Answer 3 of the following 4 questions.

Note: In the questions that follow, I reference slides from the lectures, where I use numberings from the PS-files posted on my web page. Those numbers might not match exactly the ones in your notes, so please consult my web site if you are uncertain of a reference.

Question 1. Introduction to abstraction and static analysis

From Lecture 1, Slide 17 ("A Starting Point: Trace-Based Operational Semantics") shows the trace-based operational semantics for a while-loop program.

Here is a syntax for the language in which the program was written:

\[
P : \text{ProgramPoint} \\
C : \text{Command} \\
E : \text{Expression} \\
F : \text{BasicExpression} \\
x : \text{Variable} \\
N : \text{Numeral}
\]

\[
C ::= P : x := E \mid P \text{ while } E \text{ do } C \mid C_1 \text{ ; } C_2 \\
E ::= F_1 + F_2 \mid F_1 < F_2 \\
F ::= N \mid x \\
N ::= 0 \mid 1 \mid 2 \mid ... \\
P ::= p_0 \mid p_1 \mid p_2 \mid ... \\
x ::= a \mid b \mid c \mid ... 
\]

Write an algorithm that translates a program written in the above language into a set of state transition rules that compute the program’s "concrete" execution semantics on integers.

Next, write an algorithm that translates a program written in the above language into a set of state transition rules that compute the program’s abstract semantics on the abstract data values, \{even, odd\}. (See Slide 18: "We can abstractly interpret, say, for polarity").
Finally, write a proof that, for every program, the translated rules that compute the abstract values, \{even, odd\}, of variables are sound with respect to the rules that compute the concrete semantics. (Hint: Read Slides 20 and 21: “The underlying abstract interpretation semantics.”)

Question 2. Foundations of Abstract Interpretation

From Lecture 2, read Slides 14 and 15, “Closed binary relations.” Use the definition of $\gamma$ on Slide 14 to prove the Proposition stated on Slide 15.

Question 3. Mechanics of Static Analysis

Slide 16 of Lecture 3 defines \textit{forwards-necessarily reaching definitions analysis}. Rewrite the definitions of $\text{inReach}(p_i)$ and $\mathcal{f}^{\#}_i$ on that slide so that the analysis compute \textit{forwards-possibly reaching definitions analysis}.

Use your definition to rewrite the abstract transfer functions for the program on Slide 9 and recompute the flow-analysis table for the program, which should look similar to the one on Slide 11.

Question 4. Static Analysis: Applications and Logics

In Lecture 4, Slide 17 (“Constructing an abstract logic”) shows how to generate a distributive complete lattice. Prove Items 1 and 2 on Slide 18.

Next, prove that the distributive lattice generated from \{\text{neg}, \text{zero}, \text{pos}\}, where

\begin{align*}
 n \models \text{neg} & \text{ for all } n < 0 \\
 n \models \text{pos} & \text{ for all } n > 0 \\
 0 \models \text{zero} &
\end{align*}

quotiented by the induced function, $\gamma$ (that is, $A \equiv\gamma B$ if $\gamma(A) = \gamma(B)$), is exactly the \textbf{Signs} lattice, displayed at the bottom of Slide 18.