CS103 | Handout 46
Spring 2017 | June 2, 2016

Problem Set 9

What problems are beyond our capacity to solve? Why are they so hard? And why is anything that we've discussed this quarter at all practically relevant? In this problem set – the last one of the quarter! – you'll explore the absolute limits of computing power.

As always, please feel free to drop by office hours or ask questions on Piazza if you have any questions. We'd be happy to help out.

Good luck, and have fun!

Due Wednesday, June 7th at the start of lecture.

Because this problem set is due on the last day of class, no late days may be used and no late submissions will be accepted. Sorry about that! On the plus side, we'll release solutions as soon as the problem set comes due.

Problem One: Password Checking

(We recommend reading the Guide to Self-Reference on the course website before attempting this problem.)

If you're an undergraduate here, you've probably noticed that the dorm staff have master keys they can use to unlock any of the doors in the residences. That way, if you ever lock yourself out of your room, you can, sheepishly, ask for help back in. (Not that I've ever done that or anything.) Compare this to a password system. When you log onto a website with a password, you have the presumption that your password is the only possible password that will log you in. There shouldn't be a "master key" password that can unlock any account, since that would be a huge security vulnerability. But how could you tell? If you had the source code to the password checking system, could you figure out whether your password was the only password that would grant you access to the system?

Let's frame this question in terms of Turing machines. If we wanted to build a TM password checker, "entering your password" would correspond to starting up the TM on some string, and "gaining access" would mean that the TM accepts your string. Let's suppose that your password is the string `iheartquokkas`. A TM that would work as a valid password checker would be a TM M where $\mathscr{L}(M) = \{\texttt{iheartquokkas}\}$: the TM accepts your string, and it doesn't accept anything else. Given a TM, is there some way you could tell whether the TM was a valid password checker?

Consider the following language L:

$$L = \{ \langle M \rangle \mid M \text{ is a TM and } \mathscr{L}(M) = \{\texttt{iheartquokkas}\} \}$$

Your task in this problem is to prove that L is undecidable (that is, $L \notin \mathbf{R}$). This means that there's no algorithm that can mechanically check whether a TM is suitable as a password checker. Rather than dropping you headfirst into this problem, we've split this problem apart into a few smaller pieces.

Let's suppose for the sake of contradiction that $L \in \mathbf{R}$. That means that there is some function

```
bool isPasswordChecker(string program)
```

with the following properties:

- If `program` is the source of a program that accepts just the string `iheartquokkas`, then calling `isPasswordChecker(program)` will return `true`.
- If `program` is not the source of a program that accepts just the string `iheartquokkas`, then calling `isPasswordChecker(program)` will return `false`.

We can try to build a self-referential program that uses the `isPasswordChecker` function to obtain a contradiction. Here's a first try:

```
int main() {
      string me = mySource();
      string input = getInput();

      if (isPasswordChecker(me)) {
            reject();
      } else {
            accept();
      }
}
```

This code is, essentially, a (minimally) modified version of the self-referential program we used to get a contradiction for the language A_{TM}.

(Continued on the next page.)

i. Prove that the above program is not a valid password checker.

ii. Suppose that this program is ***not*** a valid password checker. Briefly explain why no contradiction arises in this case – no formal justification is necessary.

Ultimately, the goal of building a self-referential program here is to have the program cause a contradiction regardless of whether or not it's a password checker. As you've seen in part (ii), this particular program does not cause a contradiction if it isn't a password checker. Consequently, if we want to prove that $L \notin \mathbf{R}$, we need to modify it so that it leads to a contradiction in the case where it is not a password checker.

iii. Modify the above code so that it causes a contradiction regardless of whether it's a password checker. Then, briefly explain why your modified program is correct. (No formal proof is necessary here; you're going to do that in the next step.)

iv. Formalize your argument in part (iii) by proving that $L \notin \mathbf{R}$. Use the proof that $\mathrm{A}_{\mathrm{TM}} \notin \mathbf{R}$ as a template for your proof.

