Set Theory Given X and Y are sets, with $A = X \to P(Y)$ and $B = \mathcal{P}(X \times Y)$.

We can think of A as a set of functions and B as a set of relations.

Consider $\varphi : A \to B$, such that $\varphi(f) = \{(x, y) \in X \times Y \mid y \in f(x)\}$.

We can think of $\varphi(f)$ as the set of all (x, y) such that y is one of the elements chosen by f for x.

Consider, $\psi : B \to A$, such that $(\psi(R))(x) = \{y \in Y \mid (x, y) \in R\}$.

For each x, $\psi(R)(x)$, is all the ys such that $(x, y) \in R$.

Let $f \in A$ be arbitrary. Then $\varphi(f) = \{(x, y) : y \in f(x)\}$.

Now, $\psi(\varphi(f)) = \{y \in Y : (x, y) \in \varphi(f)\} = \{y \in Y : y \in f(x)\} = f(x)$.

So, we see that $\psi(\varphi(f)) = f$.

Now, in the opposite direction, consider $R \in B$.

Then, $\psi(R)$ maps $x \mapsto \{y \in Y : (x, y) \in R\}$.

Again, after applying φ this time, we get, $\varphi(\psi(R) = \{(x, y) \in X \times Y : y \in \psi(R)(x)\}$.

By definition of $\psi(R)(x), y \in \psi(R)(x) \iff (x, y) \in R$.

Then, $\varphi(\psi)(R) = \{(x, y) \in X \times Y : (x, y) \in R\} = R$

Again, we observe $\varphi(\psi(R)) = R$.

Hence, φ and ψ are inverses of each other, and we have a 1-1 correspondence between A and B.

Figure 1: Property 1a

Figure 2: Property 1b

Figure 3: Property 2b

Model Checking

1. `CPAChecker` is a Configure Program Analysis tool that includes the analyses we discussed in class such as counterexample guided abstract refinement and lazy abstraction. Here, we are providing a specification and implementation, and using the `bfs` strategy with `CEGAR` to model check the provided files. The tool analyzes C programs against given specifications (.spc files) and identifies whether the specified properties hold. The three properties—Property1a, Property1b, and Property2b—focus on detecting specific conditions or violations within the testcase. When running Property1a, the

tool verifies if an error label occurs on a particular line of the program, indicating a property violation. For example, an output such as "Verification result: FALSE. Property violation (error label in line 1963) found by chosen configuration" suggests that the setup works correctly and the property was successfully checked. The `tcas.i` file represents a preprocessed C program for a Traffic Collision Avoidance System (TCAS). The program includes various conditions and calculations related to altitude, speed, and proximity to other aircraft with certain thresholds. In `Property1a`, we see that there's a check for the top and bottom thresholds with a goto error label when the condition is violated. The Property1a specification defines an observer automaton named Property1aAutomaton. Its purpose is to detect program locations labeled with "PROPERTY1a" and verify whether the program reaches an error state. This is a safety-critical property, likely associated with logical conditions, boundary violations, or invalid data states in the TCAS system. For example, if altitude resolutions or proximity calculations fail to meet their requirements, this property would be violated.

2. CPA Checker was extremely usable on my end. Docker desktop had a few issues, but the tool ran fine otherwise by following the instructions. I was able to pass in the inputs simply enough through the command line interface (CLI). The HTML output presented a graphical interface that included tabs that displayed figures of the Control Flow Automaton (CFA), the Abstract Reachability Graph (ARG), the source code, logs of the operations, resource usage stats, and the configurations used for the program analyses. Here, the predicate analysis uses the CEGAR platform discussed in class and uses `bfs` as the traversal strategy. Here, we can also see the counterexample trace leading to the erroneous state (when failed) and a clean graph otherwise.