Exercise 2. Consider the following functions $f : A \to B$ that maps $(\varphi : X \to \mathcal{P}(Y)) \in A$ to $f(\varphi) = \{(x, y) : x \in X, y \in \varphi(x)\} \in \mathcal{P}(X \times Y) = B$, and $g : B \to A$ that maps $\beta \in B = \mathcal{P}(X \times Y)$ to $(g(\beta) : X \to \mathcal{P}(Y), x \mapsto \{y : (x, y) \in \beta\}) \in A$. It can be verified that f, g are inverses of each other:

$$\begin{aligned}
f(g(\beta)) &= \{(x, y) : x \in X, y \in g(\beta)(x)\} \\
&= \{(x, y) : x \in X, y \in \{y : (x, y) \in \beta\}\} \\
&= \{(x, y) : x \in X, (x, y) \in \beta\} \\
&= \beta \; ; \\
g(f(\varphi)) &= (x \mapsto \{y : (x, y) \in f(\varphi)\}) \\
&= (x \mapsto \{y : (x, y) \in \{(x', y') : x' \in X, y' \in \varphi(x')\}\}) \\
&= (x \mapsto \{y : x \in X, y \in \varphi(x)\}) \\
&= (x \mapsto \varphi(x)) \\
&= \varphi \; .
\end{aligned}$$

Hence f, g are bijections between A and B, which demonstrates the correspondence between A and B.

Exercise 3. (I used the Docker image of version 2.0.)

Outputs for `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 (63
   ef7602814c) (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".
```

Outputs for `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 (63
   ef7602814c) (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".
```

Outputs for `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 (63
   ef7602814c) (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".
```

With the given commands, the CPAChecker verifies the program `tcas.i` against the properties specified in the files `Property{1a,1b,2b}.spc`; in particular, by inspecting the specification files, the CPAChecker checks whether the labels `PROPERTY{1A,1B,2B}` respectively are reachable in the program (where the results show that 1a, 2b are reachable while 1b is non-reachable). By further examining the program, it turns out that the program has the structure of first initializing a bunch of integer variables `Up_Separation`, `Down_Separation`, etc. *nondeterministically*, and then executing through a bunch of integer comparisons and branchings, some of which locates inside function calls, that are complexly related to each other. As an example, property 1a requires it to never be the case that `Up_Separation >= thresh && Down_Separation < thresh`, under the condition that `need_downward_RA` and not `need_upward_RA`, where `need_downward_RA` then refers to the condition `Non_Crossing_Biased_Climb()&& Own_Below_Threat()`, and where ...

This kind of verification against a complex mixture of comparisons and branchings over nondeterministic integers is a perfect task for CPAChecker, who is able to handle the "infinite" state space of the integers via abstraction that is refined adaptively. Nevertheless this test suite `tcas.i` seems too limited to demonstrate the full spectrum of the capacity of CPAChecker. E.g. there seems to be no recursion in the tested program, and even no arithmetic (only comparisons) among the integers; there are only if-else branchings and no loop; there is no array, pointer, etc. In fact in this setting (where there are only comparisons and branchings), it is conceivable that merely one pass of direct application of theorem solver, or maybe more specifically something similar to *topological sorting*, would suffice to solve the verification problem.

That being said, the CPAChecker indeed did a great job on the test suite despite the "simplicity" of the test. It ran fast enough, cost only multiple seconds to verify each property (though this is definitely far from being a satisfactory time one would expect for some more specialized, say, "topological sorting solver" as there are only tens of variables to sort over), and e.g. in the case of property 1a, produced concrete counterexample `Up_Separation = 4294967284U`, `Down_Separation = 88U`, ..., along with reasonably simplified (there is literally a "simplified" mode for displaying the *Abstract Reachability Graph*!) and readable visualizations of the results.

1 HW0

- **0 pts** Correct