## 2 Exercise 0F-2. Set Theory

We want to show there is a bijection between the set of functions (A)

$$f : X \to \mathcal{P}(Y)$$

and the power set (B)

$$\mathcal{P}(X \times Y).$$

Given the function $f : X \to \mathcal{P}(Y)$, we note that $f(x)$ is a subset of $Y$ for each $x \in X$. Therefore by pairing $x$ with each $y \in f(x)$, we can form another subset named $R_f$:

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

Specifically, $(x, y)$ belongs to $R_f$ when $y$ is an element of the subset $f(x) \subseteq Y$. This assignment:

$$f \;\mapsto\; R_f$$

gives a function mapping A to B

$$g : \{\, f : X \to \mathcal{P}(Y)\} \;\longrightarrow\; \mathcal{P}(X \times Y).$$

### Injectivity

To show that $g$ is injective, we must prove that if $g(f_1) \neq g(f_2)$, then $f_1 \neq f_2$. This is because the subsets $R_f \subseteq X \times Y$ are completely determined by the behavior of $f$.

Let $f_1, f_2 : X \to \mathcal{P}(Y)$ be two different functions. By the definition of a function, if $f_1 \neq f_2$, there must exist some $x \in X$ such that:

$$f_1(x) \neq f_2(x).$$

This means there is at least one element $y \in Y$ such that $y \in f_1(x)$ but $y \notin f_2(x)$ or vice versa. By the definition of $R_f$, this implies:

$$(x, y) \in R_{f_1} \quad \text{but} \quad (x, y) \notin R_{f_2}.$$

Thus, $R_{f_1} \neq R_{f_2}$. Since $g(f_1) = R_{f_1}$ and $g(f_2) = R_{f_2}$, it follows that:

$$g(f_1) \neq g(f_2).$$

This shows that if $f_1 \neq f_2$, then $g(f_1) \neq g(f_2)$, so $g$ is injective.

### Surjectivity

To show that $g$ is surjective, we must prove that for every subset

$$R \subseteq X \times Y,$$

there exists a function

$$h : X \to \mathcal{P}(Y)$$

such that

$$g(h) = R,$$

where $g(h) = R_h$ and

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

Then let $R \subseteq X \times Y$ be an arbitrary subset. We want to construct a function $h$ so that $g(h) = R$. This function $h$ can be defined by:

$$h(x) \;=\; \{\, y \in Y \;\mid\; (x, y) \in R\}.$$

That is, for each $x \in X$, $h(x)$ is the set of those $y \in Y$ for which $(x, y) \in R$.

By this definition, it follows that

$$g(h) \;=\; \{(x, y) \in X \times Y : y \in h(x)\} \;=\; \{(x, y) : (x, y) \in R\} \;=\; R.$$

Thus, for any subset $R \subseteq X \times Y$, we have found a function $h$ with $g(h) = R$. This proves that $g$ is surjective.

Question assigned to the following page: 2

### Conclusion

Since $g$ is both injective and surjective, it is a bijection and we have a one-to-one correspondence between A and B.

Question assigned to the following page: 3

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

**Property1a Output**:

```
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 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".
```

**Property1b Output**:

```
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 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,
```

Question assigned to the following page: 3

```
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".
```

**Property2a Output**:

```
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 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".
```

**Analysis** When we run the given commands, CPAChecker executes a predicate analysis algorithm to determine feasible program execution paths. We specify that it should use the input file tcas.i and a provided specification file (e.g., Property1A.spc) to check whether a specified property is reachable. These properties all concern maintaining safe altitude separation between aircraft. The file tcas.i is a valid test suite because it deals with a safety-critical piece of software (flight collision avoidance) that would clearly benefit from thorough safety verification before being deployed in real-world conditions. Furthermore, with over 2,160 lines of code, and presumably already preprocessed from a C file (as suggested by its .i extension), tcas.i is sufficiently complex to demonstrate the value of CPAChecker.

Question assigned to the following page: 3

CPAChecker accomplishes its analysis by first parsing tcas.i into a control-flow automaton (CFA). It then applies a predicate analysis algorithm to verify the program's correctness by computing reachable states.[1] When CPAChecker reports an error for Property1A, it indicates that the property is indeed reachable under some set of initial conditions. This corresponds to an error state, and CPAChecker notes that the property is found to be reachable at line 1963. While that line number alone may not be particularly illuminating, CPAChecker generates additional output in its results directory that can help clarify the path to this error. For example, CounterExample.1.html provides a web interface that includes an Abstract Reachability Graph (ARG), illustrating the program flow from the main function to the erroneous line. It also provides the counterexample in multiple formats, including text (.txt), YAML (.yml), and DOT (.dot) files.

For critical applications, I find CPAChecker to be a useful tool, though it does have some limitations. In particular, it relies on pattern matching as defined in the specification files to detect whether a labeled state is reachable. Creating these specification files is relatively straightforward since they are typically under 10 lines and use a simple syntax. However, the programmer must place the labels in the code with sound logic that captures unsafe or undesirable conditions. If this logic is not well-defined, CPAChecker may fail to detect an error, possibly giving the mistaken impression that the program is safe. On the positive side, the additional output generated by CPAChecker is especially helpful. For example, the HTML version of the counterexample displays the ARG, the CFA, log information, statistics, and configuration details. These tools aid in visualizing the path taken through the program that leads to the identified erroneous state.

## References

[1] Dirk Beyer and M. Erkan Keremoglu. Cpachecker: A tool for configurable software verification, 2009.