CS 103: Mathematical Foundations of Computing
Problem Set #1

June 24, 2019

Checkpoint Questions Due Monday, July 1st at 3:00PM Pacific time.
Remaining Questions Due Friday, July 5th at 3:00PM Pacific time.

Here we are – the first problem set of the quarter! This problem set is designed to give you practice writing proofs on a variety of different topics. We hope that this problem set gives you a sense of what proof-based mathematics is like and helps solidify your understanding of set theory.

Before you start this problem set, please do the following:

- Sign up for Piazza so that you have an easy way to ask us questions.
- Review the office hours timetable to find good times to drop on by to ask questions.
- Review Handout "Proofwriting Checklist," for a detailed set of criteria you should apply to your proofs before submitting them. We will be running this same checklist on your proofs when grading, so please be sure to look over it before submitting!

The following are additional resources and handouts that can clarify or supplement your understanding of lecture topics:

- "Guide to $\in$ and $\subseteq$" to make sure you understand the distinction between these terms.
- "Guide to Set Theory Proofs," for more information about how to structure proofs about sets and their properties.
- "Mathematical Vocabulary," which covers mathematical phrases you may need to use in your proofs and how to use them correctly.
- "Guide to Indirect Proofs," which provides some guidance about how to set up proofs by contradiction and contrapositive.
- "Ten Techniques to Get Unstuck," for advice about how to make progress on these sorts of problems when you're not sure what to do.

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

Good luck, and have fun!

Write your solutions to the following checkpoint problems and submit them through GradeScope by Monday at 3:00PM Pacific time. Note in particular that late days may NOT be used on checkpoints. These problems will be graded on a 0 / ✓ / ✓$^+$ scale. Solutions that reasonably attempt to solve all of the problems, even if the attempts are incorrect, will receive a ✓$^+$. Solutions that reasonably attempt some but not all of the problems will receive a ✓. Solutions that do not reasonably attempt any of the problems – or solutions that are submitted after the deadline – will receive a 0. Essentially, if you've made a good, honest effort to solve all of the problems and you submit on time, you should receive full credit even if your solutions contain errors.

Please make the best effort you can when solving these problems. We want the feedback we give you to be as useful as possible, so the more time and effort you put into them, the better we'll be able to comment on your proof style and technique. We will try to get these problems returned to you with feedback on your proof style by Wednesday. Submission instructions are in the "Problem Set Policies" handout.

Checkpoint Problem One: Finding Negations

In order to write a proof by contradiction or contrapositive, you'll need to determine the negation of one or more statements. In Friday's lecture, we talked about a few common classes of statements and how to form their negations. Using what you've learned, answer the following multiple-choice questions and ***briefly explain how you arrived at your answer.***

Which of the following is the negation of "everything that has a beginning has an end?"

A) Everything that does not have a beginning has an end.

B) Everything that has a beginning has no end.

C) There is something that has no beginning and has an end.

D) There is something that has a beginning and has no end.

Which of the following is the negation of "there is a successful person who is grateful?"

A) There is an unsuccessful person who is grateful.

B) There is a successful person who is ungrateful.

C) Every successful person is grateful.

D) Every successful person is ungrateful.

E) Every unsuccessful person is grateful.

F) Every unsuccessful person is ungrateful.

Which of the following is the negation of "if $A \subseteq B$, then $A - B = \varnothing$?"

A) If $A \subseteq B$, then $A - B = \varnothing$.

B) If $A \subseteq B$, then $A - B \neq \varnothing$.

C) If $A \not\subseteq B$, then $A - B = \varnothing$.

D) If $A \not\subseteq B$, then $A - B \neq \varnothing$.

E) There are sets A and B where $A \subseteq B$ and $A - B = \varnothing$.

F) There are sets A and B where $A \subseteq B$ and $A - B \neq \varnothing$.

G) There are sets A and B where $A \not\subseteq B$ and $A - B = \varnothing$.

H) There are sets A and B where $A \not\subseteq B$ and $A - B \neq \varnothing$.

