Assignment 10: Decidable, Semidecidable, and Undecidable

Due: Wednesday, December 5

- Emails about this assignment should be directed to `cs81help@cs.hmc.edu`.
- The usual collaboration rules apply. You may *discuss* an exercise with any other student(s) currently taking CS 81 as long as:
  - You contribute equally;
  - You come away from this discussion only with *understanding in your head* — no written materials or computer notes may be retained;
  - Your submission is authored solely by you, on a separate occasion.
- You should refer only to materials from this semester of CS 81 (lecture notes, handouts, textbooks, grutors, profs, etc.).
- Bring a writeup/printout to class. Illegible answers will get no credit.
- Make sure your submission includes your name!

---

1. Skim Chapters 19, 20, and 21 of Rich, paying particular attention to the examples.

   Come up with two questions about the reading; these should be questions where you're not sure of the answer. These may relate to points where the book is confusing, or simply to some related question or conjecture that occurs to you while doing the reading.

2. Consider the following languages (where M ranges over TMs). Are they decidable? If not, are they semidecidable? Prove each answer.

   $L_1 = \{ \langle M \rangle \mid M \text{ accepts at least 999 strings} \}$
   $L_2 = \{ \langle M \rangle \mid M\text{'s program includes a transition that writes a on the tape} \}$
   $L_3 = \{ \langle M \rangle \mid M \text{ eventually writes a non-blank symbol to the tape when given input } \varepsilon \}$
   $L_4 = \{ \langle M \rangle \mid M \text{ accepts the string } a \}$
   $L_5 = \{ \langle M \rangle \mid M \text{ is the only TM accepting } L(M) \}$
   $L_6 = \{ \langle M \rangle \mid M \text{ halts on input } \epsilon \text{ in no more than 1000 steps} \}$
   $L_7 = \{ \langle M, w \rangle \mid M \text{ accepts } w \text{ in an even number of steps} \}$

3. We know that A_{TM} is semidecidable but not decidable, and its complement $\neg A_{TM}$ is neither semidecidable nor decidable. Is the third language

$$R_{TM} = \{ \langle M, w \rangle \mid M \text{ halts in the reject state when run on } w \}$$

decidable, semidecidable, or neither? Prove your answer.

4. Although the language of correct addition statements

$$\mathit{ADD} = \{ x{=}y{+}z \mid x, y, z \text{ are decimal integers, and } x \text{ is the sum of } y \text{ and } z \}$$

can easily be shown not to be regular, you can surely write a program (in Python, or Java, or Turing-Machine, etc.) that takes a string and decides whether it's one of these correct sums of binary numbers; *ADD* is a decidable language.

Now, even if you restrict yourself to a single programming language, (we'll use Turing Machines), there might be many programs to decide *ADD*; they each decide whether the input is a correct binary sum or not, each in slightly different (or very different) ways. We can then consider the set *ADDERS* of all such programs.

(a) Prove that the language

$$\mathit{ADDERS} = \{ \langle M \rangle \mid M \text{ is a Turing Machine that accepts (\textit{semidecides}) the language } \mathit{ADD} \}$$

is not decidable.

(b) Prove that *ADDERS* is also not semidecidable by showing $\neg A_{TM} \leqslant \mathit{ADDERS}$. That is, reduce $\neg A_{TM}$ to *ADDERS* by showing that if *ADDERS* were *semidecidable*, then $\neg A_{TM}$ would be semidecidable too. (You may assume that *ADD* is decidable, i.e., there is at least one Turing Machine that decides it.)

(c) It is possible to encode Turing Machines as strings so that *every* string represents some Turing Machine. In this case, the complement of the formal language *ADDERS* is the set

$$\neg \mathit{ADDERS} = \{ \langle M \rangle \mid L(M) \neq \mathit{ADD} \}$$

Is $\neg$*ADDERS* decidable? If not, is it semidecidable? Prove your answers.

5. **The Elusiveness of Undecidability.** It is widely known that Halting (membership in $H = \{ \langle M, w \rangle \mid M \text{ halts on input } w \}$) is not decidable. But:

   (a) Consider the following machine, which we will call $\texttt{TM}_1$. It first confirms that the tape contains a number expressed in binary (and if not, rejects). It then enters the following loop:

   - All digits are zero, it halts and accepts (ending the loop).
   - Otherwise, it decrements the binary number by 1 (by completely overwriting it) and restarts the loop

   Does the machine $\texttt{TM}_1$ halt on the input 10000000000000?

   (b) Consider a similar machine, which we will call TM_2. It is exactly the same as TM_1, except that each time around the loop it *increments* the binary number by 1. Does the machine $\texttt{TM}_2$ halt on the input 10000000000000?

   (c) We proved in class that Halting is undecidable. Yet you just decided it twice. Why isn't this a contradiction?

   (d) $\texttt{TM}_1$ and $\texttt{TM}_2$ are admittedly not very interesting. You are probably wondering, since Halting is undecidable, where the hard Turing Machines are. Suppose I were to design a very complicated $\texttt{TM}_3$, and prove mathematically that "there is no series of steps by which you can determine whether $\texttt{TM}_3$ terminates on the input 10000000000000" (essentially, that "you will not and cannot know, even in principle, whether TM_3 terminates on this input"). Show that this supposition leads to a contradiction.

   (It follows that, although there's no algorithm to solve the halting problem in general, we cannot point to any *specific* instance as the undecidable input.)