CSE 311: Foundations of Computing I Winter 2026

Problem Set 2

Due: Monday, Jan 26th by **11:00pm**

Instructions

Write up carefully argued solutions to the following problems. Each solution should be clear enough that it can explain (to someone who does not already understand the answer) why it works.

Collaboration policy. You are required to submit your own solutions. You are allowed to discuss the homework with other students. However, the **write up** must clearly be your own, and moreover, you must be able to explain your solution at any time. We reserve ourselves the right to ask you to explain your work at any time in the course of this class.

Solutions submission. Submit your solution via Gradescope. In particular:

- Each numbered task should be solved on its own page (or pages). Do not write your name on the individual pages. (Gradescope will handle that.)
- When you upload your pages, make sure each one is **properly rotated**. If not, you can use the Gradescope controls to turn them to the proper orientation.
- Follow the Gradescope prompt to **link tasks to pages**.
- You are not required to typeset your solution, but your submission must be **legible**. It is your responsibility to make sure solutions are readable — we will *not* grade unreadable write-ups.

Task 1 – Make the First Prove [10 pts]

For each of the following, write a **formal proof** that the claim holds.

Your proofs are only allowed to use the most basic rules: Modus Ponens, Intro $\wedge$, Elim $\wedge$, Intro $\vee$, and Elim $\vee$ unless stated otherwise.

a) Given $p \wedge (q \wedge r)$, s, and $(r \wedge s) \rightarrow (v \wedge u)$, it follows that $p \wedge u$ holds.

b) Given $p \wedge q$, $\neg q \vee r$, and $(\neg p \vee r) \rightarrow s$, it follows that $s \wedge p$ holds.

For this part, you are free to use **equivalences**. Use the name of the equivalence when applying it as a rule.

Task 2 – Pack Up the Proving Van [10 pts]

For each of the following, write a formal proof that the claim holds.

Your proofs are only allowed to use the Modus Ponens, **Direct Proof**, Intro $\wedge$, Elim $\wedge$, Intro $\vee$, and Elim $\vee$. Equivalences are not allowed. (*Hint*: Direct Proof will be needed!)

a) Given $(q \wedge r) \rightarrow (s \vee u)$, $\neg p \rightarrow (r \wedge \neg u)$, and $p \vee q$, it follows that $\neg p \rightarrow s$.

Your proof is only allowed to use the rules the Modus Ponens, **Direct Proof**, Intro $\wedge$, Elim $\wedge$, Intro $\vee$, and Elim $\vee$.

b) Given $(q \vee \neg p) \rightarrow r$, $(q \vee p) \rightarrow u$, and p, it follows that $(p \rightarrow q) \rightarrow (r \wedge u)$.

Task 3 – Provin' Right Along [10 pts]

For each of the following, write a formal proof that the claim holds.

In addition to the rules from Task 2, your proofs are also allowed to use Cases and the Latin rules. Equivalences are not allowed.

a) Given $p \vee q$, $p \rightarrow (r \vee s)$, $q \rightarrow (r \vee s)$, and $(r \vee s) \rightarrow (u \wedge v)$. Show that $p \vee (v \wedge u)$ holds.

Your proof is only allowed to use the rules Modus Ponens, Intro $\wedge$, Elim $\wedge$, Intro $\vee$, and **Cases**. (In particular, you do not need Direct Proof.)

b) Given $q \rightarrow (s \vee \neg p)$, $(p \wedge s) \vee (q \wedge \neg s)$, and $p \vee s$, it follows that p holds.

Hint: You might want to use **contradiction** at some point in your proof.[1]

[1] Recall that a "proof by contradiction" uses both Reductio Ad Absurdum and Principium Contradictionis.

Task 4 – Get a Prove On [10 pts]

For each of the following, write a formal proof that the claim holds.

In addition to the rules from Task 2, your proofs are also allowed to use Intro $\forall$, Elim $\forall$, Intro $\exists$, Elim $\exists$. Equivalences are not allowed.

Let $P(x)$, $Q(x, y)$, and $R(x, y)$ be predicates defined in some fixed domain of discourse, and let c be some well-known constant in that domain.

a) Given $\forall\, x, \forall\, y, (Q(x, y) \to Q(y, x))$ and $\forall\, x, (R(x, c) \wedge \forall\, y, (R(x, y) \to Q(x, y)))$, show that $\exists\, x, \forall\, y, Q(x, y)$.

b) Given $\forall\, x, (P(x) \wedge \exists\, y, Q(x, y))$, it follows that $\forall\, x, \exists\, y, (P(x) \wedge Q(x, y))$.

The fact that we can *move* the $\exists$ outside of the $\wedge$ was noted (but not proven) in lecture. In this problem, you will prove that you can sometimes move an $\exists$ outside of a $\wedge$.

Task 5 – Viva la Div-a [10 pts]

Let domain of discourse be the integers. Consider the following claim:

$$\forall a \forall b \forall c\, (((a \mid b) \wedge (a \mid (b + c)) \to (a \mid c))$$

In English, this claim says that *differences between divisible integers are divisible*: if a divides both two integers b and $b + c$ (for any a, b, c), a also divides the difference between them, c. As an example, if you know that 37 divides both 71706 and 88578, you know that 37 also divides $88578 - 71706$.

a) Write a **formal proof** that the claim holds.

b) Translate your formal proof to an **English proof**.

Keep in mind that your proof will be read by a *human*, not a computer, so you should explain the algebra steps in more detail, whereas some of the predicate logic steps (e.g., Elim $\exists$) can be skipped.

Task 6 – Extra Credit: Put That In Your Type and Smoke It

In this problem, we will extend the machinery we used in Homework 1'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, long, 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 "$\rightarrow$" between the input and output types. For example, a function that takes an int and returns a String is written "int $\rightarrow$ 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 $\rightarrow$ (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 a value of type float $\times$ char in the last register assigned (R_N for some N):

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

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.

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 Task 1a. Give a collection of text substitutions, such as replacing all instances of "P" by "float" (these can include substitutions for atomic propositions and for operators), that will make the sequence of propositions in Task 1a *exactly match* the sequence of types in part (a).

Note: You may need to change your solution to part (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 $\texttt{LCASE}(R_1)$ returns the same value but now having type $A + B$ and $\texttt{RCASE}(R_1)$ returns the same value but now having type $B + A$ (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 some value whose type is float + (long × char) in the last register assigned:

R_1	float + String
R_2	float → (boolean + int)
R_3	String → (boolean + int)
R_4	(boolean + int) → (char × long)
$R_5 := \ldots$	$\ldots$

The first four lines again show the types of values already stored in registers R_1 through R_4. 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 Task 3a. Give a collection of text substitutions, such as replacing all instances of "P" by "float" (these can include substitutions for atomic propositions and for operators), that will make the sequence of propositions in Task 3a *exactly match* the sequence of types in part (c). (You may need to change your solution to part (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 Task 1a and 3a so that they turn into the instructions in parts (a) and (c).

f) One of the important rules **not** used in Task 1a or 3a 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?