In-Class Problems Week 3, Tue.

Problem 1.
In this problem we'll examine predicate logic formulas where the domain of discourse is $\mathbb{N}$. In addition to the logical symbols, the formulas may contain ternary predicate symbols A and M, where

$$A(k,m,n) \text{ means } k = m + n,$$
$$M(k,m,n) \text{ means } k = m \cdot n.$$

For example, a formula "Zero(n)" meaning that n is zero could be defined as

$$\text{Zero}(n) ::= A(n,n,n).$$

Having defined "Zero," it is now OK to use it in subsequent formulas. So a formula "Greater(m,n)" meaning $m > n$ could be defined as

$$\text{Greater}(m,n) ::= \exists k.\ \text{NOT}(\text{Zero}(k)) \text{ AND } A(m,n,k).$$

This makes it OK to use "Greater" in subsequent formulas.

Write predicate logic formulas using only the allowed predicates A, M that define the following predicates:

(a) Equal(m,n) meaning that $m = n$.

(b) One(n) meaning that $n = 1$.

(c) $n = i(m \cdot j + k^2)$

(d) Prime(p) meaning p is a prime number.

(e) Two(n) meaning that $n = 2$.

The results of part (e) will entend to formulas Three(n), Four(n), Five$(n), \ldots$ which are allowed from now on.

(f) Even(n) meaning n is even.

(g) (Goldbach Conjecture) Every even integer $n \geq 4$ can be expressed as the sum of two primes.

(h) (Twin Prime Conjecture) There are infinitely many primes that differ by two.

Problem 2.
If the names of procedures or their parameters are used in separate places, it doesn't really matter if the same variable name happens to be appear, and it's always safe to change a "local" name to something brand new. The same thing happens in predicate formulas.

For example, we can rename the variable x in "$\forall x. P(x)$" to be "y" to obtain $\forall y. P(y)$ and these two formulas are equivalent. So a formula like

$$(\forall x. P(x)) \text{ AND } (\forall x. Q(x)) \tag{1}$$

can be rewritten as the equivalent formula

$$(\forall y. P(y)) \text{ AND } (\forall x. Q(x)), \tag{2}$$

which more clearly shows that the separate occurrences of $\forall x$ in (1) are unrelated.

Renaming variables in this way allows every predicate formula to be converted into an equivalent formula in which every variable name is used in only one way. Specifically, a predicate formula satisfies the *unique variable convention* if

- for every variable x, there is at most one quantified occurrence of x, that is, at most one occurrence of either "$\forall x$" or "$\exists x$," and moreover, "$\forall x$" and "$\exists x$" don't both occur, and
- if there is a subformula of the form $\forall x. F$ or the form $\exists x. F$, then all the occurrences of x that appear anywhere in the whole formula are within the formula F.

So formula (1) violates the unique variable convention because "$\forall x$" occurs twice, but formula (2) is OK.

A further example is the formula

$$\begin{aligned}&[\forall x\, \exists y.\, P(x) \text{ AND } Q(x, y)] \text{ IMPLIES} \\ &\quad (\exists x.\, R(x, z)) \text{ OR } \exists x\, \forall z.\, S(x, y, w, z).\end{aligned} \tag{3}$$

Formula (3) violates the unique variable convention because there are three quantified occurrences of x in the formula, namely, the initial "$\forall x$" and then two occurrences of "$\exists x$" later. It violates the convention in others ways as well. For instance, there is an occurrence of y that is not inside the subformula $\exists y.\, P(x)$ AND $Q(y)$.

It turns out that *every* predicate formula can be changed into an equivalent formula that satisfies the unique variable convention—just by renaming some of the occurrences of its variables, as we did when we renamed the first two occurrences of x in (1) into y's to obtain the equivalent formula (2).

Rename occurrences of variables in (3) to obtain an equivalent formula that satisfies the unique variable convention. Try to rename as few occurrences as possible.

A general procedure for converting a predicate formula into unique variable format is described in Problem 7.32.

Problem 3.

Let $\tilde{\mathbf{0}}$ be a constant symbol, **next** and **prev** be function symbols taking one argument.

We can picture any model for these symbols by representing domain elements as points. The model must interpret $\tilde{\mathbf{0}}$ as some point e_0. It also interprets the symbols **next** and **prev** as total functions *nextf* and *prevf* on the domain. We can picture the functions by having an arrow labelled **next** go out from each point e into the point *nextf*(e), and an arrow labelled **prev** go out from each e into *prevf*(e). In particular,

Every point has exactly one ***next****-arrow, and exactly one* ***prev****-arrow, going out of it.*

The aim of this problem is to develop a series of predicate formulas using just the symbols $\tilde{\mathbf{0}}$, **next** and **prev** such that every model satisfying these formulas will contain contain a copy of the nonnegative integers $\mathbb{N}$, with *nextf* acting on the copy as the "$+1$" or *successor function* and *prevf* acting as the "-1" or *predecessor function*.

