CSE 311: Foundations of Computing I

Homework 3 (due Friday, October 18th 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. Do Not Iron While Wearing Shirt (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 people and activities. You should use only the predicates Loves(x, y) and Likes(x, y) which say that person x loves or likes (respectively) activity y; the predicates Person(x) and Activity(x), which say whether x is a person or activity, respectively; and the predicates $x = y$ and $x \neq y$, which say whether or not x and y are the same object. You can use constants to refer to specific people or activities such as the "Bob" in the next example.

Example: "There is some activity, other than ironing, that Bob likes.'

i. $\exists x\,((\text{Activity}(x) \wedge (x \neq \text{ironing})) \wedge \text{Likes}(\text{Bob}, x))$

ii. $\forall x\,((\text{Activity}(x) \wedge (x \neq \text{ironing})) \rightarrow \neg\,\text{Likes}(\text{Bob}, x))$
(Make sure you see why! It's important to notice domain restrictions when translating to English.)

iii. Bob does not like any activity other than ironing.
(Or less natural: Every activity other than ironing is not liked by Bob.)

(a) [9 Points] Every activity that Bob loves is also loved by someone else.

(b) [9 Points] Someone who likes every activity does not love any activity.

2. May Cause Drowsiness (26 points)

Write formal proofs of each of the following.

(a) [6 Points] Given $p \wedge q$, $p \rightarrow r$, and $r \rightarrow (s \wedge w)$, it follows that $q \wedge w$.

(b) [10 Points] Given $p \rightarrow q$, $r \rightarrow \neg p$, and $(\neg r \wedge q) \rightarrow s$, it follows that $p \rightarrow s$.

(c) [10 Points] Given $((p \wedge q) \rightarrow (p \wedge r)) \wedge ((p \wedge r) \rightarrow (p \wedge q))$, it follows that $p \rightarrow ((q \rightarrow r) \wedge (r \rightarrow q))$.

You can write your proofs by hand or in LaTeX. Alternatively, feel free to use either of the following tools:

- This tool will allow you to check the correctness of most proofs in Propositional Logic. You can use it for these problems by typing in the premise and conclusion and clicking "Start Proof". (You can take a screenshot of your completed proof and include it in your submission.)
- Cozy IDE now has a project template called "CSE 311 HW3" that is preloaded with these three problems. (As in HW2, you can download your completed project as a zip file and submit it in canvas.)

(Both tools only allow one premise, but they can take the "$\wedge$" of those facts as the premise instead.)

3. Contents Under Pressure (20 points)

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

Proof By Cases

$$\begin{array}{c} A \vee B \quad A \rightarrow C \quad B \rightarrow C \\ \hline \therefore \quad C \end{array}$$

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.

(a) [8 Points] Use the Proof By Cases rule to prove the following. Given $p \wedge (q \vee r)$, $q \rightarrow (r \wedge s)$, and $r \rightarrow (r \wedge s)$, it follows that $p \wedge (s \vee t)$

(b) [10 Points] Prove that the "Elim $\vee$" rule follows from "Proof By Cases". Specifically, use the Proof by Cases rule to prove that, given $p \vee q$ and $\neg p$, it follows that q is true. (You may not use Elim $\vee$.)

(c) [2 Points] What does (b) tell us about the relative power of "Elim $\vee$" versus "Proof By Cases" to infer new facts from given ones?

4. Not Intended For Human Consumption (20 points)

Let $Q(x)$ be a predicate defined in some, fixed domain of discourse.

(a) [8 Points] Prove that, given $\neg(\forall y\, Q(y))$, it follows that $\exists x\, (Q(x) \rightarrow \forall y\, Q(y))$.

(b) [8 Points] Prove that, given $\forall y\, Q(y)$, it follows that $\exists x\, (Q(x) \rightarrow \forall y\, Q(y))$.

Hint: It is okay to simply **repeat** a fact proved above, again, on another line below. Just cite the original line where it appeared as the explanation.

(c) [4 Points] Why can we conclude from parts (a) and (b) that $\exists x\, (Q(x) \rightarrow \forall y\, Q(y))$ is a tautology? Explain. (A formal proof is not necessary but is also allowed.)

5. Keep Away From Small Children (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.

6. Extra Credit: Some Assembly Required (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 2 (a). Give a collection of text substitutions, such as replacing all instances of "p" by "int" (these can include both atomic propositions and operators), that will make the sequence of propositions in problem 2 (a) *exactly match* the sequence of types in problem 6 (a). You may need to change your solution to problem 6 (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 $\rightarrow$ (String $\times$ char)
R_3	String $\rightarrow$ (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 3 (a). Give a collection of text substitutions, such as replacing all instances of "p" by "int" (these can include both atomic propositions and operators), that will make the sequence of propositions in problem 3 (a) *exactly match* the sequence of types in problem 6 (c). You may need to change your solution to problem 6 (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 2 (a) and 3(a) so that they turn into the instructions in problem 6 (a) and (c).

(f) One of the important rules **not** used in problems 2 (a) or 3 (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?