**Question 2:**

*Notes*:

I'll notate elements of $X \times Y$ as $(x, y)$, where $x \in X$ and $y \in Y$

$x \in S$, where $S \in B$ is slight notational abuse, but I mean if $x$ exists as the first element in any pairs of $S$.

We begin our proof by defining our function $f : B \to A$ as follows:

for each $x \in X$, if $x \in S$:

$x$ maps to all $y \in Y \mid (x, y) \in S$

if $x \notin S$

$x$ maps to $\emptyset$

We'll start by proving that this function $f$ is injective. Let $S_1, S_2 \in B$ s.t. $f(S_1) = f(S_2)$. We'll show that $S_1 = S_2$ by proving that $S_1 \subseteq S_2$ and $S_2 \subseteq S_1$.

Let $(x, y) \in S_1$

$\implies$ $f(S_1)$ includes a function that maps $x$ to a set including $y$

$\implies$ $f(S_2)$ includes a function that maps $x$ to a set including $y$ as we know $f(S_1) = f(S_2)$

$\implies$ $(x, y) \in S_2$ as this is the only way $x$ maps to a set including $y$ by our function definition

Thus, we have $S_1 \subseteq S_2$. By a symmetric proof, we get $S_2 \subseteq S_1$. Thus, we have $S_1 = S_2$ and $f$ must be injective.

Now, let us prove that $f$ must be surjective. Let $g$ be an arbitrary function from $A$. Then $g$ maps elements of $X$ to sets of elements of $Y$. Now, we construct the following set

$$S = \{(x, y), \forall x \in X, y \in Y \mid y \in g(x)\}$$

Clearly, $S \in B$ as $\{(x, y), \forall x \in X, y \in Y\} = X \times Y$. Then, $S \subseteq X \times Y$ and trivially, $S \in P(X \times Y)$ by the definition of power set. All that remains to be shown is that $f(S) = g$. When $f$ runs on $S$, it will map each $x$ to a set including every $y$ where $y \in g(x)$, faithfully recreating the sets that each $x$ should map to. Similarly, it accurately map each $x$ that does not appear to the empty set. Thus, we have shown that $f$ is surjective.

Since we have shown that $f$ is injective and surjective, then we have constructed a bijection between $B$ and $A$.

**Question 3:**

When the CPAChecker tool is run using the commands listed, it is using predicate analysis to verify whether the corresponding property specified is being violated. Property1a is defined in the code of the tcas.i file as a violation when `Up_Separation >= thresh && Down_Separation < thresh`.

CPAChecker claims to have proven that the program provided will never violate property 1b, but can in some cases violate property 1a and property 2b. For properties 1a and 2b, the CPAChecker provides counterexamples that violate these properties. In the html files for the counterexamples, CPAChecker provides control flow graphs that display the path the program will take to arrive at the property violation. Furthermore, CPAChecker generates an assignment file that shows the variable assignments that lead to the error path. In the case where CPAChecker gives us counterexamples, it'd be very easy to verify whether these counterexamples actually allow us to violate the property given that we have deterministic code—we can just run the code on the counterexample provided. In the case where CPAChecker tells us that the property is not violated, we can trust that the property will hold insofar as we can trust CPAChecker and the tcas.i file written. In a brief read-through of the tcas code, the actual functionality seems to be fairly straightforward, making the contents of the file easy to trust. CPAChecker has additionally been validated by other verification tools in many cases and appears fairly trustworthy as a piece of software.

All in all, CPAChecker seems to be a pretty usable tool as it provides so much information. The CFG and Reachability graphs help to detail counterexample cases. Furthermore, the statistics it provides gives clear insight to the end-user on how CPAChecker is operating. The combination of information it provides makes it a trustworthy tool.