Remember that you need to provide a justification for your answers. While it's not required, ideally you should be able to explain both why your answer is correct ***and*** *why all the other answers are incorrect.*

Checkpoint Problem Two: Set Theory Warmup

This question is designed to help you get used to the notation and mathematical conventions surrounding sets. Consider the following sets:

$$A = \{0, 1, 2, 3, 4\}$$
$$B = \{2, 2, 2, 1, 4, 0, 3\}$$
$$C = \{1, \{2\}, \{\{3, 4\}\}\}$$
$$D = \{1, 3\}$$
$$E = \mathbb{N}$$
$$F = \{\mathbb{N}\}$$

Answer each of the following questions and briefly justify your answers. No proofs are necessary.

i. Which pairs of the above sets, if any, are equal to one another?

ii. Is $D \in A$? Is $D \subseteq A$?

iii. What is $A \cap C$? How about $A \cup C$? How about $A \Delta C$?

iv. What is $A - C$? How about $\{A - C\}$? Are those sets equal?

v. What is $|B|$? What is $|E|$? What is $|F|$?

vi. What is $E - A$? Express your answer in set-builder notation.

vii. Is $0 \in E$? Is $0 \in F$?

Checkpoint Problem Three: Modular Arithmetic

Different numbers can yield the same remainder when divided by some number. For example, the numbers 1, 12, 23, and 34 all leave a remainder of one when divided by eleven. To formalize this relationship between numbers, we'll introduce a relation $\equiv_k$ that, intuitively, indicates that two numbers leave the same remainder when divided by k. For example, we'd say that $1 \equiv_{11} 12$, since both 1 and 12 leave a remainder of 1 when divided by 11, and that $8 \equiv_3 14$, since both 8 and 14 leave a remainder of 2 when divided by 3.
To be more rigorous, we'll formally define $\equiv_k$. For any integer k, define $a \equiv_k b$ as follows:

We say that $a \equiv_k b$ if there exists an integer q such that $a = b + kq$

We'd read the statement "$x \equiv_k y$" aloud as "*x is congruent to y modulo k*." For example, $7 \equiv_3 4$, because $7 = 4 + 3 \cdot 1$, and $5 \equiv_4 13$ because $5 = 13 + 4 \cdot (-2)$. In this problem, you will prove several properties of modular congruence.

i. Prove that for any integer x and any integer k that $x \equiv_k x$.

Be careful not to assume what you need to prove. Don't start your proof by assuming there's a choice of q where $x = x + kq$ and then solving for q. If you assume there's an integer q where $x = x + kq$, you're already assuming that $x \equiv_k x$! Look at the proofs we did in lecture with odd and even numbers as an example of how to prove that there is a number with a specific property without making any unfounded assumptions.

ii. Prove that for any integers x and y and any integer k that if $x \equiv_k y$, then $y \equiv_k x$.

Keep an eye out for your variable scoping in the above proof. Make sure you introduce the variables x, y, and k before you use them. Are they chosen arbitrarily? Do they represent specific values?

iii. Prove that for any integers x, y, and z and any integer k that if $x \equiv_k y$ and $y \equiv_k z$, then $x \equiv_k z$.

Modular congruence is a powerful mathematical tool. You'll use it later in this problem set to build a better understanding of star-drawing!

The rest of these problems are due on Friday at 3:00PM.
Please type your solutions and submit them on GradeScope.

1 Much Ado About Nothing

It can take a bit of practice to get used to the empty set. This problem will ask you to think about a few different sets related to $\varnothing$.

Go to the CS103 website and download the starter files for Problem Set One. Unpack the files somewhere convenient and open up the bundled project. Answer each part of this question by editing the relevant resource files (they're in the `res/` directory). There's information in the top of each of the files about how to represent sets; most importantly, note that to write out the empty set, you should write {} rather than using the empty-set symbol. For example, the set $\{\varnothing\}$ would be written as {{}}.

i. Edit the file `PartI.object` so that it contains a set equal to $\varnothing \cup \{\varnothing\}$.

ii. Edit the file `PartII.object` so that it contains a set equal to $\varnothing \cap \{\varnothing\}$.

iii. Edit the file `PartIII.object` so that it contains a set equal to $\{\varnothing\} \cup \{\{\varnothing\}\}$.

iv. Edit the file `PartIV.object` so that it contains a set equal to $\{\varnothing\} \cap \{\{\varnothing\}\}$.

v. Edit the file `PartV.object` so that it contains a set equal to $\wp(\wp(\varnothing))$.

vi. Edit the file `PartVI.object` so that it contains a set equal to $\wp(\wp(\wp(\varnothing)))$.

The starter code contains a driver program you can use to see the contents of your files and confirm they're syntactically correct. Submit your answers through GradeScope under "Coding Problems for Problem Set One" by uploading your edited starter files. You're welcome to submit as many times as you'd like.

