Problem Set 2

This second problem set explores mathematical logic and dives deeper into formal mathematical proofs. We've chosen the questions here to help you get a more nuanced understanding for what first-order logic statements mean (and, importantly, what they don't mean) and to give you a chance to practice your proofwriting. By the time you've completed this problem set, we hope that you have a much better grasp of mathematical logic and how it can help improve your proofwriting structure.

Before attempting this problem set, we recommend that you do the following:

- Familiarize yourself with the online Truth Table Tool and play around with it a bit to get a feel for the propositional connectives.
- Read the online “Guide to Negations” and “Guide to First-Order Translations.”
- Read Handout #12, “First-Order Translation Checklist,” to get a better sense for common errors in first-order logic translations and how to avoid them.

Good luck, and have fun!

Checkpoint Questions Due Monday, October 7th at 2:30PM.
Remaining Questions Due Friday, October 11th at 2:30PM.

You are required to abide by the Stanford Honor Code. Information about how the Honor Code applies in CS103 can be found in the “CS103 and the Stanford Honor Code” handout available on the course website.
Checkpoint Problem One: Interpersonal Dynamics

The diagram to the right represents a set of people named A, B, C, D, E, and F. If there's an arrow from a person x to a person y, then person x loves person y. We’ll denote this by writing Loves(x, y). For example, in this picture, we have Loves(C, D) and Loves(E, E), but not Loves(D, A).

There are no “implied” arrows anywhere in this diagram. For example, even though A loves C and C loves E, the statement Loves(A, E) is false because there's no direct arrow from A to E. Similarly, even though C loves D, the statement Loves(D, C) is false because there's no arrow from D to C.

Below is a series of ten first-order logic statements. For each of those statements, tell us the minimum number of additional arrows that must be added to the diagram to the right to make that formula true. You're only allowed to add arrows; you can't remove existing ones. If a formula is already true, the answer will be “zero” because no edges need to be added. Then, tell us what those arrows are, and briefly explain why that’s the smallest number of arrows that need to be added.

i. Loves(A, A) → Loves(E, E)  vi. Loves(A, C) ↔ Loves(E, C)
ii. Loves(A, B) → Loves(C, D)  vii. ∀x. ∃y. Loves(x, y)
iii. Loves(A, C) → Loves(C, F)  viii. ∀x. ∃y. (x ≠ y ∧ Loves(x, y))
iv. Loves(A, D) → Loves(D, E)  ix. ∃x. ∀y. Loves(x, y)
v. Loves(A, D) ↔ Loves(B, C)  x. ∃x. ∀y. (x ≠ y → Loves(x, y))

Checkpoint Problem Two: First-Order Fundamentals

Consider the following first-order logic statement:

\[ \exists p. \ (\text{Problem}(p) \land \forall g. \ (\text{Program}(g) \rightarrow \lnot \text{Solves}(g, p)) \) \]

Here, the predicate Problem(x) states that x is a problem, the predicate Program(y) states that y is a program, and the predicate Solves(a, b) states that a solves b.

i. Translate this formula into plain English. No justification is required.

ii. Write a first-order logic formula that is the negation of the above formula. Your formula should not include any negations, except possibly for direct negations of predicates.

We strongly recommend reading over the Guide to Negations before attempting the above problem.

iii. Using only the predicates given above, write a statement in first-order logic that says “there is a program that solves every problem.” (Alas, this isn’t true, but wouldn’t it be nice?)

We strongly recommend reading over the Guide to Logic Translations before attempting the above problem.
Problem One: Propositional Completeness

In this problem, you'll explore some redundancies within the language of propositional logic.

This problem is autograded and should be submitted as part of the coding component for this problem set. Download the starter files for Problem Set Two from the course website, extract them somewhere convenient, and edit `PropositionalCompleteness.proplogic` with your answers. There's information inside each file with information about how to structure your answer. Briefly, if the online Truth Table Tool can understand your answer, so can our autograder. As usual, feel free to submit as many times as you'd like; we'll only grade your last submission.

In lecture, we covered the seven propositional connectives, which for convenience we've listed below:

\[ \land \quad \lor \quad \neg \quad \rightarrow \quad \leftrightarrow \quad \top \quad \bot \]

We settled on this set of connectives because they're convenient and expressive. However, it turns out that we could get away with fewer connectives than this.

i. Write expression equivalent to \( \bot \) that does not use any connectives besides \( \land \), \( \lor \), \( \neg \), and \( \top \). (You're welcome to use parentheses, but should not use any variables.)

ii. Write an expression equivalent to \( p \rightarrow q \) that does not use any connectives besides \( \land \), \( \lor \), \( \neg \), and \( \top \). (You're welcome to use the variables \( p \) and \( q \), along with parentheses.)

iii. Write an expression equivalent to \( p \leftrightarrow q \) that does not use any connectives besides \( \land \), \( \lor \), \( \neg \), and \( \top \). (You're welcome to use the variables \( p \) and \( q \), along with parentheses.)

