Exercise 0F-2. Set Theory [5 points]. -

T.S. "Show there is a 1-to-1 correspondence between the sets A and B, where $A = X \to \mathcal{P}(Y)$ and $B = \mathcal{P}(X \times Y)$."

The proof proceeds by example. We will define two functions: $f : A \to B$ and $g : B \to A$. and prove that they are inverses of each other. Specifically, that $a = g(f(a))\ \forall a \in A$ and $b = f(g(b)) \forall b \in B$. By showing this, we prove that there is a 1-to-1 correspondence between A and B.

$f(a) : A \to B = \{\ (x_i, y_j) \mid \forall (x_i, Y_i') \in a,\ \forall y_j \in Y_i',\ Y' \neq \{\}\}$

T.S f is a function from A to B.

By inspection, we can see that f takes in an element from A and produces an element in B.

$(g(b))(x) : B \to A = \{\ (x, \{y_j | \forall (x, y_j) \in b\})\}$

T.S. g is a function from B to A.

By inspection, we can see that g takes in an element from B and produces and element in A.

Now we need to show that these two functions are inverses of each other. We'll start by showing $a = g(f(a))\ \forall a \in A$.

T.S. $a = g(f(a))\ \forall a \in A$

The proof proceeds directly. We will produce $a' = g(f(a))$ for an arbitrary a and show that $a = a'$.

$b' = f(a) = \{\ (x_i, y_j) \mid \forall (x_i, Y_i') \in a,\ \forall y_j \in Y_i',\ Y' \neq \{\}\}$

In b' every pair, $(x, y) \in b'$, containing x_i has a y that originated from Y_i' in a. Any x_i that was mapped to the null set in a does not get included in the set of pairs b'.

$a' = (g(b'))(x) = \{\ (x, \{y_j | \forall (x, y_j) \in b'\})\}$

In a', any x that's get passed in to the function only gets mapped to the set of y's that it was paired with in b'. However, when we constructed b', we paired each x only with the y's it was mapped to in a. For those x elements that weren't mapped to any y element in b', they get mapped to the null set in a'. However, in order for a x to not appear in b' it must have originally been mapped to the null set in a. With these facts in mind, we can see that $a'(x) = a(x)$, which shows that $a = g(f(a))$.

Peer Review ID: 62013676 — enter this when you fill out your peer evaluation via gradescope

T.S $b = f(g(b)) \forall b \in B$.

The proof proceeds directly. Once again, we will produce $b' = f(g(b))$ for an arbitrary b and show that $b = b'$.

$$a' = (g(b))(x) = \{ \ (x, \{y_j | \forall (x, y_j) \in b\})\}$$

$$b' = g(a') = \{ \ (x_i, y_j) \mid \forall (x_i, Y_i') \in a', \ \forall y_j \in Y_i', \ Y' \neq \{\}\}$$

One thing we can see here is that, for the $Y' \neq \{\}$,
$Y' = \{y_j | \forall (x_i, y_j) \in b\}$, since that's how the (x, Y') in a' pairs were formed. If we make this substitution:

$$b' = g(a') = \{ \ (x_i, y_j) \mid \forall (x_i, Y_i') \in a', \ \forall y_j \in \{y_j \mid \forall (x_i, y_j) \in b\}\}$$

From here we can see that the function is reconstructing the original (x, y) pairs from b. Which means that b' and b are the same, and therefore that $b = f(g(b)) \forall b \in B$.

Now that we have shown that $a = g(f(a)) \ \forall a \in A$ and $b = f(g(b)) \forall b \in B$, we can conclude that the functions f and g are inverses of each other, and that there is a 1-to-1 correspondence between sets A and B.

Peer Review ID: 62013676 — enter this when you fill out your peer evaluation via gradescope

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

Property1a:

```
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:

```
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:

```
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".
```

In `tcas.i` there are a collection of properties added into the code: `Property1a`, `Property1b`, `Property 2b`, etc. Each of these properties checks a certain condition for certain integer variables used in the program, such as Up_Separation, Down_Separation, Own_Track_Alt. If the condition isn't met then the code would throw an error. When we run the command for a specific property, we are checking to see if it is at all possible for the error state for that property to ever be reached. For example, `Property1a` is checking to see if Up_Separation is greater than or equal to a threshold value and if Down_Separation is less than that threshold. If so, an error is reached. When we run the CPAChecker on `Property1a`, we are checking to see if it's ever possible for the `Properti1a` error to ever be reached.

Even though `tcas.i` contains many different properties that can be used to test CPAChecker's performance, I don't believe that this file on its own is a comprehensive test suite. The reason for this is because the properties that exist in the file only

check very simple boolean expressions for integer values. All that has been proven with `tcas.i` is that CPAChecker can validate these straightforward conditions. But what about something that is a little more complicated? For example, in lecture we covered the example of a program that calls lock() and unlock() on some resource, and we wanted to make sure the program never reaches a state where it tries to call lock()/unlock() when it shouldn't. This property is much more complex than simple integer comparisons, so if we really wanted to ensure that CPACheckers was a comprehensive tool for model checking then more test files should be included in the testing suite that contain these more complicated properties.

In regards to how CPAChecker performs as a tool for model checking, I think it's incredibly powerful. One of the key aspects of model checking is not just being able to detect validity but to also showcase a potential error scenario. This is one of the key features of CPAChecker that really stood out to me. The tool produces a very clean graphical representation of the CFA and it shows the exact path it determined to be a counterexample to the validity of the property (highlighted in red). If that's still to vague for the user, the tool also highlights the lines of code it followed to produce the counterexample path so they can see exactly how the program could reach the error state. This allows to user to verify the counterexample if they still have any doubts in CPAChecker's accuracy.

1 HW0

- **0 pts** Correct