2 Set Theory in C++

The C++ standard library contains a type called `std::set` that represents a set of elements, all of which must be of the same type. For example, the type `std::set<int>` represents a set of integers, the type `std::set<std::string>` represents a set of strings, and `std::set<std::set<int>>` is a type representing a set of sets of integers.

There are all sorts of operations you can perform on `std::sets`. For example, here's how you iterate over all the elements of a set:

```
std::set<T> mySet;
for (T elem: mySet) {
    /* ... do something with the current element elem ... */
}
```

Here's how you check whether a particular value is an element of a set:

```
if (mySet.count(value)) {
    /* ... value ∈ mySet ... */
} else {
    /* ... value ∉ mySet ... */
}
```

And, finally, here's how you can get the cardinality of a set:

```
size_t size = mySet.size();
```

Here, the `size_t` type is a type representing a natural number, since sets can't have negative size. (The folks who designed the C++ standard libraries had a strong discrete math background.)

One of the major differences between the sets we've talked about in CS103 and the `std::set type` is that in discrete mathematics, sets can contain anything – numbers, philosophies, recipes, other sets, etc. – but in C++ all objects in a set must have the same type. For the purposes of this problem, we've created a custom C++ type called `Object`. Variables of type `Object` can represent just about anything, so a `std::set<Object>` represents something pretty similar to the sets we've been studying so far.

Some `Objects` are actually just `std::sets` in disguise. If you have an `Object`, you can test whether it's actually a set by using this provided helper function:

```
bool isSet(Object o);
```

This takes in an `Object`, then returns true if that `Object` is actually a set and false otherwise. If you have an `Object` that really is a set, you can convert it to a set by using this helper function:

```
std::set<Object> asSet(Object o);
```

This function takes in an `Object` that you know happens to be a set, then returns the `std::set<Object>` that it actually is.

For example, suppose you have an `Object` that you know is really the set $\{1, 2, 3, 4\}$. You could iterate over it using this code:

```
Object reallyASet = /* ... */;
for (Object x: asSet(reallyASet)) {
    /* ... do something with x ... */
}
```

In this problem, we'd like you to demonstrate your understanding of sets and set theory by coding up a number of functions in C++ that operate on sets. In doing so, we hope that you'll solidify your grasp of the distinctions between related concepts in set theory, such as the $\in$ and $\subseteq$ relations and power sets.

Open the file `SetTheory.cpp` from the starter files. There, you'll find a bunch of stubs of functions that you'll need to implement. The provided starter code contains a test harness you can use to try out your functions. You won't need to modify any of the other C++ files bundled with the starter code.

As with Problem One, you'll submit the code that you write through GradeScope separately from the rest of the problems on this problem set. The GradeScope autograder will get back to you with feedback about how you're doing on this problem, and you're welcome to submit as many times as you'd like.

i. Implement a function

```
bool isElementOf(Object S, Object T);
```

that takes as input two `Objects` S and T, then returns whether $S \in T$.

S and T might not be sets; you'll need to use the isSet and asSet functions appropriately.

ii. Implement a function

```
bool isSubsetOf(Object S, Object T);
```

that takes as input two `Objects` S and T, then returns whether $S \subseteq T$.

S and T might not be sets; use the isSet predicate to check whether the appropriate arguments are sets and asSet to get a view of them as sets.

iii. Implement a function

```
bool areDisjointSets(Object S, Object T);
```

that takes as input two `Objects` S and T, then returns whether S and T are sets where $S \cap T = \varnothing$. (Two sets with this property are called ***disjoint***.) The input parameters S and T may or may not be sets, and if they aren't, your function should return false.

iv. Implement a function

```
bool isSingletonOf(Object S, Object T);
```

that takes as input two `Objects` S and T, then returns whether $S = \{T\}$. Again, S and T may or may not be sets.

v. Implement a function

```
bool isElementOfPowerSet(Object S, Object T);
```

that takes as input two `Objects` S and T, then returns whether S and T are sets and $S \in \wp(T)$. Again, S and T may or may not be sets.

