2 Exercise 0F-2. Set Theory

For some sets X and Y, let $A = X \to \mathcal{P}(Y)$ and $B = \mathcal{P}(X \times Y)$, where $\mathcal{P}$ denotes the power set. Let us define some function $f : A \to B$ such that $f(a) = \{(x, y) | x \in X, y \in a(x)\}$.

Let there be an element $b \in B$, b is a set of pairs $(x, y) | x \in X, y \in Y$. We can define a corresponding $a \in A$, where for each pair, $y \in a(x)$. For any x that is not in a pair in b, $f(a) = \{\}$. For this b, $f(a) = b$. Thus, f is surjective.

Let there be some $a_1, a_2 \in A$ such that $f(a_1) = f(a_2) = b \in B$. By definition, for each $(x, y) \in b$, $y \in a_1(x)$ and $y \in a_2(x)$. By iterating through all the pairs that begin with a given x, this uniquely defines the values of $a_1(x)$ and $a_2(x)$. A function is uniquely defined by its set of inputs and outputs, thus $a_1 = a_2$. Therefore, f is injective.

Thus, f is a bijection.

3 Exercise 0F-3. Model Checking

Output for Property 1a

```
Language C detected and set for analysis (CPAMain.
   detectFrontendLanguageIfNecessary, INFO)
Using the following resource limits: CPU-time limit of 900s
(ResourceLimitChecker.fromConfiguration, INFO)
CPAchecker 4.0 / predicateAnalysis (OpenJDK 64-Bit Server VM 17.0.13)
started (CPAchecker.run, INFO)
Parsing CFA from file(s) "tcas.i" (CPAchecker.parse, INFO)
Using predicate analysis with MathSAT5 version 5.6.10 (9293adc746be) (May 31 2023
    12:38:06, gmp 6.2.0, 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".
```

Output for Property 1b

```
Language C detected and set for analysis (CPAMain.
   detectFrontendLanguageIfNecessary, INFO)
Using the following resource limits: CPU-time limit of 900s (ResourceLimitChecker
   .fromConfiguration, INFO)
```

```
CPAchecker 4.0 / predicateAnalysis (OpenJDK 64-Bit Server VM 17.0.13) started (
    CPAchecker.run, INFO)
Parsing CFA from file(s) "tcas.i" (CPAchecker.parse, INFO)
Using predicate analysis with MathSAT5 version 5.6.10 (9293adc746be) (May 31 2023
     12:38:06, gmp 6.2.0, 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".
```

Output for Property 2b

```
Language C detected and set for analysis (CPAMain.
    detectFrontendLanguageIfNecessary, INFO)
Using the following resource limits: CPU-time limit of 900s (ResourceLimitChecker
    .fromConfiguration, INFO)
CPAchecker 4.0 / predicateAnalysis (OpenJDK 64-Bit Server VM 17.0.13) started (
    CPAchecker.run, INFO)
Parsing CFA from file(s) "tcas.i" (CPAchecker.parse, INFO)
Using predicate analysis with MathSAT5 version 5.6.10 (9293adc746be) (May 31 2023
     12:38:06, gmp 6.2.0, 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".
```

The file tcas.i contains relevant subsets of a C program (including tcas.c and its headers) and is annotated with labels that correspond to specific properties that may lead to errors. Each property file checks whether the associated label is reached during the program's execution. The tool either verifies that the label cannot be reached or generates a counterexample, including the line number and path that violate the property. The provided code was too large for manual analysis but could be verified relatively quickly. It revealed one error related to property 2b, indicating that it serves as a reasonable test suite in this context.

CPAchecker, when operating in predicate analysis mode, requires both a specification file and a segment of C code for verification. Upon completion, it returns the verification result—either success or a counterexample—and generates an interactive HTML report. In

this case, we have confirmed that properties 1a and 2b hold for the provided code segments, while property 1b does not hold along the specific path indicated by the counterexample. The HTML report includes statistics on the number of abstraction steps taken to verify or generate a counterexample. For counterexamples, it features a control flow graph highlighting the failing path and provides navigation back to the corresponding lines of source code.

Overall, I find CPAchecker to be a usable and effective tool within the limits of its inputs. It is the responsibility of the user to define a good initial set of properties and to provide a relevant sample of the target software for verification. The tool enables users to confirm defined properties on specific code selections, but it is up to the user to ensure that the selection of code and properties is useful for effective analysis.