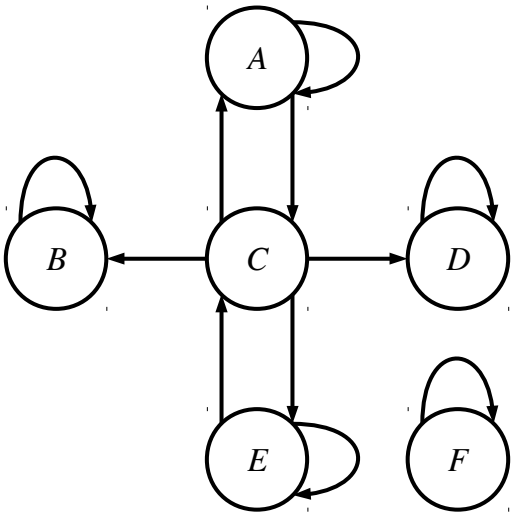
Problem Set 2

This second problem set explores mathematical logic. 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 apply first-order logic in the realm of proofs. 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.

Checkpoint Questions Due Tuesday, January 19 at 3:00PM.
Remaining Questions Due Friday, January 22 at the start of class.

Write your solutions to the following problems and submit them electronically on GradeScope by this Tuesday, January 19th at 3:00PM. As before, these problems will be graded on a 0/1/2 scale based on whether or not you have made a good, honest effort to complete all of them. We will try to get these problems returned to you with feedback on your proof style this Wednesday, January 20th.

Checkpoint Problem: Interpersonal Dynamics (2 Points if Submitted)



The diagram to the left 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 list of 16 formulas in first-order logic about the above picture. In those formulas, the letters *A* through *F* refer to individual people, and *P* represents the set of all the people. For each formula, determine whether that formula is true or false. No justification is necessary.

i. $Loves(A, C) \lor Loves(D, C)$

ii. $Loves(C, B) \lor Loves(C, D)$

iii. $Loves(B, E) \to Loves(D, A)$

iv. $Loves(A, E) \to Loves(A, A)$

v. $Loves(A, C) \to Loves(E, C)$

vi. $Loves(C, B) \to Loves(B, C)$

vii. $Loves(A, B) \leftrightarrow Loves(E, E)$

viii. $Loves(C, A) \leftrightarrow Loves(B, B)$

ix. $\forall x \in P.\ Loves(x, x)$

x. $\exists x \in P.\ Loves(x, x)$

xi. $\forall x \in P.\ \forall y \in P.\ (Loves(x, y) \leftrightarrow Loves(y, x))$

xii. $\exists x \in P.\ \exists y \in P.\ (Loves(x, y) \leftrightarrow Loves(y, x))$

xiii. $\forall x \in P.\ (Loves(x, x) \to \exists y \in P.\ Loves(x, y))$

xiv. $\forall x \in P.\ (Loves(x, x) \to \exists y \in P.\ (Loves(x, y) \land x \neq y))$

xv. $\exists x \in P.\ (Loves(x, x) \land \forall y \in P.\ Loves(x, y))$

xvi. $\exists x \in P.\ (Loves(x, x) \to \forall y \in P.\ Loves(x, y))$

The remainder of these problems should be completed and submitted online by Friday, January 22nd at the start of class.

Problem One: Implies and False (3 Points)

Although propositional logic has many different connectives, it turns out that any formula in propositional logic can be rewritten as an equivalent propositional formula that uses only the ¬, ∧, and ⊤ connectives. (You don't need to prove this). In this problem, you will prove a different result: every formula in propositional logic can be rewritten as an equivalent logical formula purely using the → and ⊥ connectives.

i. Find a formula that's logically equivalent to $\neg p$ that uses only the variable p and the → and ⊥ connectives. No justification is necessary.

ii. Find a formula that's logically equivalent to ⊤ that uses only the → and ⊥ connectives. No justification is necessary.

iii. Find a formula that's logically equivalent to $p \wedge q$ that uses only the variables p and q and the → and ⊥ connectives. No justification is necessary.