More precisely, the "copy" of $\mathbb{N}$ in a model will look like an infinite sequence of distinct points starting with e_0, with a **next**-arrow going from each point to the next in the sequence:

$$e_0 \xrightarrow{\textbf{next}} e_1 \xrightarrow{\textbf{next}} e_2 \xrightarrow{\textbf{next}} \ldots \xrightarrow{\textbf{next}} e_n \xrightarrow{\textbf{next}} \ldots$$

so that *nextf* acts like plus one. We also want *prevf* to act like minus one: whenever a **next**-arrow goes into an element e, then the **prev**-arrow out of e goes back to the beginning of the **next**-arrow:

$$\begin{tikzcd} e_0 \arrow[r, shift left, "\mathbf{next}"] & e_1 \arrow[l, shift left, "\mathbf{prev}"] \arrow[r, shift left, "\mathbf{next}"] & e_2 \arrow[l, shift left, "\mathbf{prev}"] \arrow[r, shift left, "\mathbf{next}"] & \dots \arrow[l, shift left, "\mathbf{prev}"] \arrow[r, shift left, "\mathbf{next}"] & e_n \arrow[l, shift left, "\mathbf{prev}"] \arrow[r, shift left, "\mathbf{next}"] & \dots . \arrow[l, shift left, "\mathbf{prev}"] \end{tikzcd} \tag{4}$$

Not shown is a further **prev**-arrow self-loop from e_0 to e_0, reflecting the convention for nonnegative integers that subtracting from zero has no effect.

There is a simple way to express this requirement as a predicate formula:

$$\forall x. \, \mathbf{prev}(\mathbf{next}(x)) = x \, . \tag{5}$$

Formula (5) means that the **prev**-arrow out of any point e goes back to the beginning of any **next**-arrow into e. Of course this will not be possible if there is more than one **next**-arrow into e.

There are some standard terms called *numerals* used to describe the elements $e_0, e_1, \dots$ in (4). Namely, the numeral for e_0 will be $\tilde{\mathbf{0}}$, and the numeral for e_n will be the application n times of **next** to $\tilde{\mathbf{0}}$. For example, the numeral for three would be:

$$\mathbf{next}(\mathbf{next}(\mathbf{next}(\tilde{\mathbf{0}}))). \tag{6}$$

We'll refer to the numeral for e_n as $\widetilde{\text{n}}$. So $\widetilde{3}$ refers to the term (6).

But we don't quite have the desired situation pictured in (4): there is nothing so far that prevents all the numerals having the same value. In other words, the formulas above are consistent with the formula $\mathbf{next}(\tilde{\mathbf{0}}) = \tilde{\mathbf{0}}$. (You should check this yourself right now.) We might try to fix this by requiring that no **next**-arrow can begin and end at the same element, but there still could be a pair of numeral values, each of which was *nextf* of the other. That is, the model might satisfy

$$\widetilde{2} ::= \mathbf{next}(\mathbf{next}(\tilde{\mathbf{0}})) = \tilde{\mathbf{0}}.$$

We could go on to forbid such length two cycles of **next**-arrows, but then there might be a cycle of three:

$$\widetilde{3} ::= \mathbf{next}(\mathbf{next}(\mathbf{next}(\tilde{\mathbf{0}}))) = \tilde{\mathbf{0}},$$

and so on. Fortunately, something we want to do anyway will fix this potential problem with the numeral values: we haven't yet provided a formula that will make the interpretion e_0 of $\tilde{\mathbf{0}}$ behave like zero.

(a) Write a predicate formula expressing the fact that the value of $\tilde{\mathbf{0}}$ is not plus-one of anything. Also, by convention, subtracting one from zero has no effect.

The formulas of part (a) and (5) together imply that the numeral values will yield the copy (4) of $\mathbb{N}$ we want. To verify this, we just need to show that the values of all the numerals are distinct.

(b) Explain why two different numerals must have different values in any model satisfying (5) and part (a).

Hint: It helps to describe the meaning of (5) by what it says about arrows. There is also an explanation based on the Well Ordering Principle.

So we have verified that any model satisfying (5) and the formula of part (a) has the desired copy of $\mathbb{N}$.

The distinction between formulas that are *true* over the domain $\mathbb{N}$ and formulas that are valid, that is true over *all domains* can be confusing on first sight. To highlight the distinction, it is worth seeing that the formula of part (a) together with (5) do *not* ensure that their models consist *solely* of a copy of $\mathbb{N}$.

For example, a model might consist of two separate copies of $\mathbb{N}$. We can stifle this particular possibility of extra copies of $\mathbb{N}$ pretty easily. Any such copy has to start with a zero-like element, namely one that is not in the range of *nextf*. So we just assert that there is the only zero-like element.

(c) Write a formula such that any model has only one copy of $\mathbb{N}$.

But the additional axiom of part (c) still leaves room for models that, besides having a copy of $\mathbb{N}$, also have "extra stuff" with weird properties.

(d) Describe a model that satisfies the formula of part (c) along with (5), part (a) and also satisfies

$$\exists x.\, x = \mathbf{next}(x). \tag{7}$$

Supplemental Part

(e) Prove that

$$\forall x \neq \tilde{\mathbf{0}}.\, \mathbf{next}(\mathbf{prev}(x)) = x. \tag{8}$$