2 Set Theory

We can exhibit an explicit bijection by defining two functions

$$\Phi : A \longrightarrow B \quad \text{and} \quad \Psi : B \longrightarrow A,$$

and show that Φ is an injection and a surjection.

Definition of Φ : $A \to B$

Given a function $f \in A$, i.e., $f : X \to \mathcal{P}(Y)$, define

$$\Phi(f) \;=\; \{\,(x, y) \;\mid\; x \in X,\; y \in f(x)\}\,.$$

For each $x \in X$, the image $f(x)$ is a subset of Y. Therefore, $\Phi(f)$ is the set of pairs (x, y) such that y *belongs* to the subset $f(x)$. Then, $\Phi(f) \subseteq X \times Y$, so $\Phi(f) \in B$.

Definition of $\Psi : B \to A$

Given a subset $R \in B$, i.e., $R \subseteq X \times Y$, define

$$\Psi(R) : X \;\to\; \mathcal{P}(Y), \quad x \;\mapsto\; \{\,y \in Y \;\mid\; (x, y) \in R\}\,.$$

For each $x \in X$, $\Psi(R)(x)$ is the set of all $y \in Y$ such that (x, y) lies in R. Hence $\Psi(R)(x)$ is a subset of Y. Thus, $\Psi(R)$ is indeed a function from X to $\mathcal{P}(Y)$, so $\Psi(R) \in A$.

To prove that Φ and Ψ form a bijection, we can show that

1. Φ *is injective (one-to-one).*
2. Φ *is surjective (onto).*

1. Injectivity of Φ

Suppose $f, g \in A$ (i.e., $f, g : X \to \mathcal{P}(Y)$) and assume $\Phi(f) = \Phi(g)$. We must show $f = g$.

By definition, $\Phi(f) = \Phi(g)$ means

$$\{(x, y) \mid y \in f(x)\} \;=\; \{(x, y) \mid y \in g(x)\}.$$

If f and g were different functions, then there exists some $x_0 \in X$ such that $f(x_0) \neq g(x_0)$. Then there is some y_0 in one of these sets but not the other. For example, if $y_0 \in f(x_0)$ but $y_0 \notin g(x_0)$, then $(x_0, y_0) \in \Phi(f)$ but $(x_0, y_0) \notin \Phi(g)$. This contradicts $\Phi(f) = \Phi(g)$.

Therefore, $f(x) = g(x)$ for all $x \in X$, which means $f = g$. Hence, Φ is injective.

2. Surjectivity of Φ

Let $R \subseteq X \times Y$ be any element of B. We seek $f \in A$ such that $\Phi(f) = R$.

Define $f \;=\; \Psi(R)$. Recall $\Psi(R)$ was given by

$$\Psi(R)(x) \;=\; \{\,y \in Y \;\mid\; (x, y) \in R\}.$$

Then,

$$\Phi(\Psi(R)) \;=\; \{(x, y) \mid y \in \Psi(R)(x)\} \;=\; \{(x, y) \mid (x, y) \in R\} \;=\; R.$$

Hence for every $R \in B$, we have found an $f \in A$ (namely $f = \Psi(R)$) such that $\Phi(f) = R$. Thus, Φ is surjective. Therefore, there is a *1-to-1 correspondence* (bijection) between the set of all functions $f : X \to \mathcal{P}(Y)$ and the set of all subsets of $X \times Y$.

This completes the proof of the bijection.

Question assigned to the following page: 3

3 Model Checking

When I run CPAchecker with the command *"cpa.sh -predicateAnalysis -spec Property1a.spc tcas.i"* it reads C program (tcas.i) along with the observer automaton in Property1a.spc. CPAchecker then symbolically explores every possible path in the program and checks whether any execution can reach the labeled "error" location that is associated with PROPERTY1A. Here, Property1a says, "It is forbidden for $Up\_Separation >= thresh$ and $Down\_Separation < thresh$ to happen at the same time." Since CPAchecker reports a property violation, it means there is an execution path under which $Up\_Separation >= thresh$ and $Down\_Separation < thresh$ are both true. As soon as that path is discovered, CPAchecker declares the property to be violated and outputs "FALSE.". What has been proved here is that the property in Property1a.spc does not always hold true for the tcas.i code. In the resulting HTML file, CPAchecker shows a counterexample trace leading to the error location, which confirms that the condition specified by Property1a is actually reachable. Figure 1 is the output for the first property.

