CS103 Handout 11

Fall 2012 October 22, 2012

Section Handout 4

Problem One: Finding Flaws in Proofs

Two of the first three problem sets have had erroneous proofs on them. Now that you're equipped with first-order logic, look over the following incorrect proofs. For each proof, translate the statement of the theorem into first-order logic, then translate what theorem was actually proven into first-order logic. Explain why the statement of the theorem and the statement that was proven are not the same thing.

Theorem: For any sets A and B, $A \cup B \subseteq A \cap B$

Proof: We need to show that for any arbitrary $x \in A \cup B$, $x \in A \cap B$. If $x \in A \cap B$, then $x \in A$ and $x \in B$, so $x \in A \cup B$. ■

Theorem: If a binary relation R over the set A is not reflexive, then it is irreflexive.

Proof: Since R is not reflexive, there must be at least one $x \in A$ such that xRx does not hold. Since the choice of x was arbitrary, we must therefore have that for any $x \in A$, xRx does not hold. Thus R is irreflexive. ■

Theorem: Every positive integer can be written as $2x + 3y$ for some positive integers x and y.

Proof: By contradiction; assume that no positive integer can be written as $2x + 3y$ for some positive integers x and y. But this is clearly false; for example, $50 = 2 \cdot 10 + 3 \cdot 10$. We have reached a contradiction, so our assumption must have been wrong, so any positive integer can be written as $2x + 3y$ for some positive integers x and y. ■

Problem Two: Translating into Propositional Logic

For each of the following statements, translate that statement into propositional logic using the indicated propositional variables.

i. Let *r* be "you can run" and *h* be "you can hide." Write a statement in propositional logic that says "You can run, but you can't hide."

ii. Let *r* be "it rains" and *p* be "it pours." Write a statement in propositional logic that says "when it rains, it pours."

iii. Let *p* be "contains peanuts" and *m* be "contains milk." Write a statement in propositional logic that says "contains peanuts and/or milk."

iv. Let *w* be "I will go for a walk" and *v* be "the velociraptors are loose." Write a statement in propositional logic that says "I will go for a walk unless the velociraptors are loose."

Problem Three: Propositional Negations

For each of the propositional logic statements in problem two, find a statement in propositional logic that is the negation of that statement. Your statement must not have any negations that aren't directly applied to propositions. Then prove your statement is a valid negation by using a truth table.

Problem Four: First-Order Theories

In lecture, we've seen how to use quantifiers to encode statements like "for any natural number …" or "there is some color such that …" However, how might we use quantifiers to say something like the following?

Graph G is a DAG

One way to encode this might be to have a predicate *DAG(x)* that says if a graph is a DAG, but this is somewhat unsatisfying. A better idea would be to spell out, in first-order logic, a statement like

G is directed and G contains no cycles.

To encode that G is directed, we could introduce a simple predicate like *Digraph(g)*, which states whether *g* is a directed graph. But how might we encode this second fact, that G has no cycles? One way to say this would be to say that

For any path in G, that path does not start and end at the same node.

This looks a lot better, but we're going to run into some trouble when we try to write this out in first-order logic. Recall that a path is a series of edges $((v_0, v_1), (v_1, v_2), \dots, (v_{n-2}, v_{n-1}))$ of length n. In order for this to be a path, we would need to be able to guarantee that each of the edges in the sequence are in the graph and that the sink of each edge is the source of the next edge. But more fundamentally than that, we need to be able to quantify over $v_0, v_1, \ldots, v_{n-1}$ where we don't know the choice of n. This is difficult, because we can't just write out a statement like this one:

$$\forall v_0.\ \forall v_1.\ \forall v_2.\ \ldots\ \forall v_{n-1}.\ P(v_0, v_1, \ldots, v_{n-1})$$

Because we don't know how many nodes we need to quantify over.

In this problem, you'll see how to do such a quantification. The trick will be to notice that instead of quantifying over the nodes in the path, we can quantify over paths in general. As long as we sufficiently describe what a "path" is so that we're sure that we know what we're quantifying over, we can then state that a graph where each path is not a cycle must be a DAG. The task at hand, therefore, is to specify enough to ensure that when we say something to the effect of "p is a path in G," we know enough about p and G to confirm that the description matches our intuition.

i. Suppose that we define the following predicates:

   Digraph(g), which says if *g* is a directed graph,
   Path(p, g), which says that *p* is a path in *g*, and
   $x \in S$, which says that *x* is contained in set *S*.

   We also define the function

   Length(p), which returns the length of a path.

   Write a statement in first-order logic that says "the length of any path in a graph is a natural number." You can assume that $\mathbb{N}$ refers to the set of natural numbers.

ii. Now, let's introduce another predicate:

   $x < y$, which says if natural number *x* is less than natural number *y*

   Let's also add in the functions

   Nth(p, n), which returns edge *n* (zero-indexed) from path *p*, and
   EdgeSet(g), which returns the set of edges in graph *g*.

   Write a statement in first-order logic that says "any edge in a path in some graph is an edge in that graph." You may assume that the statement from part (i) is true, meaning that any path length must be a natural number.

iii. Let's introduce three more functions:

    Source(e), which returns the source node of an edge, and
    *Sink(e),*which returns the sink node of an edge.

    That is, given an edge (u, v), calling *Source* would return u and calling *Sink* would return v. Write a statement in first-order logic that says “the sink of each edge in a path is the source of the next node in the path.” You may assume that the sentences from (i) and (ii) are true, and can assume that standard arithmetic operations are legal. Make sure to remember not to say anything about the last node in the path!

iv. Finally, using the terminology from (i), (ii), and (iii), assuming G is a constant symbol referring to some particular digraph, and assuming that the statements you've written there are true, write a statement in first-order logic that says “G contains no cycles.” You may use any arithmetic operators that you'd like, and can assume that any natural number is a constant symbol.

What you have just done in parts (i), (ii), and (iii) is specify a *first-order theory*. From a programming perspective, if you were to write a program that implemented graphs and paths, no matter what implementation you use, as long as your implementation adheres to the statements listed above, you can guarantee that statement (iv) means that G contains no cycles.