As a hint, you shouldn't need to write code that computes $\wp(T)$ explicitly. See if you can find a different way to do this.

vi. Implement a function

```
bool isSubsetOfPowerSet(Object S, Object T);
```

that takes as input two `Objects` S and T, then returns whether S and T are sets and $S \subseteq \wp(T)$. Again, S and T may or may not be sets.

vii. Implement a function

```
bool isSubsetOfDoublePowerSet(Object S, Object T);
```

that takes as input two `Objects` S and T, then returns whether S and T are sets and $S \subseteq \wp(\wp(T))$. Again, S and T may or may not be sets.

To submit your work, upload your edited starter files to GradeScope. You'll get immediate feedback on your score from our autograder, and you're allowed to resubmit as many times as you'd like.

3 Describing the World in Set Theory

The notation of set theory (e.g. $\cup, \cap, \wp, \subseteq, \in$, etc.) is a great tool for describing the real world. Answer each of the following questions by writing an expression using set theory notation, but ***without*** using plain English, ***without*** using set-builder notation, ***without*** introducing any new variables, and ***without*** using propositional or first-order logic (which we'll cover next week).

i. Let's have C be the set of US citizens, S the set of people who live in a US state, M be the set of all people eighteen and older, and V be the set of people who are allowed to vote in US presidential elections. Write an expression that says that every US citizen age eighteen and older who lives in a US state can vote in a US presidential election.

*Once you've written up your answer to this problem, take a minute to **type-check** it. As an example, suppose that you have the following answer:*

$$(C \in M) \cap (V \in M)$$

This expression can't possibly be right, and here's one way to see this. The expression $C \in M$ is of type boolean – either $C \in M$ is true or it isn't – and the same is true of $V \in M$. However, the intersection operator $\cap$ can only be applied to sets. The expression therefore contains a type error: it tries to apply an operator that only works on sets to boolean values.

ii. Suppose you're on an exciting first date. Let Y represent your hobbies and D represent your date's hobbies. Write an expression that says that you have a hobby that your date doesn't have.

You can type-check this answer in a different way. For example, suppose you came up with this expression:

$$Y \cup D$$

Here, Y and D are sets, so it's perfectly fine to write $Y \cup D$, which evaluates to an object of type set. But notice that the statement here asks you to write an expression that says "you have a hobby that your date doesn't have," and that statement is essentially of type boolean (you either do or do not have a hobby your date doesn't have). Therefore. $Y \cup D$ can't possibly be an expression with the right meaning, since the type of the expression (set) doesn't match the type of the statement (boolean).

iii. The song "I am Moana" from the movie *Moana* starts off with the following lyrics:

"I know a girl from an island / She stands apart from the crowd."

Let K be the set of all people I know, G be the set of all girls, I be the set of all people from an island, and C be the set of all people who stand apart from the crowd. Write an expression that expresses the above lyric using the notation of set theory.

iv. Let's say that a ***committee*** is a group of people, which we can think of as being represented by the set of people that comprise it. Let's have S represent the set of all students at Stanford and let F represent the set of all faculty at Stanford. Write an expression representing the set of all committees you can make from Stanford students and faculty that contain at least one student and at least one faculty member. You can assume no one is both a student and a faculty member.

Something to think about: how would you say "all committees made purely of students?"

4 Proof Critiques

One of the best ways to improve your proof *writing* is to do some proof *reading*. In doing so, you'll get a much better perspective on why certain stylistic conventions matter, both in the positive sense, ("wow, that's so easy to read!") and in the negative sense ("I can't make heads or tails of this!"). You'll also get practice tracing through mathematical arguments, which will help expand your repertoire of techniques and give you a better sense of what details to focus on in your own reasoning.
We've curated three proofs for you to review. Read these proofs, and for each one, do the following:

- ***Critique its style.*** Our Proofwriting Checklist contains specific details to review in any proof. Go through the checklist one item at a time, identifying any style errors and explaining each.
- ***Critique its reasoning.*** Are the arguments given by these proofs correct? If there are logic errors, point them out and demonstrate why they're incorrect. Then, determine whether the theorem being proved is even true in the first place. After all, you can prove anything using faulty logic!
- ***If possible, rewrite the proof.*** You now have a list of issues. Based on what you've found, do one of the following:
  - If the reasoning is correct but the proof has poor style, simply rewrite the proof to improve its style, keeping the core argument intact.
  - If the reasoning is *incorrect* but the statement being proved is still true, write a new proof of the overall theorem. Try to modify the original argument as little as possible.
  - If the reasoning is *incorrect* and the statement being proved isn't even true to begin with, briefly explain why the statement isn't true, though no formal disproof is required.

