Problem Set 7: Undecidability
CS254, Fall 2019, Anna Rafferty

You are welcome to talk to others about the problems on this problem set, but you should **read and attempt each problem prior do discussing it with others.** The write up must be your own work. That means that **you should write your answers by yourself, without looking at notes from sessions with collaborators, and you must be able to explain any answer you write down.** You should list anyone you collaborated with in your collaboration form.

Due dates: 3 problems are due Monday, October 28; the remaining problems are due Wednesday, October 30. As for all homeworks, if a problem is due on a specific day it must be turned in by 9PM on that day. See homework handout (a copy is posted on Moodle) for more specific homework policies, and check out Problem Solving Tips if you're stuck or not sure how to get started. Failing to name your files properly may result in a loss of points and potentially receiving no credit for that problem; please name your files properly!

1. Recall that, for any two languages A and B, A/B is the set of all strings w such that there exists a string $x \in B$ such that $wx \in A$. In an earlier problem set, we showed that if A is regular, A/B is regular. Our proof showed that a DFA for A/B must exist, although it didn't tell us how to directly construct one - there was a reason for that, as suggested below.

   Imagine a TM N. It takes input $\langle D, M\rangle$ where D is a DFA and M is a TM. N processes $\langle D, M\rangle$ and eventually halts with string $\langle C\rangle$ on its tape, and nothing else. C is a DFA such that $L(C) = L(D)/L(M)$. Note that N halts on all inputs and we don't care whether it accepts or rejects - we only care about the contents of the tape when N halts. Prove that N cannot exist. *Hint: Use N to build a decider for $EMPTY_{TM}$.*

   Solution: Assume that N exists. Additionally, let DFA_A be a DFA with a single state that is a final state, and that loops back to that state on any input. Thus, $L(DFA_A) = \Sigma^*$. We know from Sipser that $EMPTY_{DFA}$ is decidable: let $M_{E_{DFA}}$ be a decider for $EMPTY_{DFA}$. We'll use N, DFA_A, and $M_{E_{DFA}}$ to build a decider M' for $EMPTY_{TM}$. M' works as follows:

   On input $\langle T\rangle$:

   1. Run N on input $\langle DFA_A, T\rangle$, leaving DFA C on the tape.
   2. Run $M_{E_{DFA}}$ on $\langle C\rangle$. If it accepts, then accept $\langle T\rangle$. If it rejects, then reject $\langle T\rangle$.

   We note that $L(C) = L(DFA_A)/L(T) = \Sigma^*/L(T)$.

   If $L(T) = \varnothing$, then $L(C) = \Sigma^*/\varnothing = \varnothing$. Thus, $M_{E_{DFA}}$ will accept $\langle C\rangle$, and M' will accept $\langle T\rangle$.

   If $L(T) \neq \varnothing$, then $L(C) = \Sigma^*/L(T) = \Sigma^*$ (as proven on problem set 2). Thus, $M_{E_{DFA}}$ will reject $\langle C\rangle$, and M' will reject $\langle T\rangle$.

   Thus, M' accepts $\langle T\rangle \in EMPTY_{TM}$ and rejects $\langle T\rangle \notin EMPTY_{TM}$: M' is a decider for $EMPTY_{TM}$. But, we've previously shown that $EMPTY_{TM}$ is undecidable . This is a contradiction, so N does not exist.

2. One of the following sets if r.e. and one is not. Decide which is which, and prove that the one which is not r.e. is not r.e. and that the one that is r.e. is r.e. In your proofs, do not rely on the fact that I told you that one is r.e. and one is not, and to show the one that is not r.e. but not r.e., your proof should show that if it were r.e., you could decide a language we know is not r.e.

   - $L_1 = \{\langle M\rangle \mid |L(M)| \geq 254\}$
   - $L_2 = \{\langle M\rangle \mid |L(M)| \leq 254\}$

   Solution: L_1 is r.e. We can show this by choosing an arbitrary ordering over Σ^*, e.g., ordering strings by length and alphabetically within length $(\epsilon, 0, 1, 00, 01, 10, 11, \ldots)$. Then, we construct the same sort of machine M' we did for showing enumerators and TMs to be equivalent:

   - Repeat for i = 1, 2, 3, ...:

– Simulate M for i steps on strings $s_1, \ldots, s_i$. If at least 254 accept, then accept.

If M accepts 254 different strings, then as soon as i is bigger than the 254th string that M accepts (according to the ordering on Σ^*) and i is also bigger than the number of steps that M takes on whichever of those strings on which M is the slowest, M' will accept. Conversely, M' only accepts if we've found 254 strings that M accepts (because we actually did the simulations).

