**Assignment 1: Logically Safe**
**15-316 Software Foundations of Security and Privacy**

Due: **11:59pm**, Thursday 2/1/18
Total Points: 50

1. **Proof practice (10 points).** Conduct a proof in the propositional sequent calculus that the following formula is valid. Be sure to say which proof rules apply at each step.

$$\neg(p \vee q) \leftrightarrow \neg p \wedge \neg q$$

2. **Propositional soundness (10 points).** Use the semantics of $\rightarrow$ to prove that the $\rightarrow$R rule is sound by showing that the validity of the premises imply the validity of the conclusion.

$$(\rightarrow\text{R}) \quad \frac{\Gamma, P \vdash Q, \Delta}{\Gamma \vdash P \rightarrow Q, \Delta}$$

3. **Safe superdiversity (20 points).** The so-called "monoculture problem" of software security refers to the fact that when a large set of users runs bytecode-identical versions of the same application, then any vulnerability affecting that application will have dramatic impact as it applies to the whole population. In response to this, engineers looked for ways to diversify software by distributing different versions of the same application, each with a unique bytecode representation.

   One example of this is a technique called "superdiversification", proposed by researchers from Nokia and Microsoft in 2008. The technique is applied by a compiler when generating an executable, and performs a brute-force search of all short instruction sequences to look for semantically-equivalent machine code implementations for desired functions.

   Being a prudent engineer, you are reluctant to adopt this crazy-sounding technique without convincing evidence that the implementations it produces are in fact equivalent to the original code you wrote. If this were not the case, your application might end up with arbitrary behaviors, potentially leading to even more vulnerabilities than would otherwise be present. Luckily, you are familiar with dynamic logic, and are able to rigorously prove that such implementations are correct.

   **Part 1.** Prove that the following code negates $y$ and stores it in $x$. In other words, give a proof in the sequent calculus using the axioms of dynamic logic to show that the following formula is valid.

$$y = a \rightarrow [x := x - y; y := y + x; x := x - y]x = -a$$

   **Part 2.** Prove that the following code adds 16 to $x$. In other words, give a proof in the sequent calculus using the axioms of dynamic logic to show that the following formula is valid.

$$(x = a \wedge y = 15) \rightarrow [\texttt{while}(y > 13)\,\{y := y - 1\}; x := x + y; x := x + 3]x = a + 16$$

4. **Conditional soundness (10 points).** Use the semantics of dynamic logic to prove that the [if] axiom is sound, i.e. all its instances are valid:

$$[\texttt{if}(Q)\,\alpha\,\texttt{else}\,\beta]P \leftrightarrow (Q \rightarrow [\alpha]P) \wedge (\neg Q \rightarrow [\beta]P)$$