You should submit both your notes from the initial critique and your newly-revised proofs.

i. Critique this proof about parities (the ***parity*** of a number is whether it's even or odd.)

Theorem. The sum of an even integer and an odd integer is odd.

Proof. This proof will talk about adding together different kinds of numbers. An even integer is an integer that can be written as $2k$ for some integer k. Therefore, $m = 2k$. Similarly, an odd integer is one that can be written as $2k + 1$ for some integer k. So $n = 2k + 1$. $m + n = 2k + 2k + 1 = 4k + 1$. Therefore $m + n$ is odd. ■

ii. Critique this proof about natural numbers.

Theorem. Every natural number is odd.

Proof. Assume for the sake of contradiction that every natural number is even. In particular, that would mean that 137 is even. Since $137 = 2 \cdot 68 + 1$ and 68 is a natural number, we see that 137 is odd. We know that there is no integer n where n is both odd and even. However, $n = 137$ is both even and odd. This is impossible. We've reached a contradiction, so our assumption must have been wrong. Therefore, every natural number is odd. ■

iii. Critique this proof about modular arithmetic.

Theorem. If n is an integer and $n \equiv_2 0$, then $n \equiv_4 0$.

Proof. We will prove the contrapositive of this statement and show that if n is an integer where $n \equiv_4 0$, then $n \equiv_2 0$. Since $n \equiv_4 0$, we know $n = 0 + 4q$. This in turn tells us that $n = 2(2q)$, so there is an integer m such that $n = 2m$. Consequently, we see that $n = 0 + 2m$, and so $n \equiv_2 0$, as required. ■

5 Modular Arithmetic, Part II

The modular congruence relation you saw in the checkpoint problem has a lot of nice properties. As you saw, in many ways, it acts like regular old equality. How far does that connection extend?
Below are two statements. For each true statement, write a proof that the statement is true. For each false statement, write a disproof of the statement (take a look at the Proofwriting Checklist for information about how to write a disproof.) You can use any proof techniques you'd like.

i. Prove or disprove: for any integers x, y, z, and k, if $x \equiv_k y$ then $xz \equiv_k yz$.

This is your first example of a "prove or disprove" problem. Your first task is to figure out whether it's true or false, since if it's true you'll want to prove it and if it's false you'll want to disprove it.

Here are two strategies for approaching problems like these. First, try out a lot of examples! You'll want to get a feel for what the symbolic expression above "feels" like in practice. Second, get a sheet of scratch paper and write out both the statement and its negation. One of those statements is true, and your task is to figure out which one it is. Once you have those two statements, think about what you would need to do to prove each of them. In each case, what would you be assuming? What would you need to prove? If you can answer those questions, you can explore both options and seeing which one ends up panning out.

Finally, remember to call back to definitions! Regardless of whether you're proving or disproving this, you'll want to refer to the formal definition of modular congruence from the checkpoint problem.

ii. Prove or disprove: for any integers x, y, z and k, if $z \neq 0$ and $xz \equiv_k yz$, then $x \equiv_k y$.

6 Properties of Sets

For each of these statements about sets, decide whether that statement is true or false. Prove each true statement, and disprove each false statement.

We ***strongly*** recommend reading the Guide to Set Theory Proofs before starting this problem.

i. Prove or disprove: for all sets A, B, and C, if $A \in B$ and $B \in C$, then $A \in C$.

ii. Prove or disprove: if A, B, and C are sets where $A - C = B - C$, then $A = B$.

A question you should be able to answer before starting this problem: how do you prove that two sets are equal to one another? Based on that, how do you prove that two sets are not equal to one another? Check the Guide to Proofs on Sets for details.

iii. Prove or disprove: if A and B are sets where $\wp(A) = \wp(B)$, then $A = B$.

iv. Prove or disprove: if A, B, and C are sets where $A \Delta C = B \Delta C$, then $A = B$.

Before you turn in these proofs, read over the Proofwriting Checklist and the Guide to Proofs on Sets and confirm that your proof meets our style criteria. Here are a few specific things to look for:

- *Make sure that the structures of your proofs match the definitions of the relevant terms. To prove $S \subseteq T$, pick an arbitrary $x \in S$, then prove $x \in T$ by making claims about x. To prove $S = T$, prove $S \subseteq T$ and $T \subseteq S$.*
- *However, avoid restating definitions in the abstract. For example, rather than writing*

  "We know that $S \subseteq T$ if every element of S is an element of T.
  Therefore, since we know that $A \subseteq B$ and $x \in A$, we see that $x \in B$."