L_2 is not r.e. We can prove this by showing that if we had a TM M' that recognizes L_2, we could build a machine that recognizes $\overline{HALT}$. Assume L_2 is r.e. and let M' be a TM such that $L(M') = L_2$. Then, we construct the following TM $M_{\overline{HALT}}$ to recognize $\overline{HALT}$:

On input $\langle M, w\rangle$, $M_{\overline{HALT}}$ works as follows:

1. Construct machine M_w that on input y:
   (a) Ignore y and simulate M on w. If M halts, accept y.
2. Run M' on input $\langle M_w\rangle$. If M' accepts, then accept $\langle M, w\rangle$.

Here, if M halts on w, $L(M_w) = \Sigma^*$; otherwise, $L(M_w) = \varnothing$. Thus, if $\langle M, w\rangle \in M_{\overline{HALT}}$, $|L(M_w)| = \varnothing$, so $\langle M_w\rangle \in L_2$ and M' will accept it; because M' accepts $\langle M_w\rangle$, $M_{\overline{HALT}}$ accepts $\langle M, w\rangle$. If $\langle M, w\rangle \notin M_{\overline{HALT}}$, $|L(M_w)| = \infty$, so $\langle M_w\rangle \notin L_2$ and M' will not accept it; because M' doesn't accept $\langle M_w\rangle$, $M_{\overline{HALT}}$ doesn't accept $\langle M, w\rangle$. Thus, $M_{\overline{HALT}}$ is a recognizer for $\overline{HALT}$, but Sipser shows that this language is not r.e. Thus, our assumption is incorrect and L_2 is not r.e.

3. Sipser 5.30b,c

Solution: b. We want to show that $L = \{\langle M\rangle \mid M \text{ is a TM and } 1011 \in L(M)\}$ is undecidable via Rice's theorem. We must show that L is nontrivial and semantic. To show it is nontrival, consider a DFA that recognizes only 1011; we know we can build a TM for any DFA, so there exists a Turing machine that is in L. Additionally, consider the complement of the language of that DFA; this language is also regular (since regular languages are closed under complement), so we can also build a DFA from it and a TM from that DFA, giving us a TM that is not in L. Thus, L is nontrivial. Second, imagine we have TMs M_1 and M_2 such that $L(M_1) = L(M_2)$. $M_1 \in L$ iff $1011 \in L(M_1)$; this occurs iff $1011 \in L(M_2)$, so $M_2 \in L$ iff $M_1 \in L$. Thus,the property is semantic. By Rice's theorem, any language over TMs that is nontrivial and semantic is undecidable.

c. We want to show that $ALL_{TM} = \{\langle M\rangle \mid M \text{ is a TM and } L(M) = \Sigma^{ast}\}$ by Rice's theorem. Again, we must show that L is nontrivial and semantic. To show it is nontrivial, consider a Turing machine M where the start state is the accept state. $L(M) = \Sigma^*$, so $M \in ALL_{TM}$. Then, consider M' where the start state is the reject state. $L(M') = \varnothing$, so $M' \notin ALL_{TM}$. Thus, ALL_{TM} is nontrivial. Second, imagine we have TMs M_1 and M_2 such that $L(M_1) = L(M_2)$. $M_1 \in L$ iff $\Sigma^* = L(M_1)$; this occurs iff $\Sigma^* = L(M_2)$, so $M_2 \in L$ iff $M_1 \in L$. Thus,the property is semantic. By Rice's theorem, any language over TMs that is nontrivial and semantic is undecidable.

4. When a compiler compiles a program, it generally performs optimizations to try to improve the speed or memory usage of the code. For instance, the compiler may attempt to remove code that is never called, reducing the size of the compiled code. You might wonder whether it's possible to do this perfectly (i.e., delete all code that would never be called) - that's the question we'll address here.

   Define $UNUSED_{TM}$ as the set of $\langle M, q\rangle$ such that M is a TM and q is a state of M and there does not exist a string w such that M enters q while processing w. Decide if $UNUSED_{TM}$ is decidable, r.e. but not decidable, co-r.e. but not r.e., or none of the above. Prove your answer correct. Additionally, write a sentence about what you think this means with respect to whether we can build a compiler that is guaranteed get rid of all code that would never be used when processing any input.

Solution: $UNUSED_{TM}$ is co-r.e. but not decidable. We'll prove this showing two things (a) $UNUSED_{TM}$ is not decidable, and (b) $\overline{UNUSED_{TM}}$ is r.e.

First, we show that $UNUSED_{TM}$ is unrecognizable, which implies it is not decidable. We'll prove this by via a mapping reduction from $\overline{A_{TM}}$ to $UNUSED_{TM}$. We define our computable function $f(\langle M, w\rangle)$ as follows:

$f(\langle M, w\rangle)$:

1. Construct TM M_w which works as follows on an input s:

   (a) Simulate M on w (ignoring s). If M accepts w, accept.

