirely of strings of the form $1^n, n \geq 0$.) Also, it's clear that this is a nontrivial property. Therefore, **this is a functional property**.

RICE'S THEOREM: Functional properties of Turing machines are undecidable.

PROOF: Let L be a functional property of Turing machines. Assume (for a contradiction) that L is decidable.

Next, we make the following assumption about L: in order to have the property described by L, a Turing machine *must* halt on *at least one* input. In other words, L does not include any Turing machine that does not halt on any input. (We will justify this assumption at the end of the proof*.)

Now, note that since L is a functional property, L is nonempty. So, we can select a Turing machine, R, such that <R> is in L.

With this setup, we are now going to decide the Halting Set. (Your memory is correct – this is impossible!) That is, we'll design an algorithm which, given an arbitrary Turing machine M and input string w, will decide whether M halts on w. Here's how to do it:

1. Let a Turing machine, M, and a string, w, be given. (Note that these selections have nothing to do with property L; M and w are completely independent of that.)

2. For this choice of (M, w), we create the "auxiliary" Turing machine, $T_{M,w}$, that behaves as follows:
   a. Start with an arbitrary input string x.
   b. *Ignore* x at first; instead, elsewhere on the tape, simulate the computation of M on w.
   c. If M halts on w, erase its output, leaving only x on the tape. (Note: If M does not halt on w, then $T_{M,w}$ will not halt on any input.)
   d. Simulate the computation of R on input x. If R halts on x, the output will be $R(x)$.
   (Note: If R does not halt on x, then $T_{M,w}$ also will not halt on x.)

Now, consider the function that is actually computed by $T_{M,w}$. There are two cases:

- If M does *not* halt on w, then $T_{M,w}$ will not halt on any input. Recall: we stipulated at the outset that no Turing machine with this property can be in L. Thus, $\langle T_{M,w} \rangle \notin L$ in this case.
- If M *does* halt on w, then $T_{M,w}$ takes string x as input and then simulates machine R. The result is that $T_{M,w}$ computes the *same function* as R. Since L is a *functional* property of Turing machines and $\langle R \rangle \in L$, it follows that $\langle T_{M,w} \rangle \in L$ in this case.**

To summarize: $T_{M,w} \in L$ if and only if M halts on input w.

Now, recall our assumption at the outset of this proof: L is decidable. That is, we assumed there exists an algorithm, call it A, which decides L. So, given any Turing machine M and string w, we can apply algorithm A to $T_{M,w}$. Then, A will output "yes" if and only if $T_{M,w}$ is in L… which, as noted above, is true if and only if M halts on w. Thus, A decides the Halting Set – but the Halting Set is undecidable, so this is impossible!

We have reached a contradiction, which arises due to the assumption that L is decidable. Since the above argument holds for all functional properties*, we conclude that no functional property is decidable. ■

* Justification of the assumption that non-halting machines are not in L:
First, note that a set is decidable if and only if its complement is decidable. For, if a Turing machine outputs "yes" if $w \in L$ and "no" if $w \notin L$, simply reversing these two outputs will give us a deciding algorithm for $\bar{L}$. So, rather than writing two separate proofs, simply note that if L *does* include the set of Turing machines that never halt on any input, then the complement, $\bar{L}$ does *not* include these Turing machines. In that case, the above proof can simply be applied to $\bar{L}$ rather than to L. This would show that $\bar{L}$ is undecidable, which would imply that L is undecidable.

** Note: if L were *not* functional, then the proof would break down here. Our construction assumes that, for whatever machine R we selected, any other machine that computes the same function as R must also be in L. Without this assumption, the result that R and $T_{M,w}$ compute the same function wouldn't necessarily imply that $T_{M,w}$ is in L, and so there would be no contradiction.

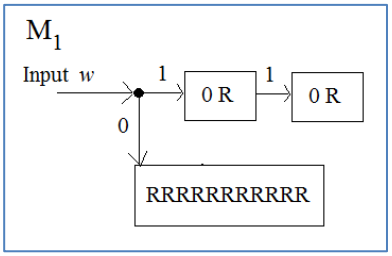
Example: Consider the set of all Turing machines that output 001 on the input 111 and do not halt on the input 000. Show that this is a functional property of Turing machines, and therefore undecidable.

Proof: Let L denote the set of all Turing machines that perform as described. If $M \in L$, then M_f (the function computed by M) is a function such that $f(111) = 001$ and $000 \notin dom(M_f)$. Any given function, f, either satisfies both of these properties (in which case $P_f \subset L$) or it violates one or both properties (in which case $P_f \cap L = \emptyset$). Therefore, this is a functional property, and therefore it is undecidable due to Rice's theorem.

Example: Consider the set of all Turing machines that output 001 on the input 111 in at most five steps, and that run for at least ten steps on the input 000. Show that this is *not* a functional property of Turing machines. Also, describe an algorithm for deciding this property.

Answer: The fact that this property references how the machine operates, rather than just what its inputs and outputs are eventually, implies that this is not a functional property. To show this conclusively, we just need to design two Turing machines, say M_1 and M_2, that compute the same function, f, such that M_1 satisfies the stated property but M_2does not. One such solution is described below. (Note that this is far from the only solution – try to think of another way to do it!)