Your answers to parts (i), (ii), and (iii) of this problem show that the the four propositional connectives \( \land \), \( \lor \), \( \neg \), and \( \top \) collectively are sufficient – the other three connectives can be rewritten purely in terms of them. However, there's some redundancy within those four connectives, and we can express all propositional formulas just using three of them.

iv. Write an expression equivalent to \( p \lor q \) that does not use any connectives besides \( \land \), \( \neg \), and \( \top \). (You're welcome to use the variables \( p \) and \( q \), along with parentheses.)

We can push this further. You can rewrite any propositional formula using just the \( \rightarrow \) and \( \bot \) connectives!

v. Write an expression equivalent to \( \top \) that does not use any connectives besides \( \rightarrow \) and \( \bot \). (You're welcome to use parentheses, but should not use any variables.)

vi. Write an expression equivalent to \( \neg p \) that does not use any connectives besides \( \rightarrow \) and \( \bot \). (You're welcome to use the variable \( p \), along with parentheses.)

vii. Write an expression equivalent to \( p \land q \) that does not use any connectives besides \( \rightarrow \) and \( \bot \). (You're welcome to use the variables \( p \) and \( q \), along with parentheses.)

As a hint, what happens if you negate an implication?

To recap: given the \( \rightarrow \) and \( \bot \) connectives, you can express \( \land \), \( \neg \), and \( \top \). From \( \land \), \( \neg \), and \( \top \) you can get \( \land \), \( \lor \), \( \neg \), and \( \top \). And from those four connectives, you can get back the original seven. Overall, any propositional formula can be expressed purely in terms of \( \rightarrow \) and \( \bot \). Nifty!
Problem Two: Set Theory and Propositional Logic

(Parts (i) – (v) of this question are autograded; please edit the file SetTheory.proplogic in the Problem Set Two starter files with your answers. Part (vi) should be submitted as part of your written answers.)

Imagine we have two sets \( A \) and \( B \) and some object \( x \). Let’s introduce two propositional variables:

- \( a \), which states that \( x \in A \), and
- \( b \), which states that \( x \in B \).

By combining \( a \) and \( b \) together in different ways using propositional logic, we can express different claims about how \( x \) relates to \( A \) and \( B \).

i. Write a statement in propositional logic using the variables \( a \) and \( b \) that says \( x \in A \cap B \).

ii. Write a statement in propositional logic using the variables \( a \) and \( b \) that says \( x \in A \cup B \).

iii. Write a statement in propositional logic using the variables \( a \) and \( b \) that says \( x \in A \setminus B \).

iv. Write a statement in propositional logic that says \( x \in A \Delta B \). To receive credit, your solution should use at most two connectives.

We care about the total number of connectives you use, not how many different types of connectives you use. For example, the formula \( p \land q \land r \land s \rightarrow t \) has four connectives.

Now, suppose we have some third set \( C \). Let’s introduce a third propositional variable \( c \) that means \( x \in C \).

v. Write a statement in propositional logic that says that \( x \in A \Delta B \Delta C \). To receive credit, your solution should use at most two connectives. (Note that \( A \Delta B \Delta C = (A \Delta B) \Delta C = A \Delta (B \Delta C) \).)

If you’ve solved part (iv), you should be able to get a solution that uses four connectives. By rewriting parts of the expression, you can then drop down to two.

vi. Using your answer to part (v) as a starting point, simplify the expression \( (A \Delta B) \Delta B \) as much as possible. Briefly justify your answer. No formal proof is necessary. (Please write up your solution to this problem in your written assignment submission, since we want to see your justification in addition to your answer.)

Generally speaking, if you ever find yourself in possession of a strange set-theoretic expression involving the set combination operations \( \cap, \cup, \setminus, \text{ or } \Delta \), you can use the above technique to get a slightly better handle at what you’re looking in fact. In fact, many rules about how these set-theoretic operators work follow directly from propositional logic!

