CSE 311: Foundations of Computing I

Homework 3 (due Friday, October 22nd at 11:00 PM)

Directions: *Write up carefully argued solutions to the following problems. Your solution should be clear enough that it should explain to someone who does not already understand the answer why it works. However, you may use results from lecture, the theorems handout, and previous homeworks without proof.*

1. That's What I'm Talking About (0 points)

Approximately how much time (in minutes) did you spend on each problem of this homework? Were any problems especially difficult or especially interesting?

2. Asking For a Friend (18 points)

For each of the following English statements, (i) translate it into predicate logic, (ii) write the negation of that statement in predicate logic with the negation symbols pushed as far in as possible, and then (iii) translate the result of (ii) back to (natural) English.

Let the domain of discourse be all people on Facebook. You should use only the predicate Friend(x, y), which says that x and y are friends, and the predicates $x = y$ and $x \neq y$, which say whether or not x and y are the same persons. You may assume that friendship is a symmetric relationship, so Friend(x, y) and Friend(y, x) are the same. You can use constants to refer to specific people such as the "Bob" in the next example.

Example: Bob is friends with someone else.

i. $\exists x\,((x \neq \mathrm{Bob}) \land \mathsf{Friend}(x, \mathrm{Bob}))$ — note the *domain restriction* when negating

ii. $\forall x\,((x \neq \mathrm{Bob}) \rightarrow \neg\,\mathsf{Friend}(x, \mathrm{Bob}))$

iii. Everyone else is not friends with Bob.

(a) [9 Points] Two people have Bob as a mutual friend only if they are also friends.

(b) [9 Points] A group of three people are all friends with each other.

3. What Does That Prove? (10 points)

Theorem: Show that s follows from $(\neg q \lor r) \rightarrow \neg p$, $\neg p \rightarrow \neg r$ and $r \lor (q \land s)$.

(The rule "Proof by Cases", used here, is defined in Problem 5.)

"Proof":

1.	$(\neg q \lor r) \rightarrow \neg p$	Given
2.	$(\neg q \lor \neg\neg r) \rightarrow \neg p$	Double Negation: 1
3.	$\neg(q \land \neg r) \rightarrow \neg p$	DeMorgan's Law: 2
4.	$p \rightarrow (q \land \neg r)$	Contrapositive: 3
5.	$p \rightarrow \neg r$	$\land$ Elim: 4
6.	$\neg p \rightarrow \neg r$	Given
7.	$\neg r$	Proof by cases: 5, 6
8.	$r \lor (q \land s)$	Given
9.	$q \land s$	$\lor$ Elim: 8, 7
10.	s	$\land$ Elim: 9

(a) [6 Points] What is the most significant error in this proof? Give the line and briefly explain why it's wrong.

(b) [4 Points] Explain how to fix the proof by replacing the line with the most significant error with a correct inference of the same fact. (Your replacement may need multiple lines to prove that fact.)

4. Burden of Proof (26 points)

Write formal proofs of each of the following.

Feel free to use this web site again to complete the problem. Click on "CSE 311 HW3" to create a new project that is preloaded with a file for each of the parts below. Once you have solved each part, take a screenshot of your completed solution and include it in your submission. Some additional notes on usage:

- The up/down arrows indicate whether the reasoning is forward (down) or backward (up).
- The "Intro $\vee$" and "Direct Proof" rules can only be used *backward* at present.
- The web site only allows a single premise, so for parts with multiple premises, it will have a single premise that is the *conjugation* of all the stated ones. Get the individual premises using "Elim $\wedge$".

(a) [6 Points] Given $r \wedge s$, $r \to t$, and $t \to (u \wedge v)$, it follows that $s \wedge v$.

(b) [6 Points] Given $r \to s$ and $\neg r \to s$, it follows that $s \vee t$ is true.

(c) [6 Points] Given $p \to (r \wedge \neg s)$, $s \vee t$, and $(r \wedge t) \to u$, it follows that $p \to u$.