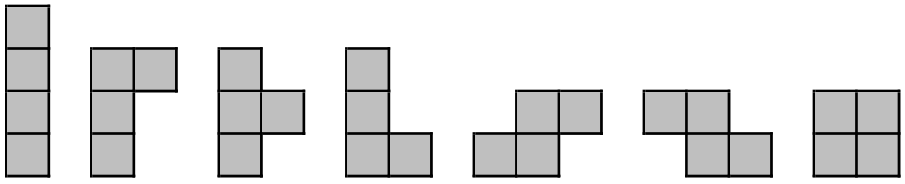
  instead remove that first sentence and just write something like this:

  "Since $x \in A$ and $A \subseteq B$, we see that $x \in B$."

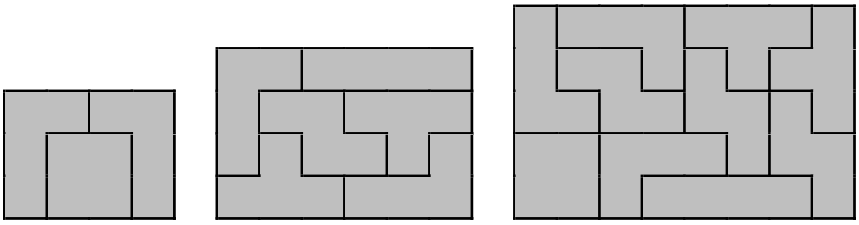
  *Whoever is reading your proof knows all the relevant definitions. They're more interested in seeing **how those definitions interact with one another** than **what those definitions are**.*
- *Make sure you clearly indicate what each variable means and whether it's chosen arbitrarily or chosen to have a specific value. For example, if you refer to variables like A, B, or C, you should clearly indicate whether they're chosen arbitrarily or refer to specific values.*
- *When talking about an arbitrary set A, it's tempting to list off the elements of A by writing something like $A = \{x_1, x_2, \ldots, x_n\}$. The problem is that writing $A = \{x_1, x_2, \ldots, x_n\}$ implicitly says that the set A is finite, since you're saying it only has n elements in it. This is a problem if A is infinite. In fact, if A is infinite, because of Cantor's theorem, you can't necessarily even write $A = \{x_1, x_2, x_3, \ldots\}$, since you might run out of natural numbers with which to name the elements of A!*

7 Tiling with Polyominoes

The game *Tetris* involves working with seven shapes of tiles, each of which is made from exactly four squares. The pieces are shown here:



We can use the Tetris pieces to ***tile*** various geometric figures. In the context of recreational mathematics (yes, that's a thing!), "tiling" refers to covering up some underlying figure using tiles drawn from some collection of shapes. You're allowed to rotate and flip the tiles however you'd like, and you can assume you have as many copies of each tile as you need. You don't even need to use all the tiles. However, the tiles must cover the entire underlying figure, the tiles can't hang off of the original figure, and the tiles aren't allowed to overlap. For example, here's how we'd tile a 3×4 chessboard, a 4×6 chessboard, and a 5×8 chessboard with Tetris pieces:



Mathematicians and computer scientists have spent a lot of time thinking about what sorts of tilings are possible, and what sorts of tilings are impossible. For example, in the course notes (pages 60 - 61), there's a proof that it's impossible to tile an 8×8 chessboard missing two opposite corners using 2×1 tiles.

i. Prove or disprove: it's possible to tile a 9×10 chessboard with Tetris pieces.

Up to this point, we've assumed you have as many copies of each Tetris piece as you'd like. There's a more restricted kind of tiling called a perfect tiling in which you get exactly one copy of each tile, and you're required to form a tiling using exactly one copy of each tile. None of the above tilings are perfect.

ii. Prove or disprove: it's possible to perfectly tile a 4×7 chessboard with the seven Tetris pieces.

8 Yablo's Paradox

A ***logical paradox*** is a statement that results in a contradiction regardless of whether it's true or false. One of the simplest paradoxes is the ***Liar's paradox***, which is the following:

This statement is false.

If this statement is true, then by its own admission, it must be false – a contradiction! On the other hand, if the above statement is false, then the statement "This statement is false" is false, and therefore the statement "This statement is false" is true – a contradiction! Since this statement results in a contradiction regardless of whether it's true or false, it's a paradox.

