Exercise 0F-2. Set Theory [5 points]. Let $f : B \to A$ such that for any $b \in \mathcal{P}(X \times Y)$, $f(b)$ returns the function $g : X \to \mathcal{P}(Y)$, where g is a piecewise function defined as follows:

$$g(x) = \begin{cases} \{y_j : (x, y_j) \in b\} & \text{if } x \text{ is in one of } b\text{'s coordinate pairs} \\ \emptyset & \text{if } x \in X, \text{ but } x \text{ is not one of } b\text{'s coordinate pairs} \end{cases}$$

Claim: f is a bijection. We prove this by showing that it is both injective and surjective.

Injectivity: Suppose $f(b_1) = f(b_2)$ for $b_1, b_2 \in \mathcal{P}(X \times Y)$. Then $f(b_1) = g_1 : X \to \mathcal{P}(Y)$ such that $g_1(x_i \in b_1) = \{y_j : (x_i, y_j) \in b_1\}$ and $g_1(x_i \notin b_1 \wedge x_i \in X) = \emptyset$. Similarly, $f(b_2) = g_2 : X \to \mathcal{P}(Y)$ such that $g_2(x_i \in b_2) = \{y_j : (x_i, y_j) \in b_2\}$ and $g_2(x_i \notin b_2 \wedge x_i \in X) = \emptyset$. Since all $g(x_i) = \emptyset$ mappings are the same between g_1 and g_2, b_1 and b_2 must have the same x_i coordinates. Since the $g(x_i) = \{y_j : (x_i, y_j) \in b_z\}$ mappings are the same between g_1 and g_2, b_1 and b_2 must also have the same y_i coordinates. Hence, $b_1 = b_2$ as they must be the same element of the $\mathcal{P}(X \times Y)$. Therefore, f is injective.

Surjectivity: Let $a \in X \to \mathcal{P}(Y)$. a is a function mapping every $x \in X$ to an element of $\mathcal{P}(Y)$. Suppose we want to have a particular function a such that $a(x_1) = z_1, a(x_2) = z_2, \ldots, a(x_n) = z_n$, where $z_j \in \mathcal{P}(Y)$ are particular elements of the power set of Y; i.e., particular subsets of Y. We can construct a by choosing $b \in \mathcal{P}(X \times Y)$ such that for each $a(x_k) = z_k$, we include $(x_k, z_1), (x_k, z_2), \ldots, (x_k, z_n)$ in b. (If $a(x_k) = z_k = \emptyset$, we don't add any (x_k, z_i) elements to b.) We see by construction that $f(b) = a$ because the set b contains all the coordinate pairs necessary to make f output our desired a by the piecewise method described above. Hence, f is surjective.

Exercise 0F-3. Model Checking [10 points].

Property 1a Run Output:

```
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, r
eentrant) and JFactory 1.21. (PredicateCPA:PredicateCPA.<init>, INFO)

Using refinement for predicate analysis with PredicateAbstractionRefinementStrategy strategy. (PredicateCPA:PredicateCPARe
finer.<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".
```

Property 1b Run Output:

```
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, r
eentrant) and JFactory 1.21. (PredicateCPA:PredicateCPA.<init>, INFO)

Using refinement for predicate analysis with PredicateAbstractionRefinementStrategy strategy. (PredicateCPA:PredicateCPARe
finer.<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".
```

Property 2b Run Output:

```
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, r
eentrant) and JFactory 1.21. (PredicateCPA:PredicateCPA.<init>, INFO)

Using refinement for predicate analysis with PredicateAbstractionRefinementStrategy strategy. (PredicateCPA:PredicateCPARe
finer.<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".
```

I found CPAChecker to be a very usable and intuitive program without any configuration issues on WSL, and I think that the `tcas.i` program is a representative, though incomplete, test suite illustrating CPAChecker's usefulness. Although my knowledge of Traffic Collision Avoidance Systems and airplanes is virtually non-existance, I've deduced that the (a) properties flag violations where the TCAS mistakenly directs a plane to fly upwards, and the (b) properties flag violations where the TCAS mistakenly directs a plane to fly downwards. In particular, `Property1a` is violated if the system decides a downward Resolution Advisory (RA) is necessary and `( Up_Separation` $\geq$ `thresh &&` `Down_Separation` $<$ `thresh )`; i.e., we've ordered the plane to fly down but the downward separation to the next plane is below some safety threshold, whereas the upward separation exceeds the allowed threshold. `Property1b` measures the same thing as `Property1b`, but with the downward and upward threshholds switched, while `Property2a` (and `Property2b`, respectively) are violated when a downward (or upward, respectively) RA is ordered and both directions are below the safety threshhold, but the downward (or upward, respectively) direction has even less space between planes.

Providing counterexamples showing that `Property1a` and `Property2b` can be violated demonstrate both CPAChecker's competence as a software verification program and `tcas.i`'s usefulness as a test suite – detecting a potentially fatal error in a collision avoidance system and giving the developers an opportunity to correct this pre-deployment is an incredible benefit. Seeing CPAChecker validate `Property1b` establishes further confidence in CPAChecker and the `tcas.i` test suite, as the checker correctly verifies that this property holds.

However, we should be wary of preemptively calling CPAChecker a *correct* verifier or of con-

sidering `tcas.i` a *comprehensive* test suite. It's possible other tests could cause CPAChecker to run without bound/until terminated, and the `tcas.i` test suite doesn't verify all of the potentially important properties one could potentially verify in a collission avoidance system (e.g. horizontal distance threshholds, distance thresholds when descending on the landing strip, etc.). CPAChecker can provide strong evidence of errors in (and later correctness of) the `tcas.i` system, and it's good that this test suite provides good coverage of verficiation of "correct" software and of exposing bugs in incorrect software. However, many and varied test suites will be necessary to provide enough compelling evidence to support a claim that CPAChecker verifies software correctly.

1 HW0

- **0 pts** Correct