That result in part (vi) is an interesting one. You may want to keep it in mind for later in the quarter. 😊
Problem Three: Executable Logic

This problem, and the ones after it, concern worlds populated by cats, robots, and humans. Love is in the air, and for whatever reason it seems appropriate to pin down the group dynamics using first-order logic. Let’s assume we have the predicates

- \textit{Person}(p), which states that \( p \) is a person;
- \textit{Cat}(c), which states that \( c \) is a cat;
- \textit{Robot}(r), which states that \( r \) is a robot; and
- \textit{Loves}(x, y), which states that \( x \) loves \( y \).

As a note, the \textit{Person}, \textit{Cat}, and \textit{Robot} predicates are mutually exclusive. For example, you can’t have a robot cat or a cat person. (I mean, you can have a “cat person” in the sense that you can have a person who likes cats, but not someone who is literally both a cat and a person. 😊) Additionally, you can assume that each entity in the world is either a person, a cat, or a robot. Finally, note that love is not necessarily symmetric. It’s possible for \( A \) to love \( B \) but not the other way around. (Alas!)

Below is a list of six first-order logic statements. Your task is to implement the six C++ functions defined in the file \textit{ExecutableLogic.cpp}, each of which determines whether one of the first-order logic formulas are true about a set of robots, cats, and people. Each robot, cat, or person is represented using a variable of type \textit{Entity}, and we’ve provided the following C++ functions to you, which mirror the four predicates given above:

```
bool Person(Entity p);
bool Cat   (Entity c);
bool Robot (Entity r);
bool Loves (Entity x, Entity y);
```

Our provided starter files will run the six functions you’ll implement on some sample worlds, which you can use to test out your implementations.

i. Consider the following first-order logic formula:

\[ \exists x. \text{Cat}(x) \]

Write C++ code for a function

```
bool isFormulaTrueFor_partI(std::set<Entity> S)
```

that takes in a set \( S \) and returns whether the above formula is true for the entities in that set.

ii. Repeat the above exercise with this first-order logic formula:

\[ \forall x. \text{Robot}(x) \]

iii. Repeat the above exercise with this first-order logic formula:

\[ \exists x. (\text{Person}(x) \land \text{Loves}(x, x)) \]

iv. Repeat the above exercise with this first-order logic formula:

\[ \forall x. (\text{Cat}(x) \rightarrow \text{Loves}(x, x)) \]

v. Repeat the above exercise with this first-order logic formula:

\[ \forall x. (\text{Cat}(x) \rightarrow \\
\quad \exists y. (\text{Person}(y) \land \neg \text{Loves}(x, y)) ) \]

\textit{It’s a lot easier to write code for this one if you use a helper function.}

vi. Repeat the above exercise with this first-order logic formula:

\[ \exists x. (\text{Robot}(x) \leftrightarrow \\
\quad \forall y. \text{Loves}(x, y)) \]
Problem Four: First-Order Negations

For each of the first-order logic formulas below, find a first-order logic formula that is the negation of the original statement. Your final formula must not have any negations in it except for direct negations of predicates. For example, given the formula

$$\forall c. (\text{Cat}(c) \rightarrow \exists p. (\text{Person}(p) \land \text{Loves}(p, c)),$$

you could give the formula

$$\exists c. (\text{Cat}(c) \land \forall p. (\text{Person}(p) \rightarrow \neg \text{Loves}(p, c)),$$

However, you couldn’t give as an answer the formula

$$\exists c. (\text{Cat}(c) \land \neg \exists p. (\text{Person}(p) \land \text{Loves}(p, c)),$$

since the inner negation could be pushed deeper into the expression.

To submit your answers, edit the file `FirstOrderNegations.fol` with your final formulas. That file contains information about how to format your answers.

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

i. $$\forall p. (\text{Person}(p) \rightarrow \exists c. (\text{Cat}(c) \land \text{Loves}(p, c) \land \forall r. (\text{Robot}(r) \rightarrow \neg \text{Loves}(c, r)),$$

ii. $$(\forall x. (\text{Person}(x) \leftrightarrow \exists r. (\text{Robot}(r) \land \text{Loves}(x, r)))) \rightarrow (\forall r. \forall c. (\text{Robot}(r) \land \text{Cat}(c) \rightarrow \text{Loves}(r, c)),$$

Be careful – make sure you understand how that formula is parenthesized before you try negating it.

iii. $$\forall c. (\text{Cat}(c) \rightarrow \exists r. (\text{Robot}(r) \land \forall x. (\text{Loves}(c, r) \leftrightarrow r = x)),$$

iv. $$\exists x. (\text{Cat}(x) \land (\forall r. (\text{Loves}(r, x) \rightarrow \text{Robot}(r)) \lor \forall p. (\text{Loves}(p, x) \rightarrow \text{Person}(p)))$$
Problem Five: This, But Not That

Below is a series of pairs of statements about groups of cats, robots, and people. For each pair, find the absolute simplest world in which the first statement is true and the second statement is false. (By “absolute simplest,” we mean using as few entities as possible, and having as few entities love each other as possible.)