Paradoxes often arise as a result of self-reference. In the Liar's Paradox, the paradox arises because the statement itself directly refers to itself. However, it's not the only paradox that can arise from self-reference. This problem explores a paradox called ***Yablo's paradox***.

Consider the following collection of infinitely many statements numbered $S_0, S_1, S_2, \ldots$, where there is a statement S_n for each natural number n. These statements are ordered in a list as follows:

(S_0): All statements in this list after this one are false.
(S_1): All statements in this list after this one are false.
(S_2): All statements in this list after this one are false.
...

More generally, for each $n \in \mathbb{N}$, the statement (S_n) is

(S_n): All statements in this list after this one are false.

Surprisingly, the interplay between these statements makes every statement in the list a paradox.

i. Prove that if any statement in this list is true, it results in a contradiction.

Your result needs to work for any choice of statement in the list, not just one of them. Follow the template for proving a universally-quantified statement

ii. Prove that every statement in this list is a paradox.

Something to ponder: how do you negate a universally-quantified statement?

Now, consider the following modification to this list. Instead of infinitely many statements, suppose that there are "only" $10,000,000,000$ statements. Specifically, suppose we have these statements:

(T_0): All statements in this list after this one are false.
(T_1): All statements in this list after this one are false.
(T_2): All statements in this list after this one are false.
...
$(T_{9,999,999,999})$: All statements in this list after this one are false.

There's still a lot of statements here, but not infinitely many of them. Interestingly, these statements are all perfectly consistent with one another and do not result in any paradoxes.

iii. For each statement in the above list, determine whether it's true or false and explain why your choices are consistent with one another.

Going forward, don't worry about paradoxical statements in CS103. We won't talk about any more statements like these. ☺

9 The Star-Drawing Saga, Part II

On Problem Set 0, you got to play around with star drawing and discovered that certain combinations of a number of points p and a step size s led to simple stars (for example, the {7 / 2} and {8 / 3} stars), while other combinations did not (for example, the {8 / 2} or {12 / 3} stars). Why exactly is this? Over the course of the next several problem sets, you'll discover exactly why!

The main insight that we'll use in analyzing stars is to recognize a connection between star-drawing, which you saw on Problem Set Zero, and modular congruence, which you explored in this problem set. Imagine that you set out to draw a p-pointed star by drawing p points around a circle. Let's number those points $0, 1, 2, \ldots, p-1$, going clockwise around the circle. (Of course we're using zero-indexing - we're computer scientists!) We'll then draw the star by starting at position 0, then drawing a line to the point s positions clockwise from us, then drawing a line from there to the point s positions clockwise from that, etc. until we end back up at position 0.

i. Let's suppose we're following the above procedure and stop after drawing t total lines and end up on point n on the star. Explain, intuitively, why $n \equiv_p s \cdot t$.

Mathematicians conventionally use single-letter variable names for everything, which can lead to really dense formulas. Computer scientists love to use longer variable names to improve readability. If you're having trouble interpreting that above formula, consider writing a "cheat sheet" that lists each variable along with a more verbose description of what it represents.

If we pick a number of points p and a step size s, we end up with a simple star if we end up passing through each of the p points before ending back at our starting point. The modular equation you just saw in part (i) of this problem tells us where on the star we'll be at each point in time. Putting these two ideas together gives us this fundamental fact about stars:

> ***Simple Star Criterion***: A p-point star with a step size of s is a simple star if and only if for every natural number n, there is a natural number t such that $n \equiv_p s \cdot t$.

This statement is slightly dense, and like all mathematically dense statements, one of the best ways to get a handle on what it says is to try it out in some specific cases.

ii. Consider a star polygon with five points and a step size of two (the {5 / 2} star, or the standard 5-pointed star). This is an example of a simple star – you are able to hit all 5 points before ending up back at your starting point.

Now consider the {7 / 2} star, {9 / 2} star, and {11 / 2} star. Show that these stars also meet the simple star criterion by filling in the table below, listing off values of t such that $n \equiv_p s \cdot t$. We have filled in the first row for the {5 / 2} star for you.

n	***0***	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***
Value of t ***where*** $n \equiv_5 2 \cdot t$	0	3	1	4	2						
Value of t ***where*** $n \equiv_7 2 \cdot t$											
Value of t ***where*** $n \equiv_9 2 \cdot t$											
Value of t ***where*** $n \equiv_{11} 2 \cdot t$											

It's somewhat tricky to fill this table in purely by eyeballing the mathematical formula and trying to guess numbers. Instead, think back to the intuition behind this formula. You might want to try drawing the star and seeing whether that gives you any new insights.

