Problem Set 1

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 #07, "Set Theory Definitions," for a refresher on key terms, definitions, and theorems about set theory that might come up in this problem set.
- Review Handout #08, "Guide to Proofs," which has advice about how to write and structure your proofs.
- Review Handout #09, "Mathematical Vocabulary," which covers mathematical phrases you may need to use in your proofs and how to use them correctly.
- Review Handout #10, "Guide to Indirect Proofs," which provides some guidance about how to set up proofs by contradiction and contrapositive.
- Review Handout #11, "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.
- Review Handout #12, "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!***
- Review the online "Guide to $\in$ and $\subseteq$" to make sure you understand the distinction between these terms.

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!

Checkpoint Questions Due Monday, October 2nd at 2:30PM Pacific time.
Remaining Questions Due Friday, October 6th at 2:30PM Pacific time.

Write your solutions to the following checkpoint problems and submit them through GradeScope by Monday, October 2nd at 2:30PM Pacific time. These problems will be graded on a 0 / 1 / 2 scale. Solutions that reasonably attempt to solve all of the problems, even if the attempts are incorrect, will receive two points. Solutions that reasonably attempt some but not all of the problems will receive one point. Solutions that do not reasonably attempt any of the problems – or solutions that are submitted after the deadline – will receive zero points.

Essentially, if you've made a good, honest effort to solve all of the problems and you submit on time, you should receive two points even if your solutions contain errors.

Please make the best effort you can when solving these problems. We want the feedback we give you on your solutions 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, October 4th. Submission instructions are included 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 $A - B \neq \varnothing$.

Checkpoint Problem Two: Multiples of Three

In class, we talked a fair amount about odd and even numbers, which arise when dividing numbers in half. This question generalizes the idea of "even" and "odd" to similar terms that arise when dividing by three.

An integer is called a ***multiple of three*** if it can be written as $3k$ for some integer k. An integer is ***congruent to one modulo three*** if it can be written as $3k + 1$ for some integer k, and an integer is ***congruent to two modulo three*** if it can be written as $3k + 2$ for some integer k. For each integer n, exactly one of the following is true (you don't need to prove this):

- n is a multiple of three.
- n is congruent to one modulo three.
- n is congruent to two modulo three.

We'd like you to prove this result:

For every integer n, n is a multiple of three if and only if n^2 is a multiple of three.

To do this, we'll have you prove the following two statements:

For any integer n, if n a multiple of three, then n^2 is a multiple of three.

For any integer n, if n^2 is a multiple of three, then n is a multiple of three.

We've broken this question down into a few parts.

i. Prove the first of these statements with a direct proof.

ii. Prove the second of these statements using a proof by contrapositive. Make sure that you state the contrapositive of the statement explicitly before you attempt to prove it.

iii. Prove, by contradiction, that $\sqrt{3}$ is irrational. Make sure that you explicitly state what assumption you are making before you derive a contradiction from it. Recall from lecture that a rational number is one that can be written as p / q for integers p and q where $q \neq 0$ and p and q have no common divisor other than ± 1.

The remainder of these problems should be completed and submitted through GradeScope by Friday, October 6th at 2:30PM.

Problem One: Set Theory Warmup

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

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

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.

Problem Two: The Power Set Revisited

In our first lecture, we saw an operation called the power set that, given a set S, produces a set $\wp(S)$ consisting of all the subsets of the set S. Why didn't we introduce an operation that, given a set S, produces a set consisting of all the elements of S?

Problem Three: 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 the same type. For example, the type `std::set<int>` represents a set of integers, a `std::set<std::string>` represents a set of strings, and a `std::set<std::set<int>>` is a set of sets of integers.

There are all sorts of operations you can perform on `std::set`s. 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 size 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 implemented the C++ standard libraries had a strong discrete math background.)

One of the major differences between the sets that we've been talking about in CS103 and the `std::set` type is that in discrete mathematics, sets can contain anything – numbers, philosophical concepts, 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 `Object`s are actually just `std::set`s 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 the ∈ and ⊆ relations and power sets.

(Continued on the next page)

Go to the CS103 website and download the starter files for Problem Set One. Unpack the files somewhere convenient, open up the bundled project, and load the file `SetTheory.cpp`. There, you'll find a bunch of stubs of functions that you'll need to implement. You won't need to modify any of the other files bundled with the starter code, though if you're curious you're welcome to read over them.

You'll submit the code that you write through GradeScope separately from the rest of the problems on this problem set. The GradeScope autograder should 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 `Object`s `S` and `T`, then returns whether $S \in T$. Note that `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 an object `S` and an object `T`, then returns whether $S \subseteq T$. Note again `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***.) Again, 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.