Problem Two: L_{D}, Cantor's Theorem, and Diagonalization

Here's another perspective of the proof that $L_{\mathrm{D}} \notin \mathbf{RE}$. Suppose we let TM be the set of all encodings of Turing machines. That is,

$$TM = \{ \langle M \rangle \mid M \text{ is a TM} \}$$

We can then define a function $\widetilde{\mathscr{L}} : TM \to \wp(TM)$ as follows:

$$\widetilde{\mathscr{L}}(\langle M \rangle) = \mathscr{L}(M) \cap TM$$

This question explores some properties of this function.

i. Briefly describe, in plain English, what $\widetilde{\mathscr{L}}(\langle M \rangle)$ represents. *(You shouldn't need more than a sentence.)*

ii. Trace through the proof of Cantor's theorem from the Guide to Cantor's Theorem, assuming that the choice of the function f in the proof is the function $\widetilde{\mathscr{L}}$. What is the set D that is produced in the course of the proof?

Problem Three: Double Verification

This problem explores the following beautiful and fundamental theorem about the relationship between the **R** and **RE** languages:

$$\text{If } L \text{ is a language, then } L \in \mathbf{R} \text{ if and only if } L \in \mathbf{RE} \text{ and } \overline{L} \in \mathbf{RE}$$

This theorem has a beautiful intuition: it says that a language L is decidable ($L \in \mathbf{R}$) precisely if for every string in the language, it's possible to prove it's in the language ($L \in \mathbf{RE}$) and, simultaneously, for every string not in the language, it's possible to prove that the string is not in the language ($L \in \mathbf{RE}$). In this problem, we're going to ask you to prove one of the two directions of this theorem.

Let L be a language where $L \in \mathbf{RE}$ and $\overline{L} \in \mathbf{RE}$. This means that there's a verifier V_{yes} for L and a verifier V_{no} for $\overline{L}$. In software, you could imagine that V_{yes} and V_{no} correspond to methods with these signatures:

```
bool imConvincedIsInL(string w, string c)
bool imConvincedIsNotInL(string w, string c)
```

Prove that $L \in \mathbf{R}$ by writing pseudocode for a function

```
bool isInL(string w)
```

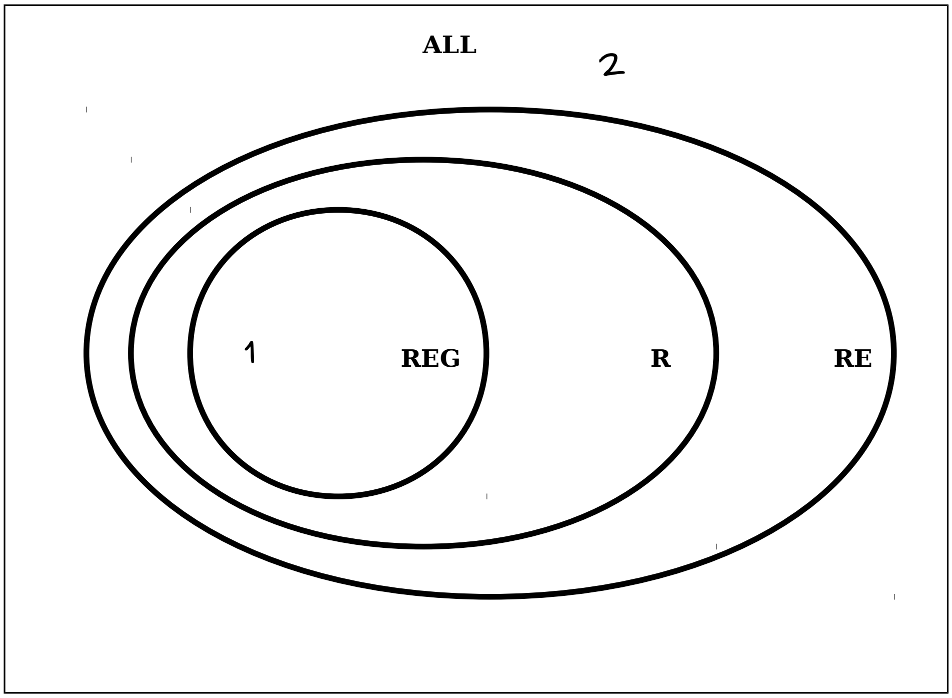
that accepts as input a string w, then returns true if $w \in L$ and returns false if $w \notin L$. Then, write a brief proof explaining why your pseudocode meets these requirements. You don't need to write much code here. If you find yourself writing ten or more lines of pseudocode, you're probably missing something.