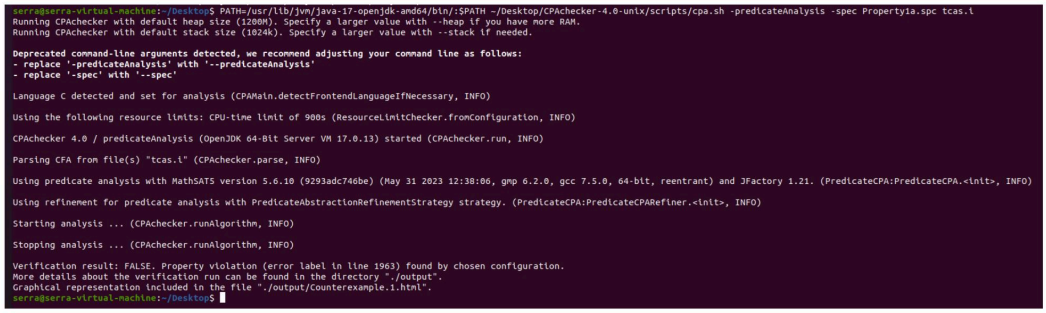

Figure 1: Output of cpa.sh -predicateAnalysis -spec Property1a.spc tcas.i

The other specification, Property1b is designed to detect whether there is any point in the execution where $Up\_Separation < thresh$ and $Down\_Separation >= thresh$ happen at the same time (leading the program to label that point as PROPERTY1B and call error()). Under "Property1b," CPAchecker's analysis concludes TRUE, meaning it finds no execution path where $Up\_Separation < thresh$ and $Down\_Separation >= thresh$ can occur simultaneously. As a result, the tool reports "No property violation found," signifying the code is safe with respect to that particular condition. The final HTML report for that analysis will also show no error paths, confirming that the verification ended without discovering a violation. Figure 2 is the output for the second property.

Figure 2: Output of cpa.sh -predicateAnalysis -spec Property1b.spc tcas.i

The last property is also going to be violated according to the results of the CPAChecker. In this case it finds a counterexample that satisfies the property given as, $Up\_Separation < thresh \wedge Down\_Separation < thresh \wedge Up\_Separation < Down\_Separation$. Figure 3 is the output of the CPAChecker given the last property. As for usability, I found it a little bit complicated but after passing the hardest part of the learning

Question assigned to the following page: 3

curve it was a lot more easy to understand. The HTML output that it provides at the end might be really helpful to debug the system since it outputs the path that lead to the counter example. One possible safety problems about this is that, a "TRUE" does not mean that your real world system is most certainly safe. Threats to validity include possible mismatches between the real environment and the model. If the software interacts with external data or hardware not fully captured in the verification model, the outcome may not translate to real-world scenarios. One more thing to think as a researcher is that configuration options in CPAchecker also need careful documentation to ensure reproducible and credible results.

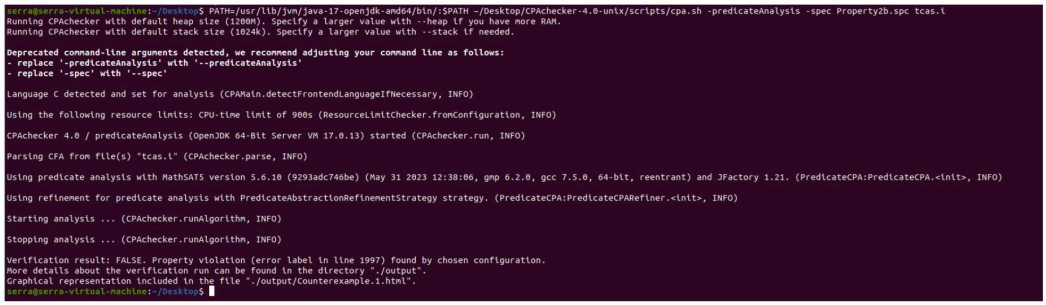

Figure 3: Output of cpa.sh -predicateAnalysis -spec Property2b.spc tcas.i