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 `SetTheory.cpp` file to GradeScope. You'll get immediate feedback on your score from our autograder.

As a hint for parts (v), (vi), and (vii) – you should not need to explicitly compute any power sets.

Problem Four: 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 Ø. We'd like you to submit your answers electronically by editing some of the resource files (they're in the `res/` directory) that were shipped along with the starter code for this problem set. There's information in the top of each of the files about how to represent sets in those files; most importantly, note that to write out the empty set, you should write `{}` rather than using the empty-set symbol. For example, the set {Ø} would be written as `{{}}`.

i. Edit the file `PartI.object` so that it contains a set equal to Ø ∪ {Ø}.

ii. Edit the file `PartII.object` so that it contains a set equal to Ø ∩ {Ø}.

iii. Edit the file `PartIII.object` so that it contains a set equal to {Ø} ∪ {{Ø}}.

iv. Edit the file `PartIV.object` so that it contains a set equal to {Ø} ∩ {{Ø}}.

v. Edit the file `PartV.object` so that it contains a set equal to ℘(℘(Ø)).

vi. Edit the file `PartVI.object` so that it contains a set equal to ℘(℘(℘(Ø))).

To submit your work, upload all six of these files to GradeScope. (Don't forget to also upload the file `SetTheory.cpp` from Problem Three!)

Problem Five: Properties of Sets

Here are some claims about properties of sets. Some of them are true and some of them are false. For each true statement, write a proof that the statement is true. For each false statement, write a disproof of the statement. You can use any proof techniques you'd like.

If you want to disprove a statement, you can structure your disproof like this:

> ***Claim:*** [*statement to disprove goes here*].
>
> ***Disproof:*** We will show that the negation of this statement, namely, [*fill this in*], is true. [*Then go prove the negation of the statement*]. ■

In particular, notice that you shouldn't call the statement you're disproving a theorem, since the term "theorem" specifically refers to a statement that you'll prove is true. Similarly, you shouldn't call the justification you're writing up a proof, since the term "proof" in mathematics means an argument that shows something is true, and that's precisely the opposite of what you'll be doing. Aside from these terminology differences, proofs and disproofs follow the same general structure: you'll give a line of reasoning establishing why some result happens to be true. The only difference is that in a disproof, you're proving that the negation of some statement is true.

i. For all sets A, B, and C, if $A \in B$ and $B \in C$, then $A \in C$.

ii. For all sets A and B, if $\wp(A) = \wp(B)$, then $A = B$.

iii. For all sets A, B, and C, if $A \subseteq B$ and $A \subseteq C$, then $A \subseteq B \cap C$.

iv. For all sets A, B, and C, if $A \subsetneq B$ and $A \subsetneq C$, then $A \subsetneq B \cap C$. (The notation $A \subsetneq B$ says that A is a ***strict subset*** of B, meaning that $A \subseteq B$ and $A \neq B$.)

v. There exists a set A where $\wp(A) = \{A\}$.

Some general notes on these above proofs:

- Make sure to call back to definitions. For example, if you're trying to prove that a set S is a subset of a set T, follow the pattern from lecture: pick an arbitrary $x \in S$, then prove that $x \in T$.
- Be careful with your variables. Make sure you clearly indicate what each variable means and whether it's chosen arbitrarily or chosen to have a specific value.

Problem Six: Two Is Irrational?

In lecture, we proved that $\sqrt{2}$ is irrational, and in the checkpoint problem you proved that $\sqrt{3}$ is irrational. Below is a purported proof that $\sqrt{4}$ is irrational:

> ***Theorem*:** $\sqrt{4}$ is irrational.
>
> ***Proof:*** Assume for the sake of contradiction that $\sqrt{4}$ is rational. Then there must exist integers p and q where $q \neq 0$, where $p / q = \sqrt{4}$, and where p and q have no common factors other than 1 and -1.
>
> Starting with $p / q = \sqrt{4}$ and squaring both sides tells us that $p^2 / q^2 = 4$. We can then cross-multiply by q^2 to see that $p^2 = 4q^2$. Since q^2 is an integer and $p^2 = 4q^2$, we see that p^2 is a multiple of four, and therefore that p is a multiple of four. This tells us that $p = 4n$ for some integer n.
>
> Since $4q^2 = p^2$ and $p = 4n$, we can use some algebraic substitutions to show that $4q^2 = (4n)^2 = 16n^2$, so $q^2 = 4n^2$. Since n^2 is an integer and $q^2 = 4n^2$, we see that q^2 is a multiple of four, so q is a multiple of four as well. But since both p and q are multiples of four, we see that p and q share a common divisor other than ±1, contradicting our initial assumption. We have reached a contradiction, so our assumption must have been incorrect. Thus $\sqrt{4}$ is irrational. ■

This proof has to be wrong, because $\sqrt{4} = 2 = {}^2/_1$, so it is indeed rational!

