

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, ask questions on Piazza, or send us emails if you have any questions. We'd be happy to help out.

Good luck, and have fun!

Due Friday, December 4 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!

Problem One: Computable Bijections (4 Points)

A function $f: \Sigma^* \rightarrow \Sigma^*$ is called a *computable function* if it is possible to write a function with signature

```
string compute_f(string input)
```

with the following property: for any $w \in \Sigma^*$, $\text{compute_f}(w) = f(w)$. In other words, a function is computable if you can write a piece of code that, given any string w , returns the value of $f(w)$.

A *computable bijection* is a computable function f that's a bijection. We know that all bijections are invertible, and, interestingly, it turns out that all computable bijections are invertible as well.

Prove that if f is a computable bijection, then f^{-1} is computable. To do so, given an arbitrary computable bijection f , give pseudocode for a function `compute_f_inverse` that, given a string w , returns a string x such that $f(x) = w$. Then, write a brief proof explaining why your code is correct.

As a hint, 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.

Problem Two: Double Verification (4 Points)

Here's an interesting fact: if L is a language where $L \in \mathbf{RE}$ and $\bar{L} \in \mathbf{RE}$, then $L \in \mathbf{R}$.

Let L be a language where $L \in \mathbf{RE}$ and $\bar{L} \in \mathbf{RE}$. This means that there's a verifier V_{yes} for L and a verifier V_{no} for \bar{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)
```

Your task is to 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.

As a hint, 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.

Problem Three: Password Checking (8 Points)

Consider the following language:

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

In the previous problem set, you proved that $L \notin \mathbf{R}$. In this problem, you'll prove that $L \notin \mathbf{RE}$.

Let's suppose for the sake of contradiction that $L \in \mathbf{RE}$. This means that there must be some verifier for the language L . In software, we can express that verifier as a function

```
bool imConvincedIsPasswordChecker(string program, string c)
```

with the following properties:

- This function always returns a value.
- If `program` is a password checker, then there is some choice of `c` where `imConvincedIsPasswordChecker(program, c)` returns true.
- If `program` is not a password checker, then `imConvincedIsPasswordChecker(program, c)` returns false for all choices of `c`.

We can now try to write a self-referential program that uses the above function to cause a contradiction. Here's a first attempt:

```
bool imConvincedIsPasswordChecker(string program, string certificate) {
    /* ... some implementation ... */
}