To submit your answers, edit the file ThisButNotThat.worlds in the res/ directory. There’s information in that file about how to specify your worlds.

Make this statement true… … and this statement false.

i. \(\forall y. \exists x. Loves(x, y)\) \(\exists x. \forall y. Loves(x, y)\)

ii. \(\forall x. (\text{Person}(x) \lor \text{Cat}(x))\) \((\forall x. \text{Person}(x)) \lor (\forall x. \text{Cat}(x))\)

iii. \((\exists x. \text{Robot}(x)) \land (\exists x. Loves(x, x))\) \(\exists x. (\text{Robot}(x) \land Loves(x, x))\)

iv. \((\forall x. \text{Cat}(x)) \rightarrow (\forall y. Loves(y, y))\) \(\forall x. \forall y. (\text{Cat}(x) \rightarrow Loves(y, y))\)

v. \(\exists x. (\text{Robot}(x) \rightarrow \forall y. \text{Robot}(y))\) \((\forall x. \text{Robot}(x)) \lor (\forall x. \neg \text{Robot}(x))\)

As a hint, if you want to make a statement false, make its negation true.

That last one is subtle. Does the statement in the left column look fishy to you?

Problem Six: Translating into Logic

In each of the following, write a statement in first-order logic that expresses the indicated sentence. Your statement may use any first-order construct (equality, connectives, quantifiers, etc.), but you must only use the predicates Person, Robot, Cat, and Loves.

To submit your answers, edit the file TranslatingIntoLogic.fol with your formulas. There’s information in that file about the expected format for your answers.

i. Write a statement in first-order logic that says “robots do not love.” (How sad!)

As a reminder, love is considered directional. Even if robots do not love, it’s possible that people or cats might love robots. For example, I could love my Roomba even if it feels nothing toward me. 😃

ii. Write a statement in first-order logic that says “each robot loves every cat, but no cat loves any person.”

iii. Write a statement in first-order logic that says “each cat only loves itself.” (Okay, I’m not that cynical about cats. But it’s still a good exercise to translate this statement!)

iv. Write a statement in first-order logic that says “if you pick a person, you’ll find that they love a cat if and only if they also love a robot.”

v. Write a statement in first-order logic that says “each person loves exactly two cats and nothing else.” To clarify, each person is allowed to love a different pair of cats.

vi. Write a statement in first-order logic that says “no two robots love exactly the same set of cats.”

Looking for a good read on the theme of people, robots, pets, and love? Check out Ted Chiang’s novella The Lifecycle of Software Objects, which explores these ideas in depth. 😃
Problem Seven: Hereditary Sets

Let’s begin with a fun little definition:

A set $S$ is called a **hereditary set** if every $x \in S$ is also a hereditary set.

This definition might seem strange because it’s self-referential – it defines hereditary sets in terms of other hereditary sets! However, it turns out that this is a perfectly reasonable definition to work with, and in this problem you’ll explore some properties of hereditary sets.

i. Given the self-referential nature of the definition of hereditary sets, it’s not even clear that hereditary sets even exist at all! As a starting point, prove that there is at least one hereditary set.

When confronted with a new definition, it never hurts to try out some examples. So pick a set, any set, and then apply the above definition to see whether it’s hereditary. If so, great! You’re done. If not, make sure you can explain why not. Then, based on what you found, pick another set and repeat this process. After a few iterations, you will likely converge on an answer.

This strategy, by the way, is a **great** way to get a handle on any new definition.

ii. Prove that if $H$ is a hereditary set, then $\mathcal{P}(H)$ is also a hereditary set.

The hardest part of this problem is determining how to set this proof up in a way that’s precise and rigorous. Here are some things to think about:

- What is the general procedure for proving an implication? What’s the antecedent here? What’s the consequent?
- Consider the statement “every $x \in S$ is also a hereditary set.” Is that an existentially-quantified statement, a universally-quantified statement, both, or neither? Based on what you know about how to prove existentially-quantified and universally-quantified statements, what should you do to prove this statement?