2. Output $\langle M_w, q_{acc}\rangle$, where q_{acc} is the accept state of M_w.

If $\langle M, w\rangle \in \overline{A_{TM}}$, then $w \notin L(M)$, meaning M never accepts w. That means that M_w never accepts any string. Thus, M_w never enters its accept state, so $\langle M_w, q_{acc}\rangle \in \mathit{UNUSED}_{TM}$. Similarly, if $\langle M, w\rangle \notin \overline{A_{TM}}$, then $w \in L(M)$, meaning M accepts w and consequently M_w accepts every string. Then M_w does enter its accept state on (all) strings, so $\langle M_w, q_{acc}\rangle \notin \mathit{UNUSED}_{TM}$. Thus, we have defined a computable function f where $\langle M, w\rangle \in \overline{A_{TM}}$ if and only if $f(\langle M, w\rangle) = \langle M_w, q_{acc}\rangle \in \mathit{UNUSED}_{TM}$. Since $\overline{A_{TM}}$ is not recognizable and we have shown $\overline{A_{TM}} \leq_M \mathit{UNUSED}_{TM}$, UNUSED_{TM} is not recognizable.

Next, we'll show that $\overline{\mathit{UNUSED}_{TM}}$ is r.e. by building a recognizer for it. We construct the following TM M_U, which on input s works as follows:

1. Decodes s into a pair $\langle M, q\rangle$. If s can't be decoded in this way, accept.

2. For each of $i = 1, 2, 3, \ldots$:

   (a) Run M for i steps on each of $s_1, \ldots, s_i$, where $s_1, \ldots, s_i$ are the first i strings in canonical order. If state q is ever entered, then we accept.

This is similar to our enumerator proof: if M enters q on any string, then it must enter q after a finite number of steps on some string. Since for any string and number of steps, there's a finite point at which M_U will run M for that number of steps on that string, meaning that M_U will accept $\langle M, q\rangle$. If M never enters q, then this machine will never accept. Thus, $s \in L(M_U)$ if and only if s doesn't represent a TM-state pair or s represents a pair where M enters q - i.e., $s \in L(M_U)$ iff $s \in \overline{\mathit{UNUSED}_{TM}}$. Thus, $\overline{\mathit{UNUSED}_{TM}}$ is r.e. because we can build a recognizer for it. (Note: If you ignored the possibility of s not being able to be decoded into a pair $\langle M, q\rangle$, that's fine.)

Thus, we've shown that UNUSED_{TM} is not r.e. but it is co-r.e. because its complement is r.e. This means that it's impossible for the compiler to do the optimization above perfectly: it's impossible to on all inputs to decide if a state (i.e., bit of code) for a TM is ever needed.

Note: Another perfectly valid proof strategy here would be to show UNUSED_{TM} is undecidable rather than unrecognizable.

5. Sipser 5.35. *Hint: You'll likely want to look back at Sipser's examples or class and worksheet examples that involve CFGs and decidability/recognizability.*

Solution: a. We show that $NECESSARY_{CFG} = \{\langle G, A\rangle \mid A \text{ is a necessary variable in } G\}$ is recognizable. We can do this by constructing a TM that first modifies G to remove A and all productions that include A; call the new grammar G'. We then use a canonical ordering on strings (as in the second problem). For each string w, we simulate $A_{CFG}(\langle G', w\rangle)$ and $A_{CFG}(\langle G, w\rangle)$. We know that A_{CFG} is decidable from Sipser, thus both of these will either accept or reject. If the latter accepts and the former rejects, this machine accepts: the string w could not be produced when A was removed from the grammar.

b. We show that $NECESSARY_{CFG}$ is undecidable. We do so by a mapping reduction from ALL_{CFG} to $\overline{NECESSARY_{CFG}}$. We know ALL_{CFG} is undecidable, so this will imply that $\overline{NECESSARY_{CFG}}$ is undecidable; this means its complement $NECESSARY_{CFG}$ is also undecidable. We construct a mapping function f. On input $\langle G\rangle$, f constructs the following output G':

- G' is identical to G, except it has a new variable A and some rules involving A. We add rules $S \to A$ for start symbol S and $A \to \epsilon \mid aA$ for all $a \in \Sigma$.

The language of G' is Σ^*: using just the new rules, we can see that the grammar produces any string in Σ^*. $\langle G', A\rangle \in \overline{NECESSARY_{CFG}}$ iff this grammar has the same language without A; i.e., if $L(G) = \Sigma^*$, meaning $\langle G\rangle \in ALL_{CFG}$. Thus, this is a mapping reduction from ALL_{CFG} to $\overline{NECESSARY_{CFG}}$, meaning that since we know ALL_{CFG} is undecidable, $\overline{NECESSARY_{CFG}}$ is undecidable as well. $NECESSARY_{CFG}$ must also be undecidable since decidable languages are closed under complement.

6. **Optional additional problems:** 5.5 (this helps to make clear what we know if we have a mapping reduction from one language to another), 5.10 (this is a good typical example of a proof by contradiction that something is undecidable).