*Terminal Outputs:*

Property1a

```
Running CPAchecker with default heap size (1200M). Specify a larger value with -heap if you have more RAM.
Running CPAchecker with default stack size (1024k). Specify a larger value with -stack if needed.
Language C detected and set for analysis (CPAMain.detectFrontendLanguageIfNecessary, INFO)

Using the following resource limits: CPU-time limit of 900s (ResourceLimitChecker.fromConfiguration, INFO)

CPAchecker 2.0 / predicateAnalysis (OpenJDK 64-Bit Server VM 11.0.5) started (CPAchecker.run, INFO)

Parsing CFA from file(s) "tcas.i" (CPAchecker.parse, INFO)

Using predicate analysis with MathSAT5 version 5.6.5 (63ef7602814c) (Nov  9 2020 09:01:58, gmp 6.1.2, gcc 7.5.0, 64-bit,
 reentrant) and JFactory 1.21. (PredicateCPA:PredicateCPA.<init>, INFO)

Using refinement for predicate analysis with PredicateAbstractionRefinementStrategy strategy. (PredicateCPA:PredicateCPA
Refiner.<init>, INFO)

Starting analysis ... (CPAchecker.runAlgorithm, INFO)

Stopping analysis ... (CPAchecker.runAlgorithm, INFO)

Verification result: FALSE. Property violation (error label in line 1963) found by chosen configuration.
More details about the verification run can be found in the directory "./output".
Graphical representation included in the file "./output/Counterexample.1.html".
```

Property1b

```
Running CPAchecker with default heap size (1200M). Specify a larger value with -heap if you have more RAM.
Running CPAchecker with default stack size (1024k). Specify a larger value with -stack if needed.
Language C detected and set for analysis (CPAMain.detectFrontendLanguageIfNecessary, INFO)

Using the following resource limits: CPU-time limit of 900s (ResourceLimitChecker.fromConfiguration, INFO)

CPAchecker 2.0 / predicateAnalysis (OpenJDK 64-Bit Server VM 11.0.5) started (CPAchecker.run, INFO)

Parsing CFA from file(s) "tcas.i" (CPAchecker.parse, INFO)

Using predicate analysis with MathSAT5 version 5.6.5 (63ef7602814c) (Nov  9 2020 09:01:58, gmp 6.1.2, gcc 7.5.0, 64-bit,
 reentrant) and JFactory 1.21. (PredicateCPA:PredicateCPA.<init>, INFO)

Using refinement for predicate analysis with PredicateAbstractionRefinementStrategy strategy. (PredicateCPA:PredicateCPA
Refiner.<init>, INFO)

Starting analysis ... (CPAchecker.runAlgorithm, INFO)

Stopping analysis ... (CPAchecker.runAlgorithm, INFO)

Verification result: TRUE. No property violation found by chosen configuration.
More details about the verification run can be found in the directory "./output".
Graphical representation included in the file "./output/Report.html".
```

Property2b

```
Running CPAchecker with default heap size (1200M). Specify a larger value with -heap if you have more RAM.
Running CPAchecker with default stack size (1024k). Specify a larger value with -stack if needed.
Language C detected and set for analysis (CPAMain.detectFrontendLanguageIfNecessary, INFO)

Using the following resource limits: CPU-time limit of 900s (ResourceLimitChecker.fromConfiguration, INFO)

CPAchecker 2.0 / predicateAnalysis (OpenJDK 64-Bit Server VM 11.0.5) started (CPAchecker.run, INFO)

Parsing CFA from file(s) "tcas.i" (CPAchecker.parse, INFO)

Using predicate analysis with MathSAT5 version 5.6.5 (63ef7602814c) (Nov  9 2020 09:01:58, gmp 6.1.2, gcc 7.5.0, 64-bit,
 reentrant) and JFactory 1.21. (PredicateCPA:PredicateCPA.<init>, INFO)

Using refinement for predicate analysis with PredicateAbstractionRefinementStrategy strategy. (PredicateCPA:PredicateCPA
Refiner.<init>, INFO)

Starting analysis ... (CPAchecker.runAlgorithm, INFO)

Stopping analysis ... (CPAchecker.runAlgorithm, INFO)

Verification result: FALSE. Property violation (error label in line 1997) found by chosen configuration.
More details about the verification run can be found in the directory "./output".
Graphical representation included in the file "./output/Counterexample.1.html".
```

1 HW0

- **0 pts** Correct