Since you can express ¬, ∧, and ⊤ using just → and ⊥, every possible formula in propositional logic can be expressed using purely the → and ⊥ connectives. Nifty!

Problem Two: Ternary Conditionals (4 Points)

Many programming languages support a *ternary conditional operator*. For example, in C, C++, and Java, the expression x `?` y `:` z means "evaluate the boolean expression x. If it's true, the entire expression evaluates to y. If it's false, the entire expression evaluates to z."

In the context of propositional logic, we can introduce a new ternary connective `?:` such that p `?` q `:` r means "if p is true, the connective evaluates to the truth value of q, and otherwise it evaluates to the truth value of r."

i. Based on this description, write a truth table for the `?:` connective.

ii. Find a propositional formula equivalent to p `?` q `:` r that does not use the `?:` connective. Justify your answer by writing a truth table for your new formula.

It turns out that it's possible to rewrite any formula in propositional logic using only `?:`, ⊤, and ⊥. The rest of this question will ask you to show this.

iii. Find a formula equivalent to $\neg p$ that does not use any connectives besides `?:`, ⊤, and ⊥. No justification is necessary.

iv. Find a formula equivalent to $p \rightarrow q$ that does not use any connectives besides `?:`, ⊤, and ⊥. No justification is necessary.

Since all remaining connectives can be written purely in terms of ¬ and →, any propositional formula using the seven standard connectives can be rewritten using only the three connectives `?:`, ⊤, and ⊥.

The fact that all propositional formulas can be written purely in terms of `?:`, ⊤, and ⊥ forms the basis for the *binary decision diagram*, a data structure for compactly encoding propositional formulas. Binary decision diagrams have applications in program optimization, graph algorithms, and computational complexity theory. Take CS166 or CS243 for more info!

Problem Three: First-Order Negations (5 points)

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, the negation of the formula $\forall x.\ (P(x) \to \exists y.\ (Q(x) \wedge R(y)))$ could be found by pushing the negation in from the outside inward as follows:

$$\neg(\forall x.\ (P(x) \to \exists y.\ (Q(x) \wedge R(y))))$$
$$\exists x.\ \neg(P(x) \to \exists y.\ (Q(x) \wedge R(y)))$$
$$\exists x.\ (P(x) \wedge \neg\exists y.\ (Q(x) \wedge R(y)))$$
$$\exists x.\ (P(x) \wedge \forall y.\ \neg(Q(x) \wedge R(y)))$$
$$\exists x.\ (P(x) \wedge \forall y.\ (Q(x) \to \neg R(y)))$$

Show every step of the process of pushing the negation into the formula (along the lines of what is done above). You don't need to formally prove that your negations are correct.

You may want to read over Handout 12 before attempting this problem.

i. $\exists p.\ (\mathit{Problem}(p) \wedge \forall g.\ (\mathit{Program}(g) \to \neg\mathit{Solves}(g, p)))$

ii. $\forall x \in \mathbb{R}.\ \forall y \in \mathbb{R}.\ (x < y \to \exists q \in \mathbb{Q}.\ (x < q \wedge q < y))$

iii. $(\forall x.\ \forall y.\ \forall z.\ (R(x, y) \wedge R(y, z) \to R(x, z))) \to (\forall x.\ \forall y.\ \forall z.\ (R(y, x) \wedge R(z, y) \to R(z, x)))$

iv. $\forall x.\ \exists S.\ (\mathit{Set}(S) \wedge \forall z.\ (z \in S \leftrightarrow z = x))$

v. $\forall k.\ (\mathit{SixClique}(k) \to \exists t.\ (\mathit{Triangle}(t, k) \wedge (\forall e.\ (\mathit{Edge}(e, t) \to \mathit{Red}(e)) \vee \forall e.\ (\mathit{Edge}(e, t) \to \mathit{Blue}(e)))))$

Problem Four: Love, Love Me Do (4 Points)

This question explores what happens when you interchange the order of quantifiers in a first-order logic statement.

