0F-2. Define $f : \mathcal{P}(X \times Y) \to (X \to \mathcal{P}(Y))$ by mapping $H \subseteq X \times Y$ to the function $f_H : X \to \mathcal{P}(Y)$ given by

$$f_H(x) = \{y \in Y \mid (x, y) \in H\}.$$

We claim f is a bijection, so let us first show that f is injective. Assume $f(G) = f(H)$ for some $G, H \subseteq X \times Y$. Then $f_G = f_H$, so for any $x \in X$ we have that $f_G(x) = f_H(x)$. In particular, $(x, y) \in G$ if and only if $y \in f_G(x)$, so if and only if $y \in f_H(x)$ by the mentioned equality, thus if and only if $(x, y) \in H$. This gives that $G = H$, and it follows that f is injective.

Now let us show that f is surjective. Assume g is a function $X \to \mathcal{P}(Y)$. Define $G \subseteq X \times Y$ by

$$G = \{(x, y) \in X \times Y \mid y \in g(x)\}.$$

We claim $f(G) = g$. Assuming $x \in X$, note that $y \in f_G(x)$ if and only if $(x, y) \in G$, thus if and only if $y \in g(x)$, so $f_G(x) = g(x)$. Since this holds for any $x \in X$, indeed $f(G) = f_G = g$, and it follows that f is surjective. Since f is injective and surjective, it is a bijection as desired. □

0F-3. From our brief experiment with CPAChecker, it seems as though this tool has great potential for the formal verification of real-world programs. However, even in our small test cases, some validity and usability concerns arise which could limit its widespread adoption. Consider, for example, Property1a. Running the given commands, CPAChecker constructs the CFA of tcas.i, builds an abstraction, then runs through a CEGAR algorithm to determine if the automaton described by Property1a.spc can ever reach an error state. The output of this command is shown below. The error reporting is superb, with the tool clearly indicating that a violation of the spec was found, then pointing the user to a graphical output which highlights the control flow path and variable assignments which would allow the violation to occur.

```
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 11.0.9.1) 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. (PredicateCP
A: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".
```

Output for Property1a

Digging more into what Property1a actually proves, however, some downsides are apparent. The spec file itself describes an automaton that enters an error state if the label "Property1a" is reachable. Unfortunately, what this label corresponds too isn't immediately apparent, and requires looking through tcas.i to discover that the label is only reached if

```
Up_Separation >= alim && Down_Separation < alim
```

Peer Review ID: 61884965 — enter this when you fill out your peer evaluation via gradescope

is true on line 2134 of `alt_sep_test()`. This intermingling of code and specification would become more difficult at larger scales, making it difficult to tell what is actually being proved and allowing more possibilities for user error. While these properties could be modeled in the spec itself, doing so would be difficult for a non-expert.

Beyond even the scalability concerns, there are concerns with the correctness of the result itself. The properties here and tcas.i form a fairly small, unrepresentatively simple test case. A real-world program would include complexities that CPAChecker models poorly, such as pointers, complex mathematical operations, OS calls, and more complicated properties that require checking. Yet, even with these simple test cases, CPAChecker incorrectly reported that Property1a was not violated as recently as version 1.6.1. While a counterexample can now be found in version 2.0.0, it's highly likely other such bugs like are still present, which would be much more difficult to detect in large real-world programs. Couple this with potential bugs in the tools CPACheker relies on like the MATHSAT5 theorem prover, the likelihood of a false result is compounded. Despite this, CPAhecker still paints a hopeful future for formal verification, but it leaves much to be desired before such a tool can become commonplace.

```
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 11.0.9.1) 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. (PredicateCP
A: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".
```

Output for 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 11.0.9.1) 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. (PredicateCP
A: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".
```

Output for Property2b

Peer Review ID: 61884965 — enter this when you fill out your peer evaluation via gradescope

1 HW0

- **0 pts** Correct