Homework 2

Due: Wednesday, February 14, 2024

All homeworks are due at 11:59 PM on Gradescope.

Please do not include any identifying information about yourself in the handin, including your Banner ID.

Be sure to fully explain your reasoning and show all work for full credit.

Problem 1

Welcome to Jurassic Park! We are about to open the park, but we need to make sure that everyone that visits, no matter the language, can understand what is going on! Translate the sentences below into formulas of first-order logic. Note: the sets listed here can be used as *domains* of quantification. That is, you could write $\forall x : D, ...$ to quantify over all dinosaurs. You should not use any other domains of quantification.

- Sets:
  - D: The set of dinosaurs
- Predicates
  - $T(x)$: "x is a Tyrannosaurus Rex"
  - $H(x)$: "x is an herbivore"
  - $L(x, y)$: "x and y live in the same area"
  - $<$, $\leq$, $\geq$, and $>$ have their familiar meanings when applied to numbers.
  - $=$ is always a predicate in first order logic!
- Functions
  - $w(x)$: the amount of food x eats per day
  - $h(x)$: the height of x
- Constants
  - b: Bronchy, the most famous dino in the park
  - r: Rocky, the most dangerous dino in the park

a. An herbivore and a non-herbivore live in the same area.

b. There is a Tyrannosaurus Rex that is taller than any other dinosaur.

c. No herbivore eats more food than Rocky.

d. Rocky is taller than Bronchy, but Bronchy is at least as tall as every herbivore that lives in the same area as him.

e. Explain your answer in part b. Suppose that you had to argue this proposition was true. How would you justify the claim? What proof rules would you use?

Solution:

Students may come up with alternative translations. These are fine if they are logically equivalent.

a. $\exists x, \exists y : D, L(x, y) \land H(x) \land \neg H(y)$

b. $\exists x : D, T(x) \land \forall y : D, \neg y = x \to h(x) > h(y)$

c. $\forall x : D, H(x) \to w(x) \leq w(r)$ or equivalently $\neg\exists x : D, H(x) \land w(x) > w(r)$

d. $h(r) > h(b) \land \forall x : D, H(x) \land L(x, b) \to h(b) \geq h(x)$

e. Students should give an explanation of the first-order formula they came up with.

   An argument for this proposition would go as follows: I first use existential introduction and exhibit some dinosaur d, of which I must now show that the desired property holds. That property is a conjunction, so by and introduction, I separately prove each conjunct. First, I show that $T(d)$, i.e., that d is a T-rex. Next, I show the right-hand conjunct, which is a universally-quantified formula. To do this, universal-quantifier introduction says I let y be an arbitrary dinosaur and show that the right-hand claim holds. This claim is an implication, so implication introduction says I assume the antecedent (that y is distinct from my specific dinosaur d) and show that the conclusion follows. The conclusion is a numerical claim involving the $>$ predicate, so I use arithmetical reasoning to conclude that $h(d) > h(y)$.

Problem 2

In this question, we will consider first order logic with the following symbols:

- Sets:
  - $\mathbb{N}$: the set of natural numbers
  - List $\mathbb{N}$: the set of expressions in Python representing lists of natural numbers.
- Predicates
  - isSorted(x): "x is a sorted list"
  - isPrefixOf(s, l): "the list s is a prefix of the list l"
- Functions
  - reverse(l): the list whose elements are the elements of l in the opposite order.
  - sort(l): the list whose elements are the elements of l ordered from least to greatest.
  - drop(l): the list whose elements are the elements of l with the last one removed.
- Constants
  - All "list literals," specific lists of natural numbers that we can write down in full form: for example, $[1, 3, 2]$, $[55, 22]$, $[0]$, $[]$

a. For each of the following expressions, state whether it is a *term*, a *formula*, both, or neither. Explain your answer for each item.

1. $\forall x : \text{List } \mathbb{N}, \text{isSorted}(\text{sort}(x))$
2. $\text{sort}([1, 2, 3]) \land \text{drop}([1, 2, 3])$
3. $\exists l : \text{List } \mathbb{N}, \neg\text{isSorted}(l) \land \text{isSorted}(\text{drop}(l))$
4. $\text{sort}(\text{drop}(\text{reverse}([22, 11, 44])))$
5. $\forall l : \text{List } \mathbb{N}, \exists m : \text{List } \mathbb{N}, \text{isPrefixOf}(m, \text{isSorted}(l))$
6. $\exists x : \text{List } \mathbb{N}, \text{reverse}(\text{isSorted}(x)) = x$

b. Translate the following sentences of first order logic into English sentences. Try to make your translations as natural as you can—think about how you might say the property out loud if you were describing your own code.