Consider the following two statements in first-order logic:

$$\exists x \in P.\ \forall y \in P.\ \mathit{Loves}(x, y)$$

$$\forall y \in P.\ \exists x \in P.\ \mathit{Loves}(x, y)$$

Here, we'll imagine that P is a set of people, and $\mathit{Loves}(x, y)$ means that person x loves person y.

i. Translate these two statements into English.

ii. Prove that these statements are *not* equivalent to one another. To do so, come up with a group of people P such that one of these statements is true and the other is false, then justify why your choice works.

iii. One of these statements implies the other. Determine which statement implies the other, then prove it.

As a note on your answers for parts (ii) and (iii) of this problem: even though you're writing proofs about first-order logic, remember that your proof should still follow the same style guides as normal proofs: you should write in complete sentences, label formulas or substeps where appropriate, etc. In particular, in the course of writing up your answers, please only use notation from first-order logic when you're directly discussing the formulas, not as a substitute for plain English.

Problem Five: 'Cause I'm Happy (6 Points)

Let P be an arbitrary, nonempty set of people and let $Happy(x)$ mean that person x is happy.

Below are six statements in first-order logic. Each of these statements is one of two types:

- The statement is true only when either everyone in P is happy or no one in P is happy.
- The statement is always true, regardless of which people in P are happy.

For each statement, determine which of the two types that statement is and write a short proof explaining why. Your proofs should definitely demonstrate why each statement is of the type you choose; for the purposes of this problem, disproving that a statement is of one type is not sufficient to establish that it must be of the other type.

As a hint, when working on this problem, you may want to draw out some diagrams like the one in the checkpoint problem to help reason about which type of statement is which. We do *not* recommend trying to translate these statements back into English to reason about them – some of these statements have no reasonable representation in English.

i. $(\forall x \in P.\ Happy(x)) \rightarrow (\exists y \in P.\ Happy(y))$

ii. $(\exists x \in P.\ Happy(x)) \rightarrow (\forall y \in P.\ Happy(y))$

iii. $\forall x \in P.\ (Happy(x) \rightarrow (\exists y \in P.\ Happy(y)))$

iv. $\exists x \in P.\ (Happy(x) \rightarrow (\forall y \in P.\ Happy(y)))$

v. $\forall x \in P.\ \exists y \in P.\ (Happy(x) \rightarrow Happy(y))$

vi. $\exists y \in P.\ \forall x \in P.\ (Happy(x) \rightarrow Happy(y))$

Problem Six: Translating into Logic (10 points)

In each of the following, you will be given a list of first-order predicates and functions along with an English sentence. In each case, 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, functions, and constants provided. You do not need to provide the simplest formula possible, though we'd appreciate it if you made an effort to do so. ☺

i. Given the predicate

Natural(x), which states that x is an natural number

and the functions

$x + y$, which represents the sum of x and y, and
$x \cdot y$, which represents the product of x and y

write a statement in first-order logic that says "for any $n \in \mathbb{N}$, n is even if and only if n^2 is even."

ii. Given the predicates

Person(p), which states that p is a person;
Kitten(k), which states that k is a kitten; and
HasPet(o, p), which states that o has p as a pet,

write a statement in first-order logic that says "someone has exactly two pet kittens and no other pets."

iii. The ***axiom of pairing*** is the following statement: given any two distinct objects x and y, there's a set containing x and y and nothing else. Given the predicates

$x \in y$, which states that x is an element of y, and
Set(S), which states that S is a set,

write a statement in first-order logic that expresses the axiom of pairing.

iv. Given the predicates

$x \in y$, which states that x is an element of y, and
Set(S), which states that S is a set,

write a statement in first-order logic that says "every set has a power set."

v. Given the predicates

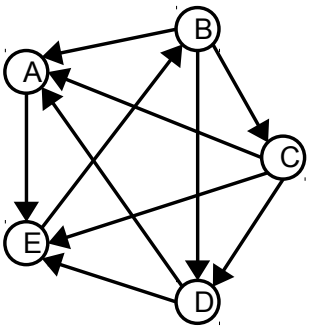
Lady(x), which states that x is a lady;
Glitters(x), which states that x glitters;
IsSureIsGold(x, y), which states that x is sure that y is gold;
Buying(x, y), which states that x buys y; and
StairwayToHeaven(x), which states that x is a Stairway to Heaven;

write a statement in first-order logic that says "there's a lady who's sure all that glitters is gold, and she's buying a Stairway to Heaven."*

---

* Let's face it – the lyrics to Led Zeppelin's "Stairway to Heaven" are impossible to decipher. Hopefully we can gain some insight by translating them into first-order logic!

