Exercise 0F-2

Given two sets X and Y, we have $A = X \to \mathcal{P}(Y)$ and $B = \mathcal{P}(X \times Y)$. We define a function $f : B \to A$ as follows for all $S \in B$ (meaning $S \subset (X \times Y)$):

$$f(S) = \text{function } g : X \to \mathcal{P}(Y) \text{ where } g(x) = \{y \in Y : (x, y) \in S\}$$

We want to show that this function is a bijection. First, suppose that $f(S) = f(S')$ for some $S, S' \in B$. They both map to the same g which means that, for every $x \in X$, the sets $\{y \in Y : (x, y) \in S\}$ and $\{y \in Y : (x, y) \in S'\}$ are equivalent. If this is the case, then $S = S'$, and our f is injective.

Now, fix any $h \in A$. Construct $S = \{(x, y) \in (X \times Y) : y \in h(x)\}$. Then we have $f(S) = g$ where $g(x) = \{y \in Y : (x, y) \in S\} = \{y \in Y : y \in h(x)\} = h(x)$. So for every $h \in A$, we can construct an $S \in B$ such that $f(S) = h$. So f is surjective as well.

Since f is injective and surjective, it must be a bijective function expressing a relationship between A and B.

■

Exercise 0F-3

Property1a.spc:

```
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 2.0.1-svn / 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:

```
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 2.0.1-svn / 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:

```
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 2.0.1-svn / 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".
```

When running predicate analysis, CPAchecker converts the program into a directed graph whose transitions are code fragments and whose nodes are defined by certain predicates either holding or not holding during execution. It then performs iterative analysis to determine whether the state machine defined by this graph violates the specification – in our case, the different specifications were assertions that the program could not reach certain labels, which could only occur when the variables held certain "bad" values. We managed to prove that the PROPERTY1B label can never be reached with the code in tcas.i, while the PROPERTY1A and PROPERTY2B ones can.

With only this test case as demonstration, CPAchecker seems somewhat difficult to use on the input side. The actual meat of the properties – how `Up_Separation` relates to `thresh`, for example – is encoded in the code rather than in the spec. I take this to mean that the code has already passed through its `slic` (from SLAM) equivalent, but I think it would be easier to reason about if we started from the basic C code with a more descriptive spec. As such, I wouldn't say that tcas.i is a good test suite from a usability standpoint, though it *does* showcase examples of both valid and invalid specs.

On the *output* side of CPAchecker, though, it seems very usable. The interactive counterexample graphs that are outputted make it very easy to follow along with the invalid execution path. And I presume, based on walking through the output counterexamples myself, that CPAchecker has generated the correct results. As such, my impression of CPAchecker is generally positive, though the method of specifying correct behavior could probably be streamlined.

1 HW0

- **0 pts** Correct