(d) [8 Points] Given $p \to ((s \to t) \wedge (t \to s))$, it follows that $((p \wedge s) \to (p \wedge t)) \wedge ((p \wedge t) \to (p \wedge s))$.

5. Proof Positive (12 points)

In this problem, we will consider the following, new inference rule:

Proof By Cases

$$\frac{A \vee B \quad A \to C \quad B \to C}{\therefore \quad C}$$

This rule says that, if we know that either A or B is true and that both A implies C and B implies C, then it follows that C is true. If it is A that is true, then we get that C is true by Modus Ponens and likewise if B is true instead. This is a valid rule of inference and you are **free to use it** outside of this problem as well.

(Note that you proved the special case $B = \neg A$ in part (b) of the previous problem! In that case, we do not need the assumption $A \vee B = A \vee \neg A$, as that is always true — it is the Law of the Excluded Middle.)

(a) [6 Points] Use the Proof By Cases rule to prove the following. Given $r \wedge (s \vee t)$, $s \to (t \wedge u)$, and $t \to (t \wedge u)$, it follows that $r \wedge (u \vee v)$. (You may not use Elim $\vee$.)

(b) [6 Points] Prove that the "Elim $\vee$" rule follows from "Proof By Cases". Specifically, use the Proof by Cases rule to prove that, given $p \vee r$ and $\neg p$, it follows that r is true. (You may not use Elim $\vee$.)

6. Something to Prove (18 points)

Let $P(x, y)$ and $Q(z)$ be predicates defined in some fixed domain of discourse.

(a) [6 Points] Prove that, given $\exists x\, \forall y\, P(x, y)$, it follows that $\forall y\, \exists x\, P(x, y)$.

(Note that we showed in class that the converse of this statement does not hold!)

(b) [6 Points] Let c be a fixed object. Prove $\neg Q(c)$ given $\forall x\, (Q(x) \to \forall y\, P(x, y))$ and $\neg P(c, c)$.

(c) [6 Points] Let c be a fixed object. Prove that, given $\forall y\, P(c, y)$ and $\forall x\, (Q(x) \to P(x, c))$, it follows that $\forall x\, (Q(x) \to \exists y\, (P(x, y) \wedge P(y, x)))$.

7. Back to Square One (16 points)

Recall that an integer n is a *square* iff there exists a $k \in \mathbb{Z}$ such that $n = k^2$. Formally, with our domain of discourse as the integers, we can define $\text{Square}(n) := \exists k\, (n = k^2)$.

(a) [2 Points] Write the following claim in Predicate Logic: if integers n and m are squares, then nm is a square. (Be careful!)

(b) [10 Points] Give a formal proof of the claim from part (a). In addition to the inference rules discussed in class, you can also rewrite an algebraic expression to equivalent ones using the rule "Algebra". (E.g., you could write "$a(b + 1) - a = ab$" with Algebra as the rule / explanation.)

(c) [4 Points] Translate your formal proof from part (b) into an English proof.

8. Extra Credit: A Step In the Right Direction (0 points)

In this problem, we will extend the machinery we used in HW1's extra credit problem in two ways. First, we will add some new instructions. Second, and more importantly, we will add *type information* to each instruction.

Rather than having a machine with single bit registers, we will imagine that each register can store more complex values such as

Primitives These include values of types int, float, boolean, char, and String.

Pairs of values The type of a pair is denoted by writing "$\times$" between the types of the two parts. For example, the pair $(1, \text{true})$ has type "int $\times$ boolean" since the first part is an int and the second part is a boolean.

Functions The type of a function is denoted by writing a "$\to$" between the input and output types. For example, a function that takes an int as argument and returns a String is written "int $\to$ String".