First, we show a block diagram for M_1 (see the diagram to the right).



Keep in mind that in a block diagram, the lack of an available transition means that the machine halts, rather than "getting stuck."

So, this machine operates as follows:

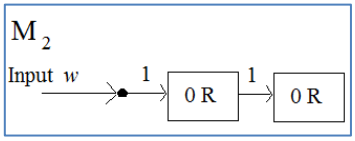
- Immediately halts on empty input
- On an input starting with a 0, moves 11 steps to the right, then halts (note that no changes are made to the tape)
  Note that this makes M_1 "run for at least 10 steps on input 000," as required
- On an input starting with two 1's, replaces both of these 1's with 0's, then halts
  Note that M_2 thus halts on 001 when given input 111, and this takes only two steps; thus M_1 "output[s] 001 on the input 111 in at most five steps," as required
- On an input starting with 10 (or just the input string 1), the machine will overwrite the initial 1 with a 0, then halt.

We have to take care to make M_1 and M_2 (below) compute the same function, so let's state this function explicitly:
For $w \in \{0,1\}^*$:

$$\begin{aligned} f(w) &= w && \text{if } w \text{ does not start with a 1} \\ f(1) &= 0 \\ f(11w') &= 00w' && \text{if } w = 11w' \text{ for some } w' \in \{0,1\}^* \\ f(10w') &= 00w' && \text{if } w = 10w' \text{ for some } w' \in \{0,1\}^* \end{aligned}$$

Now, it remains to design a machine, M_2, that also computes the same function f, but which does not have the stated property.

There are many ways to accomplish this. One very simple solution is shown in the diagram to the right.



The only difference between M_2 and M_1 is that M_2 immediately halts on any string not starting with a 1, while M_1 would first move the r/w head 11 moves to the right before halting. In each case, the input is unchanged if it does not start with a 1. For inputs that do start with a 1, the operation of M_2 is exactly the same as that of M_1. Thus, the two machines will always give the same outputs for identical inputs; that is, they compute the same function. On the other hand, M_2 violates the "at least ten steps on the input 000" rule, since it would immediately on that input.

Thus, there exist two machines that compute the same function such that one has the stated property but the other does not. Since such an example exists, it follows that this is not a functional property of functions.

Example: Describe an algorithm for deciding the non-functional property considered in the preceding example.

A decision algorithm for this property would be to simply run a given machine on the inputs 111 and 000, and observe the results. We can easily verify whether it halts in at most five steps on 111 and runs for at least ten steps on 000.

CAUTION: The fact that a property is non-functional does not automatically imply that it is decidable; in other words, the converse of Rice's Theorem is not true. All we can say with certainty, in all cases, is the following:

IF a property is functional, THEN it is undecidable.
…or, equivalently, the contrapositive…
IF a property is decidable, THEN it must be non-functional..

So, we can use Rice's theorem to show that a property is undecidable, or (with its contrapositive) to show that a property is non-functional. In retrospect, then, we could have shown the property from the previous example was non-functional simply by describing the decision algorithm for it, thus saving a lot of time and effort. …but, that wouldn't have been nearly as much fun!

Practice Exercises:
Determine whether each of the following properties are decidable. If it is decidable, describe an algorithm for deciding it.

1. The set of all Turing machines that halt on all input strings of odd length, but do not halt on any string of even length

2. The set of machines that never halt when given a blank input tape (i.e., input string λ).

3. The set of machines that halt immediately when given a blank input tape.

4. The set of machines which, when in state q_1 and reading a 0, go to a final state.

5. The set of all machines that halt on a finite number of strings.

[Solutions on the next page]

Solutions:

1. This is a functional property, so Rice's theorem tells us it is **undecidable**. Let M be any Turing machine, and let f be the function it computes. Then, f has the stated property if, and only if, $dom(f)$ includes all strings of odd length and no strings of even length. Since this determination depends entirely on the function computed by M, this is a functional property.

2. This is a functional property, so it is **undecidable**. A Turing machine has this property if and only if λ is not in the domain of the function that it computes.

3. This is **decidable**: the deciding algorithm is to simply run a given machine on a blank input tape, and see if it immediately halts. (Note: "immediately" here means that no moves are made before halting, so this determination could be made right away.)

Note: It's tempting, after reading #2, to also declare #3 to be a functional property – at first glance, we might conclude that a machine has this property if and only if λ IS in the domain of the computed function. And this would be correct, if not for the "immediately" part of the stated property. "Immediately" gets to how the machine computes, rather than what it computes – that is, it is structural rather than functional. If #3 had read "the set of machines that halt on a blank input tape," without the word "immediately," then it would also be an example of a functional property, and it would be undecidable.

4. This is **decidable**. Given a machine M, the algorithm would be to scan the (finite) code for M, to determine whether any of its transitions are of the form $\delta(q_1, 0) = (q_F,*,*)$ where q_F is a final state. A machine has the stated property if and only if it has a transition rule of this form.

5. This is a functional property, so it is **undecidable**. A Turing machine has this property if and only if the domain of the function it computes is finite.