Exercise 0F-2. Set Theory [5 points]. To Show: There is a 1-1 correspondence, or bijection between A and B, where $A = X \to \mathcal{P}(Y)$ and $B = \mathcal{P}(X \times Y)$
Proceed by proof by example
Define function $f : B \to A = \{(b, \{(x, \{y|(\_, y) \in b\})|x \in X\})|b \in B\}$
Function f maps elements in B to elements in A by mapping each set in B, b to a function, and that function maps all of the elements of x to a set which contains the elements of y for which that value of x appears in a pair with in b.

To Show: f is an injection
Proceed by proof by contradiction

Define two sets, b_1, b_2 such that $b_1, b_2 \in B$, $b_1 \neq b_2$, and $f(b_1) = f(b_2)$
Since $b_1, b_2 \in B$, then by definition of B and of powersets, $b_1 \subseteq (X \times Y)$ and $b_2 \subseteq (X \times Y)$

Since $b_1 \neq b_2$, then there exists a pair (x', y') such that either $(x', y') \in b_1$ and $(x', y') \notin b_2$ or $(x', y') \notin b_1$ and $(x', y') \in b_2$. The names b_1 and b_2 are arbitrary, define (x', y') such that $(x', y') \in b_1$ and $(x', y') \notin b_2$

Define f_{b_1} as $f(b_1) = f_{b_1}$ such that $y' \in f_{b_1}(x')$ by definition of f
Define f_{b_2} as $f(b_2) = f_{b_2}$. Since $(x', y') \notin b_2$, then $y' \notin f_{b_2}(x')$.

Because $y' \notin f_{b_2}(x')$ and $y' \in f_{b_1}(x')$, then $f_{b_1}(x') \neq f_{b_2}(x')$, and $f_{b_1} \neq f_{b_2}$. This means $f(b_1) \neq f(b_2)$. This contradicts the statment "Define two sets, b_1, b_2 such that $b_1, b_2 \in B$, $b_1 \neq b_2$, and $f(b_1) = f(b_2)$". Because this statement has been contradicted, this means if $b_1, b_2 \in B$ and $b_1 = b_2$, $f(b_1) = f(b_2)$. This is the definition of an injection. f has been proved to be an injection.

To Show: f is a surjection
Proceed by proof by contradiction

Define a such that $a \in A$ and $a \notin f(B)$. Since B is defined as $\mathcal{P}(X \times Y)$, then there exists a set in B that includes every subset of $\{(x, y)|x \in X \& y \in Y\}$

A consists of the set of all functions that map values of X to $\mathcal{P}(Y)$

The function f maps all elements in B to a value in A such that the value in A is a function that maps all of the values in X to a subset of Y that includes all values that appear in a pair with that given value in X in the given value in B.

Since B includes all possible subsets of $(X \times Y)$, it includes all possible combinations of values of y appearing or not appearing in a pair with all possible values of X. Thus, all possible functions that map values in X to a value in $\mathcal{P}(Y)$ must appear as an output

from f, by the definition of f. Because of the previous statement, a cannot exist. Thus, $\forall a.(a \in A \& a \in f(B)$ is true. This is the definition of surjection, so f is a surjection.

f has been proven to be both an injection and surjection above. A bijection is a function that is both an injection and surjection. By the definition of bijection, f is a bijection. Because f has proven to be a bijection, there does exist a 1-1 correspondence between A and B.

Exercise 0F-3. Model Checking [10 points]. Property1a.spc output:

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

CPAchecker 2.0 / predicateAnalysis (OpenJDK 64-Bit Server VM 11.0.9.1) 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:PredicateCPARefiner.<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.spc output:

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

CPAchecker 2.0 / predicateAnalysis (OpenJDK 64-Bit Server VM 11.0.9.1) 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:PredicateCPARefiner.<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.spc output:

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

CPAchecker 2.0 / predicateAnalysis (OpenJDK 64-Bit Server VM 11.0.9.1) 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:PredicateCPARefiner.<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".
```

Response:
When I run the CPA checker tool, it is checking if the input program has the ability to violate the given property. It checks if this property can be violated on any possible path. More specifically, the given properties are checking if the Up_Separation and Down_Separation variables are set properly according to the given threshold. `property1a` checks if any possible path includes the error path in the function `property1a`. The error path in `property1a` is only reachable if the Up_Separation is greater than or equal to the threshold and the Down_Separation is below the threshold. Since `property1a` results in

a violation, there is a possible path where Up_Separation is greater than or equal to the threshold and Down_Separation is below the threshold.

Besides `property1a`, I was also given `property1b` and `property2b`. Looking at all three of these properties, I have found that `tcas.i` is a reasonable test suite. These properties are aiming to confirm that the Down_Separation and Up_Separation variables have been set correctly. Using this test suite to verify that there is no possible path that leads to these improper values for Up_Separation and Down_Separation is reasonable. Even though `property1b` errors when Up_Separation is greater than Down_Separation and `property2b` errors when Up_Separation is less than Down_Separation, `tcas.i` is still reasonable because `property2a` is in a different conditional block than `property2b`, so any execution path that includes `property2a` could not include `proeprty2b`. This means the two properties do not violate each other, and the test suite remains reasonable. As a whole, tcas.i is a reasonable test suite for CPA Checker. The properties illustrate how CPA Checker can guarantee that a certain value is constrained in all paths according to these properties. CPA checker can also uses this test suite to illustrate how it shows a counter example when the property on this variable is violated. `tcas.i` is able to demonstrate the ability of CPA checker to validate a property and demonstrate a counterexample when a property has been violated.

CPA Checker has proved that there are possible paths in the code in which Up_Separation and Down_Separation violate the expected values relative to a threshold and each other with respect to `property1a` and `property2b`, but there are no possible paths that violate `property1b` and that property can be considered safe. Specific paths have also been found that violates `property1a` and `property2b` respectively. This information is helpful in debugging this code, as it lays out a specific case that results in the violated property, and a developer can use this trace to find the bug. CPA Checker can then be used iteratively until no possible paths exist that violate `property1a` and `property2b`. CPA Checker has proved to be a very helpful tool in debugging `tcas.i` since the properties confirm the values of member values before they are used elsewhere for functionality relevant to the traffic collision avoidance. On the whole, CPA Checker is a useful tool, as it can be used to ensure mission critical variables conform to the expected values. It can be used to ensure that a developer fixes all possible edge cases for a given property.

1 HW0

- **0 pts** Correct