One observation you may have made between Problem Set 0 and filling out the table above is that if you have an odd number of points, you can always form a simple star by taking steps of size two.

iii. Using the simple star criterion, prove that $\{p \ / \ 2\}$ is a simple star for any odd natural number p.

This result is a generalization of what you did in part (ii) of this problem, since that question asked you to work out the math for $p = 7, 9, 11$ and this question asks for a more general argument. You might want to try solving this problem by looking at the table you've generated and seeing if you spot a pattern. Feel free to try other concrete values of p as well. Once you have the pattern, formalize your reasoning by writing it up as a proof.

On each problem set, we'll provide some optional fun problems for extra credit. These are problems that can be solved purely using the techniques you've learned so far, but which will require some thought. Each problem, in our opinion, has a beautiful solution that will give you a much deeper understanding of core concepts, so we strongly encourage you to play around with them if you get the chance. Please feel free to work on as many of these problem as you'd like, though please only submit a solution to at most one of the optional fun problems on each problem set.

When we compute final grades at the end of the quarter, we compute the grading curve without any extra credit factored in, then recompute grades a second time to factor in extra credit. This way, you're not at any disadvantage if you decide not to work through these problems. If you do complete the extra credit problems, you may get a slight boost to your overall grade.

As a matter of course policy, we don't provide any hints on the extra credit problems – after all, they're supposed to be challenge problems! However, we're happy to chat about them after the problem sets come due.

Optional Fun Problem One : Hat Colors

You and nine of your friends are brought into a room together. The CS103 TAs go around and place a hat on everyone's head. There are ten different colors of hats – red, orange, yellow, green, blue, magenta, black, white, gray, and gold. It's possible that everyone gets a hat of a different color. It's possible that everyone gets a hat of the same color. In fact, aside from the restriction that each hat is one of the ten colors mentioned earlier, there are no restrictions about how many hats there are of each color.

You and your friends can all look around to see what color hats everyone else is wearing, but no one can see their own hat color. The staff will then go around and ask everyone to write down a guess of what color their hat is. We'll then collect those guesses and reveal them all at once. You and your friends succeed if at least one of you is able to correctly guess their own hat color. You and your friends all lose if no one correctly guesses their own hat color.

Before we place hats on everyone, we'll allow you and your friends to work out a strategy about how you're going to guess, but once the hats are on no communication of any kind - whether explicit or implicit - is permitted between the group. Devise a strategy that *guarantees* at least one person will correctly guess their own hat color, even if we know what strategy you'll be using. Then, using the mathematical notation and tools you've learned so far in this course, formally prove that your strategy works. For example, here are a few strategies you could choose from that *don't* work:

- Everyone could pick which color they're going to guess before going into the room, and then write down that guess regardless of what color hats they see on everyone else. If we assign hats randomly, then the probability that this works is approximately $1 - e^{-1}$ (take CS109 if you're curious why this is!) However, if we overhear your discussion, we could be mean and deliberately give everyone the wrong hat color, making your probability of success exactly 0.
- Everyone could arrange themselves in a circle, then guess the color of the hat of the person immediately to their left. If we choose hat colors randomly, then there's a decent probability that someone will correctly guess their own hat color. But if we know that this is what you're going to do, we can just give everyone a different hat color and you're guaranteed to lose

I first heard this problem from Michelle McGhee, CS major (theory track), storyteller, and Ultimate Frisbee player, when she was helping staff Girl Code @Stanford in 2018. Her senior project was a program that generated freestyle rap lyrics.

Optional Fun Problem Two: Infinite Deviation

In our first class meeting, we sketched out a proof of Cantor's theorem. If you'll recall, this proof worked by assuming there was a pairing between the elements of a set S and the subsets of that set S, then constructing a set that was different in at least one position from each of the paired sets.

Suppose you try to pair off the elements of $\mathbb{N}$ with the subsets of $\mathbb{N}$. Show that, no matter how you do this, there's always at least one set $X \subseteq \mathbb{N}$ that differs in infinitely many positions from each of the paired sets. Justify your answer, but no formal proof is necessary.