The theorem you proved in this problem is extremely useful for building an intuition for what languages are decidable. You'll see this in the next problem

Problem Four: The Lava Diagram

Below is a Venn diagram showing the overlap of different classes of languages we've studied so far. We have also provided you a list of twelve numbered languages. For each of those languages, draw where in the Venn diagram that language belongs. As an example, we've indicated where Language 1 and Language 2 should go. No proofs or justifications are necessary – the purpose of this problem is to help you build a better intuition for what makes a language regular, **R**, **RE**, or none of these.

We strongly recommend reading over the Guide to the Lava Diagram before starting this problem.



1. Σ^*
2. L_D
3. $\{ \texttt{a}^n \mid n \in \mathbb{N} \}$
4. $\{ \texttt{a}^n \mid n \in \mathbb{N}$ and is a multiple of 137 $\}$
5. $\{ \texttt{1}^n\texttt{+}\texttt{1}^m\overset{?}{\texttt{=}}\texttt{1}^{n+m} \mid m, n \in \mathbb{N} \}$
6. $\{ \langle M \rangle \mid M$ is a Turing machine and $\mathscr{L}(M) \neq \emptyset \}$
7. $\{ \langle M \rangle \mid M$ is a Turing machine and $\mathscr{L}(M) = \emptyset \}$
8. $\{ \langle M \rangle \mid M$ is a Turing machine and $\mathscr{L}(M) = L_D \}$
9. $\{ \langle M, n \rangle \mid M$ is a TM, $n \in \mathbb{N}$, and M ***accepts*** all strings in its input alphabet of length at most $n \}$
10. $\{ \langle M, n \rangle \mid M$ is a TM, $n \in \mathbb{N}$, and M ***rejects*** all strings in its input alphabet of length at most $n \}$
11. $\{ \langle M, n \rangle \mid M$ is a TM, $n \in \mathbb{N}$, and M ***loops*** on all strings in its input alphabet of length at most $n \}$
12. $\{ \langle M_1, M_2, M_3, w \rangle \mid M_1, M_2$, and M_3 are TMs, w is a string, and at least two of M_1, M_2, and M_3 accept w. $\}$

Problem Five: The Big Picture

We have covered a *lot* of ground in this course throughout our whirlwind tour of computability and complexity theory. This last question surveys what we have covered so far by asking you to see how everything we have covered relates.

Take a minute to review the hierarchy of languages we explored:

$$\mathbf{REG} \subsetneq \mathbf{CFL} \subsetneq \mathbf{P} \overset{?}{=} \mathbf{NP} \subsetneq \mathbf{R} \subsetneq \mathbf{RE} \subsetneq \mathbf{ALL}$$

The following questions ask you to provide examples of languages at different spots within this hierarchy. In each case, you should provide an example of a language, but you don't need to formally prove that it has the properties required. Instead, describe a proof technique you could use to show that the language has the required properties. There are many correct answers to these problems, and we'll accept any of them.

i. Give an example of a regular language. How might you prove that it is regular?

ii. Give an example of a context-free language is not regular. How might you prove that it is context-free? How might you prove that it is not regular?

iii. Give an example of a language in **P**.

iv. Give an example of a language in **NP**-complete language. *(We'll talk about this on Monday.)*

v. Give an example of a language in **RE** not contained in **R**. How might you prove that it is **RE**? How might you prove that it is not contained in **R**?

vi. Give an example of a language that is not in **RE**. How might you prove it is not contained in **RE**?

Extra Credit Problem: $\mathbf{P} \overset{?}{=} \mathbf{NP}$ (Worth an A+, $1,000,000, and a Stanford Ph.D)

Prove or disprove: **P** = **NP**.