Problem Seven: Tournament Winners (13 Points)



A ***tournament*** is a contest among n players. Each player plays a game against each other player, and either wins or loses the game (let's assume that there are no draws). We can visually represent a tournament by drawing a circle for each player and drawing arrows between pairs of players to indicate who won each game. For example, in the tournament to the left, player A beat player E, but lost to players B, C, and D.

A ***tournament winner*** is a player in a tournament who, for each other player, either won her game against that player, or won a game against a player who in turn won his game against that player (or both). For example, in the graph on the left, players *B*, *C*, and *E* are tournament winners. However, player *D* is ***not*** a tournament winner, because he neither beat player *C*, nor beat anyone who in turn beat player *C*. Although player *D* won against player *E*, who in turn won against player *B*, who then won against player *C*, under our definition player *D* is ***not*** a tournament winner. *(Make sure you understand why!)*

i. Suppose that there's a tournament among a group of players, including someone named Amy. Given the predicates

   - *Player*(*p*), which states that *p* is a player, and
   - *Beats*(*p*, *q*), which states that *p* beat *q* in the game they played,

   along with the constant symbol *Amy*, write a statement in first-order logic that says "Amy is a tournament winner."

ii. Negate the first-order formula that you wrote in part (i) and push the negations as deep as possible. Your resulting formula should only have negations directly applied to predicates. Write your answer in the style of Problem Three, showing each step.

iii. Translate your negated formula from part (ii) back into English. Make sure you understand why the statement you ended up with is indeed the negation of the original statement.

One of the most fundamental results about tournaments is the following: any tournament with at least one player has a tournament winner. In part (iv) of this problem, you'll prove that result.

iv. Let *T* be a tournament and *p* be a player in that tournament. Prove the following statement: if *p* won more games than anyone else in *T* or is tied for winning the most number of games, then *p* is a winner in *T*. *(Hint: There's a reason we had you do parts (i), (ii), and (iii) of this problem.)*

Although tournaments can have many winners, there's only one possible way that a tournament can have exactly one winner. The last two parts of this problem explore this case.

If *T* is a tournament, a ***subtournament*** of *T* is a tournament *T'* formed by picking a subset of the players in *T* and considering just the games those players played against each other. (Note that, just like any set is a subset of itself, any tournament is a subtournament of itself).

v. Let *T* be a tournament and *p* be a player in that tournament. Prove that if *p* lost any games, then at least one of the people who beat *p* must be a tournament winner in *T*. *(Hint: think about splitting T into two subtournaments. Also, there's a reason we asked part (iv) of this question.)*

vi. Let *T* be a tournament and let *p* be a player in that tournament. Prove that *p* is the *only* winner in the tournament if and only if *p* won all of her games. *(Hint: there's a reason why we asked part (v) of this problem.)*

Tournaments have many applications in computer science, especially in the intersection of social science, game theory, and algorithm design. One area of research, *computational social choice theory*, studies the computational difficulty of finding ways to obtain a desired result from an election given information about how different candidates would fare head-to-head. If you're interested in learning more, talk to Prof. Virginia Williams.

Extra Credit Problem: Unbounded Love (1 Point Extra Credit)

Given the predicates

- *Person*(*p*), which states that *p* is a person, and
- *Loves*(*x*, *y*), which states that *x* loves *y*,

write a statement in first-order logic with the following properties: the statement is always false if there are finitely many people, but there is some way for it to be true if there are infinitely many people. Then, prove that your formula has these properties.