After you’ve written up a draft of your proofs, take a minute to read over them and apply the criteria from the Proofwriting Checklist. Here are a few specific things to watch out for:

- Although we’ve just introduced first-order logic as a tool for formalizing definitions and reasoning about mathematical structures, the convention is to **not** use first-order logic notation (connectives, quantifiers, etc.) in written proofs. In a sense, you can think of first-order logic as the stage crew in the theater piece that is a proof – it works behind the scenes to make everything come together, but it’s not supposed to be in front of the audience. Make sure that you’re still writing in complete sentences, that you’re not using symbols like $\forall$ or $\rightarrow$ in place of words like “for any” or “therefore,” etc.
- A common mistake we see people make when they’re just getting started is to restate definitions in the abstract in the middle of a proof. For example, we commonly see people say something like “since $A \subseteq B$, we know that every element of $A$ is an element of $B$.” When you’re writing a proof, you can assume that whoever is reading your proof is familiar with the definitions of relevant terms, so statements like the one here that just restate a definition aren’t necessary. Instead of restating definitions, try to apply those definitions. A better sentence would be something to the effect of “Since $x \in A$ and $A \subseteq B$, we see that $x \in B$,” which uses the definition to conclude something about a specific variable rather than just restating the definition. See the Guide to Proofs on Set Theory for more details.

Hereditary sets are used in **foundational mathematics**, the study of how to assemble all mathematical structures from simple structures. If you’d like to learn more about them, take Math 161!
Problem Eight: 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:

(P): Statement (P) is false.

If statement (P) is true, then by its own admission statement (P) is false – a contradiction! On the other hand, suppose that statement (P) is false. Then we know that “statement (P) is false” is false, meaning that statement (P) 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 beautiful, subtle 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:

\[
\begin{align*}
(S_0): & \quad \text{All statements in this list after this one are false.} \\
(S_1): & \quad \text{All statements in this list after this one are false.} \\
(S_2): & \quad \text{All statements in this list after this one are false.} \\
\vdots
\end{align*}
\]

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

\[
(S_n): \quad \text{All statements in this list after this one are false.}
\]

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:

\[
\begin{align*}
(T_0): & \quad \text{All statements in this list after this one are false.} \\
(T_1): & \quad \text{All statements in this list after this one are false.} \\
(T_2): & \quad \text{All statements in this list after this one are false.} \\
\vdots \\
(T_{9,999,999,999}): & \quad \text{All statements in this list after this one are false.}
\end{align*}
\]

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. 😊
Problem Nine: Tournament Champions, Part Two

On Problem Set One, you explored *tournaments*, contests between \(n \geq 0\) players. You in particular learned about *tournament champions*, players who satisfy a particular set of criteria. This problem picks up where we left off. We recommend that you look back over Problem Set One for the definitions that you saw there and to review what results about tournaments you’ve already proved.

Let’s begin by introducing the notion of a *loop* in a tournament. A loop in a tournament \(T\) is a series of \(n \geq 3\) different players \(p_1, p_2, \ldots, p_n\) such that player \(p_1\) won against player \(p_2\), player \(p_2\) won against player \(p_3\), \ldots, and, finally, player \(p_n\) won against player \(p_1\). The *length* of a loop is the number of players in the loop.

i. Give an example of a loop of length five in the tournament shown above. No justification is necessary.

ii. Prove that if a tournament has any loops at all, then it must have a loop of length three. *This would be a great spot to draw pictures and try out examples. Try drawing tournaments with different numbers of players and loops of different sizes and see if you notice anything.*

As a hint, use a *proof by extreme case* like what you did in Problem Set One. Focus on the smallest loop in the tournament – what can you say about it?

Given any tournament \(T\), if you select a collection of players from \(T\), you can form a *subtournament* consisting of just those players and the games they’ve played against one another.

iii. Let \(T\) be an arbitrary tournament. Prove that if \(p\) is a player in \(T\) who lost at least one game, then at least one of the players who won against \(p\) is a tournament champion. *Before writing this proof, take a minute to articulate what it is that you need to do here. This statement has both universal and existential components. How do you prove a universally-quantified statement? How do you prove an existentially-quantified statement?*  

As a hint, split \(T\) into three groups: a subtournament of players who won against \(p\), a subtournament of players who lost to \(p\), and player \(p\) himself. *This is sometimes called the bowtie decomposition of a tournament. (Curious where the name comes from? Draw pictures!)*

iv. Prove that there are no tournaments with exactly two champions.

Optional Fun Problem: Insufficient Connectives (Extra Credit)

Every propositional logic formula could be written in terms of just \(\rightarrow\) and \(\bot\). However, you *cannot* express every possible propositional logic formula using just the \(\leftrightarrow\) and \(\bot\) connectives. Prove why not.