1. $\forall l : \text{List } \mathbb{N}, \text{isSorted}(l) \to \text{isSorted}(\text{reverse}(l))$
2. $\exists l : \text{List } \mathbb{N}, \forall m : \text{List } \mathbb{N}, \text{isPrefixOf}(l, m)$
3. $\forall l : \text{List } \mathbb{N}, \neg\text{isSorted}(l) \to \exists p : \text{List } \mathbb{N}, (\text{isSorted}(p) \land \text{isPrefix}(p, l))$

Solution:

a.
1. This is a formula. It expresses a proposition about the term sort(x).
2. This is not well formed. You cannot combine terms with a logical connective like $\land$, which combines formulas.
3. This is again a formula, expressing a proposition about the existence of a list with a certain property.
4. This is a term. It represents a list. We know it's a term because it's a sequence of applications of functions, which produce terms.
5. This is not well formed. isPrefixOf is a predicate, but we have put a formula as its second argument, where it expects a term.
6. This is not well formed. We have given a formula as the argument to the function reverse, but should have given a term.

b.
1. The reverse of every sorted list is also sorted.
2. There is some list that is a prefix of every list. (This must be the empty list!)
3. Every unsorted list has a sorted prefix. ALTERNATIVELY: for every list l, if l is unsorted then there is a sorted list p that is a prefix of l.

Problem 3

This problem is a Lean question!

This homework question can be found by navigating to `BrownCs22/Homework/Homework02.lean` in the directory browser on the left of your screen in your Codespace. The comment at the top of that file provides more detailed instructions.

You will submit your solution to this problem separately from the rest of the assignment. Once you have solved the problem, download the file to your computer (right-click on the file in the Codespace directory browser and click "Download"), and upload it to Gradescope.

Problem 4 (Mind Bender — *Extra Credit*)

This mindbender is a real thinker, so get ready! A note: we're talking about *propositional* logic here, not first order logic. No quantifiers, no terms, just propositions and connectives.

We've seen a few ways to determine if a formula of propositional logic is valid. One way is to write out the full truth table, checking that every truth assignment makes the entire formula true. Another way is to write the negation of the formula in conjunctive normal form. If there are *any* clauses in the CNF expression, we can read off a falsifying assignment, and hence the formula is not valid.

For this question, we want to think about testing for *provability* instead of for *validity*. A formula φ is *provable* if there is a valid sequence of proof rules that will reduce a goal of proving φ to a proof state with no goals remaining. Our goal is to design an algorithm that will take in a propositional formula φ and, if the formula is provable, produce such a sequence of proof rules as its output.

While you don't need to think of this in terms of Lean, it can be helpful to visualize it: given a propositional formula, the algorithm could output a Lean proof of the formula.

Your task: think about how you would design an algorithm like this, and describe it! We're not expecting you to write any code. (Pseudocode is fine, if you'd like.) Explain what information your algorithm bases its "decisions" on, and give some examples of it "running" successfully.

Solving this task perfectly is hard, and we expect few or no fully correct answers to this question. Think about what kinds of propositions your algorithm might *fail* to prove, if any! Why is it unable to handle these?

Solution:

There are some key ideas that we were hoping to see in answers to this question.

- Approaches should be *goal-directed*. Apply intro rules to $\wedge$, $\leftrightarrow$, $\neg$, $\rightarrow$ goals greedily.
- Can also destruct $\wedge$, $\vee$, $\leftrightarrow$ hypotheses greedily: "hypothesis-directed" approach.
- Note that a combination of these will be necessary: to be successful you can't work only on the goals, or only on the hypotheses.
- Distinguish "safe" vs "unsafe" moves. For example: for an "or" goal, picking

a side can lose information, so this is unsafe. Should be done with some kind of strategy/backtracking plan.

- Modus ponens is also interesting. Using it in the "apply" (goal-directed) style is unsafe. It can be used greedily in the "have" (hypothesis-populating) style.
- Very subtle point, great if anyone picks up on it: proof by contradiction (in the classical sense: assuming $\neg p$ and showing a contradiction to prove p) complicates things! This rule is neither goal nor hypothesis directed.
- A rough strategy sketch could be this: do all of the safe unpacking/rearranging; then do unsafe case splits, backtracking as necessary; greedily apply modus ponens throughout to saturate context; if all else fails, try proof by contradiction and recurse.

  This isn't fully formed, but gets the rough idea across.

Brute-force approaches that didn't offer any significant theoretically-motivated insights received no credit.