What error does this proof make that lets it conclude $\sqrt{4}$ is irrational? Be specific.

Problem Seven: Piano Tuning

At a first glance, irrational numbers can seem like a purely mathematical idea without any practical applications. It turns out, though, that irrational numbers have real-world implications.

Prove that $\sqrt[12]{2}$, the twelfth root of two, is irrational. Interestingly, this result means that it's impossible to tune a piano such that every half step, perfect fifth, perfect fourth, and octave are all properly in tune. If you're curious about why this is, there's a great Minute Physics video about the different ratios that arise in music and how this relates to them. It's online at https://youtu.be/1Hqm0dYKUx4.

As a hint, do *not* attempt to prove this result by starting with the proof that $\sqrt{2}$ is irrational and making appropriate modifications – that will get really messy really fast. Instead, see if you can prove the following intermediary result, and build your proof around it:

If $\sqrt[12]{2}$ is rational, then $\sqrt{2}$ is rational.

You may want to check the Mathematical Prerequisites handout for a refresher on higher-order roots.

Problem Eight: 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 $a - b = kq$

For example, $7 \equiv_3 4$, because $7 - 4 = 3 = 3 \cdot 1$, and $13 \equiv_4 5$ because $13 - 5 = 8 = 4 \cdot 2$. If $x \equiv_k y$, we say that *x is congruent to y modulo k*, hence the terminology in the checkpoint problem. In this problem, you will prove several properties of modular congruence.

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

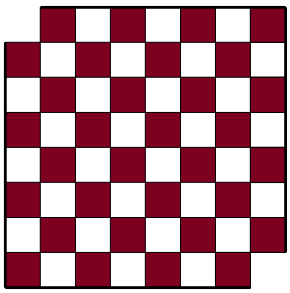
As you're working through the above proof, be careful not to assume what you need to prove. Don't start your proof off by assuming that there's a choice of q where $x - x = kq$ and then solving for q. After all, if you assume there's an integer q where $x - x = kq$, you're already assuming that $x \equiv_k x$! Instead, proceed along the lines of what we did in lecture when proving that if n is even, then n^2 is even: manipulate the expression $x - x$ and see if you can find a way to express it as something of the form kq.

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

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 $x \equiv_k z$.

The three properties you have just proven show that modular congruence is an *equivalence relation*. Equivalence relations show up everywhere in computer science, and we'll talk about them in week three.

Problem Nine: Tiling a Chessboard



Suppose you have a standard 8×8 chessboard with two opposite corners removed, as shown here. In the course notes (pages 60 - 61), there's a proof that it's impossible to tile this chessboard using 2×1 dominoes. This question considers what happens if you try to tile the chessboard using *right triominoes*, L-shaped tiles that look like this:

i. Prove it's impossible to tile an 8×8 chessboard missing two opposite corners with right triominoes.

ii. For $n \geq 3$, is it *ever* possible to tile an $n \times n$ chessboard missing two opposite corners with right triominoes? If so, find a number $n \geq 3$ such that it's possible and show how to tile an $n \times n$ chessboard missing two opposite corners with right triominoes. If not, prove that for every $n \geq 3$, it's impossible to tile an $n \times n$ chessboard missing two opposite corners with right triominoes.

Problem Ten: 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 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 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, …, 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 every statement in this list is a paradox. Some hints on this problem:
   - We've asked you to prove a universal statement (every element in this list is a paradox). What is the general template for proving a universal statement?
   - Split your proof into two parts. First, show that you get a contradiction if any of the statements in the list are true. Then, show that you get a contradiction if any of the statements in the list are false.
   - You should assume, as we've been doing in class, that every statement is either true or false. You don't need to worry about statements that are neither true nor false or statements that are simultaneously true *and* false.

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.

ii. 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. ☺

Optional Fun Problem: The Mouse and the Cheese (1 Point Extra Credit)[*]

On each problem set, we'll provide an optional fun problem for extra credit. 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.

Suppose that you have a 3" × 3" × 3" cube of cheese subdivided into twenty-seven 1" × 1" × 1" smaller cubes of cheese. A mouse wants to eat the entire cube of cheese and does so as follows: she first picks any small cube to eat first, then moves to an adjacent small cube of cheese (i.e. a cube that shares a face with the cube that was just eaten) to eat next, then repeats this process.

Prove that the mouse can't eat the centermost cube of cheese last.

[*] Adapted from Problem 4E of *A Course in Combinatorics, Second Edition* by Lint and Wilson.