We add type information, describing what is stored in each each register, in an additional column next to the instructions. For example, if R_1 contains a value of type int and R_2 contains a value of type int $\to$ (String $\times$ int), i.e., a function that takes an int as input and returns a pair containing a String and an int, then we could write the instruction

$$R_3 := \texttt{CALL}(R_1, R_2) \qquad\qquad \text{String} \times \text{int}$$

which calls the function stored in R_2, passing in the value from R_1 as input, and stores the result in R_3, and write a type of "String $\times$ int" in the right column since that is the type that is now stored in R_3.

In addition to `CALL`, we add new instructions for working with pairs. If R_1 stores a pair of type String $\times$ int, then $\texttt{LEFT}(R_1)$ returns the String part and $\texttt{RIGHT}(R_1)$ returns the int part. If R_2 contains a char and R_3 contains a boolean, then $\texttt{PAIR}(R_2, R_3)$ returns a pair of containing a char and a boolean, i.e., a value of type char $\times$ boolean.

(a) Complete the following set of instructions so that they compute, in the final register assigned, a value of type float $\times$ boolean:

R_1	int $\times$ float
R_2	int $\to$ String
R_3	String $\to$ (char $\times$ boolean)
$R_4 := \ldots$	$\ldots$

The first three lines show the types **already stored** in registers R_1, R_2, and R_3 at the start, before your instructions are executed. You are free to use the values in those registers in later instructions.

Since we have unlimited space, store into a *new register* on each line. **Do not reassign** any registers.

(b) Compare the types listed next to these instructions to the propositions listed on the lines of your proof in Problem 4(a). Give a collection of text substitutions, such as replacing all instances of "r" by "int" (these can include both atomic propositions and operators), that will make the sequence of propositions in Problem 4(a) *exactly match* the sequence of types in Problem 8(a). (You may need to change your solution to Problem 8(a) slightly to make this work.)

(c) Now, let's add another way to form new types. If A and B are types, then $A + B$ will be the type representing values that can be of either type A or type B. For example, String + int would be a type of values that can be strings or integers.

To work with this new type, we need some new instructions. First, if R_1 has type A, then the instruction $\texttt{CASE}(R_1)$ returns the same value but now having type $A + B$. (Note that we can pick any type B that we want here.) Second, if R_2 stores a value of type $A + B$, R_3 stores a function of type $A \to C$ (a function taking an A as input and returning a value of type C), and R_4 stores a function of type $B \to C$, then the instruction $\texttt{SWITCH}(R_2, R_3, R_4)$ returns a value of type C: it looks at the value in R_2, and, if it is of type A, it calls the function in R_3 and returns the result, whereas, if it is of type B, it calls the function in R_4 and returns the result. In either case, the result is something of type C.

Complete the following set of instructions so that they compute, in the final register assigned, a value of type int $\times$ (char + boolean):

R_1	int $\times$ (float + String)
R_2	float $\to$ (String $\times$ char)
R_3	String $\to$ (String $\times$ char)
$R_4 := \ldots$	$\ldots$

The first three lines again show the types of values already stored in registers R_1, R_2, and R_3. As before, do not reassign any registers. Use a new register for each instruction's result.

(d) Compare the types listed next to these instructions to the propositions listed on the lines of your proof in Problem 5(a). Give a collection of text substitutions, such as replacing all instances of "r" by "int" (these can include both atomic propositions and operators), that will make the sequence of propositions in Problem 5(a) *exactly match* the sequence of types in Problem 8(c). (You may need to change your solution to Problem 8(c) slightly to make this work.)

(e) Now that we see how to match up the propositions in our earlier proofs with types in the code above, let's look at the other two columns. Describe how to translate each of the rules of inference used in the proofs from both Problem 4(a) and 5(a) so that they turn into the instructions in Problem 8(a) and 8(c).

(f) One of the important rules **not** used in Problems 4(a) or 5(a) was Direct Proof. What new concept would we need to introduce to our assembly language so that the similarities noted above apply could also to proofs that use Direct Proof?