Assignment 2: Dynamic Logic
15-316 Software Foundations of Security and Privacy

1. **Structured chaos (25 points).** Sometimes when modeling a computation, we need to avoid making assumptions about what exactly might transpire at runtime. For example, suppose that we wish to write a program that accepts alphanumeric user input for further processing. We don't know exactly what the user will type, but we do know that it will be a string of characters drawn from the set $\{a, \ldots, z, A, \ldots, Z, 0, \ldots, 9\}$. The way to model this situation formally is using *nondeterminism* with constraints. In this problem, we will explore how to do this in dynamic logic.

   Part 1 (10 points). Extend the language discussed in lecture by defining the semantics of a constrained nondeterministic assignment command, $x := Q(x)$. Informally, this command should nondeterministically assign a value to x that satisfies the formula $Q(x)$. Here, the notation $Q(x)$ means that Q is a formula with a free variable x. For example, after running $x := x > y$, the variable x could be assigned any integer greater than y in the current state. If $Q(x)$ is not satisfiable, e.g. if $Q(x)$ is equivalent to $x < 0 \wedge x > 0$, then the command should not enter any final state (i.e., should not terminate).

   Solution.

Part 2 (15 points). Design an axiom that allows you to reason about box modalities around nondeterministic branches:

$$[x := Q(x)]p(x) \leftrightarrow \ldots$$

The right side of this equivalence should not contain a box or diamond modality, but only first-order formulas. Prove that your axiom is valid using your semantics from Part 1.

Solution.

2. **Verified safety (15 points).** In the previous homework, you looked for ways to exploit a flawed runtime memory safety monitor. Recall that the safety policy aimed to ensure that a given program cannot write outside the range `0x800300`–`0x8003FF` (inclusive). Another way to enforce this policy is to verify that the program will not write outside the bounds, before executing it; this removes the need for any runtime monitors.

However, non-determinism arising from inputs that are unknown before execution can pose a challenge. Consider the program α below.

$$\alpha \equiv s := s \geqslant 0 \vee s < 0; p := p + s$$

Part 1 (5 points). Explain why the following formula is not valid by giving a trace of α that violates the safety policy.

$$\texttt{0x8000300} \leqslant p \leqslant \texttt{0x8000400} \rightarrow [\alpha]\texttt{0x8000300} \leqslant p \leqslant \texttt{0x8000400}$$

Your trace should be given as a sequence of states that show the values of s and p at each step. The easiest way to format this is as a table, e.g.:

	s	p
Initial state	$\ldots$	$\ldots$
$\vdots$	$\vdots$	$\vdots$

Solution.

Part 2 (10 points). Identify a formula $Q(s)$ to replace the nondeterministic assignment in α with that will make the program satisfy the safety policy. Is the formula from Part 1 now valid with your fix? If so, provide a sequent deduction using the axioms of dynamic logic. If not, then identify the premise in an attempted sequent deduction that is not valid.

Solution.