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

    for (int i = 0 to infinity) {
        for (each string c of length i) {
            if (imConvincedIsPasswordChecker(me, c)) {
                accept();
            }
        }
    }
}
```

This code is, essentially, a minimally-modified version of the self-referential program we used to get a contradiction for the language *LOOP*.

- Suppose that this program is a valid password checker. Briefly explain why running this program leads to a contradiction.
- Suppose that this program is *not* a valid password checker. Briefly explain why running this program does *not* lead to a contradiction.
- Modify the code above to address the deficiency you identified in part (ii). Then, briefly explain why your modified program leads to a contradiction regardless of whether it's a valid password checker.
- Formalize your argument in part (iii) by proving that $L \notin \mathbf{RE}$. Use the proof that *LOOP* is not an **RE** language as a template.

Problem Four: Equivalent TMs (6 Points)

If you've taken CS106A, CS106B, or CS107, you've probably noticed that we have a lot of section leaders and TAs on staff. This is partially so that we can provide lots of one-on-one support and assistance in those courses, but part of it is also due to the fact that it's really hard to grade programming assignments. This question explores why.

When teaching a programming class, it would be really nice if we could fully autograde student programming submissions. Ideally, we'd like to be able to write our own reference solution to one of the programming problems, then check, for each student submission, whether that submission is in some way “equivalent” to our reference solution. If it is, then the submission must be correct, and if it isn't, then the submission must be incorrect.

Let's reformulate this as an equivalent problem about Turing machines. Suppose we have a student-submitted TM M_1 and a reference TM M_2 . We'd like to be able to check whether these TMs have the same languages, that is, whether $\mathcal{L}(M_1) = \mathcal{L}(M_2)$. (This isn't perfectly analogous to our original problem, but it's a close enough match.) We'd like to see whether we can write a TM that can check whether these two TMs have the same language, and, if not, at least whether we can write a TM that checks whether these two TMs have different languages.

Consider the following language EQ_{TM} :

$$EQ_{TM} = \{ \langle M_1, M_2 \rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } \mathcal{L}(M_1) = \mathcal{L}(M_2) \}$$

In other words, EQ_{TM} is the set of all pairs of TMs that have the same language.

It turns out that this language is not in **RE**, meaning that there is no general way to prove that two programs are equivalent. We'd like you to prove this result here.

- i. Suppose there is a function

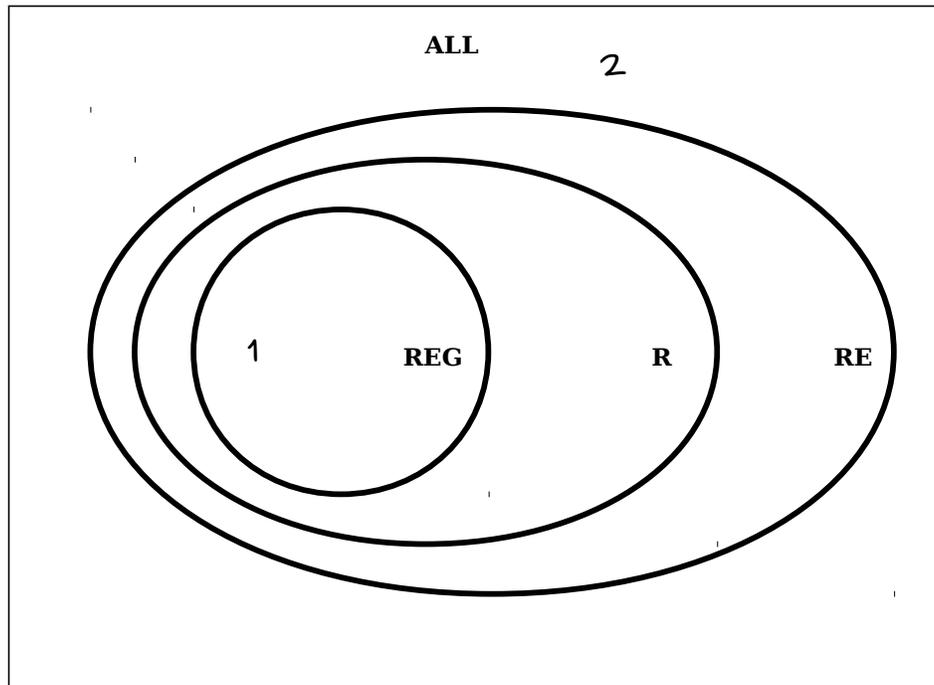
bool imConvincedAreEqual(string p1, string p2, string certificate)

that acts as a verifier for EQ_{TM} . Design a self-referential program that uses this function to obtain a contradiction. (*Hint: Consider passing the program's own source as the first argument and a fixed string containing the text of a program of your choice as the second argument.*)

- ii. Using your program from part (i) as a template, write a formal proof that $EQ_{TM} \notin \mathbf{RE}$. (*A note: if you have a copy of the Sipser textbook, in Chapter Five, Sipser proves that this language is not **RE** using mapping reductions. You're welcome to read over this proof if you'd like, but your answer to this question should use self-reference rather than mapping reductions.*)

Problem Five: The Lava Diagram (10 Points)

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 12 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.



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

(We will cover the material necessary to solve the remaining problems on Monday.)

Problem Six: Resolving $P \stackrel{?}{=} NP$ (8 Points)

This problem explores the question

What would it take to prove whether $P = NP$?

For each statement below, decide whether the statement would definitely prove $P = NP$, definitely prove $P \neq NP$, or would do neither. Write “ $P = NP$,” “ $P \neq NP$,” or “neither” as your answer to each question – we will not award any credit if you write “true” or “false,” since there are three possibilities for each statement. No justification is necessary.

1. There is a P language that can be decided in polynomial time.
2. There is an NP language that can be decided in polynomial time.
3. There is an NP -complete language that can be decided in polynomial time.
4. There is an NP -hard language that can be decided in polynomial time.
5. There is an NP language that *cannot* be decided in polynomial time.
6. There is an NP -complete language that *cannot* be decided in polynomial time.
7. There is a polynomial-time *verifier* for every language in NP .
8. There is a polynomial-time *decider* for every language in NP .
9. There is a language $L \in P$ where $L \leq_p SAT$.
10. There is a language $L \in NP$ where $L \leq_p SAT$.
11. There is a language $L \in NPC$ where $L \leq_p SAT$.
12. There is a language $L \in P$ where $SAT \leq_p L$.
13. There is a language $L \in NP$ where $SAT \leq_p L$.
14. There is a language $L \in NPC$ where $SAT \leq_p L$.
15. All languages in P are decidable.
16. All languages in NP are decidable.

Problem Seven: The Big Picture (6 Points)

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:

$$\mathbf{REG} \subset \mathbf{CFL} \subset \mathbf{P} \stackrel{?}{=} \mathbf{NP} \subset \mathbf{R} \subset \mathbf{RE} \subset \mathbf{ALL}$$

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**. How might you prove it is in **P**?
- iv. Give an example of a language in **NP** that is not known to be in **P**. How might you prove that it is in **NP**? Why don't we know whether it's in **P**?
- 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: $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ (Worth an A+, \$1,000,000, and a Stanford Ph.D)

Prove or disprove: $\mathbf{P} = \mathbf{NP}$.