Exercise 0F-2. Set Theory [5 points]. This exercise is meant to help you refresh your knowledge of set theory and functions. Let X and Y be sets. Let $\mathcal{P}(X)$ denote the powerset of X (the set of all subsets of X). There is a 1-1 correspondence (i.e., a bijection) between the sets A and B, where $A = X \to \mathcal{P}(Y)$ and $B = \mathcal{P}(X \times Y)$. Note that A is a set of functions and B is a (or can be viewed as a) set of relations. This correspondence will allow us to use functional notation for certain sets in class. This is Exercise 1.4 from page 8 of the Winskel textbook.

Demonstrate the correspondence between A and B by presenting an appropriate function and proving that it is a bijection. For example, you might construct a function $f : B \to A$ and prove that f is an injection and a surjection.

Solution:

Proof. Let $(f(b))(x) = \{y : (x, y) \in b\}$. We wish to prove that this function is a 1-1 correspondence between A and B.

First, we must prove that f is an injection. To do this, we must show that

$$b_1 \neq b_2 \implies f(b_1) \neq f(b_2) \tag{1}$$

for all $b_1, b_2 \in B$. Suppose that, for $b_1, b_2 \in B$, $b_1 \neq b_2$. This means that

$$\{(x, y) \in b_1 : x \in X \text{ and } y \in Y\} \neq \{(x, y) \in b_2 : x \in X \text{ and } y \in Y\}. \tag{2}$$

As a result, it must be the case that

$$\exists x' \in X \,.\, \{y : (x', y) \in b_1\} \neq \{y : (x', y) \in b_2\} \tag{3}$$

because otherwise $b_1 = b_2$, meaning that inequality (2) would not be true. As a result of inequality (3), it must be true that $f(b_1) \neq f(b_2)$ because otherwise, by definition of f,

$$\forall x \in X \,.\, \{y : (x, y) \in b_1\} = \{y : (x, y) \in b_2\}$$

which directly contradicts inequality (3). Thus, we have shown statement (1) to be true, which proves that f is an injection.

Now, we prove that f is a surjection. To do this, we must show that

$$\forall a \in A \,.\, \exists b \in B \,.\, f(b) = a \tag{4}$$

Suppose we have an arbitrary $a \in A$. Now, suppose we have a $b \in B$ defined as

$$b = \{(x, y) : y \in a(x)\} \tag{5}$$

where $x \in X$ and $y \in Y$. From here, we have

$$\begin{aligned}(f(b))(x) &= \{y : (x, y) \in b\} \\ &= \{y : (x, y) \in \{(x', y) : y \in a(x')\}\} \\ &= \{y : y \in a(x)\} \\ &= a(x)\end{aligned}$$

where $x, x' \in X$. Thus, $f(b)$ is exactly the function a. Because a is arbitrary, this proves statement (4), which proves that f is a surjection.

We have proved that f is both an injection and a surjection. By definition, this means that f is a 1-1 correspondence between A and B. □

Exercise 0F-3. Model Checking [10 points]. This answer should appear after the first page of your submission and may be shared during class peer review.

Download the CPAChecker software model-checking tool using the instructions on the homework webpage. Read through enough of the manual to run the tool on the `tcas.i` testcase provided on the homework webpage. Check the three properties given. For each command, copy or screenshot the last ten non-empty lines of output from CPAChecker and include them as part of your answer to this question.

It is your responsibility to find a machine on which CPAChecker works properly (but feel free to check the class forum if you are getting stuck).

Hint: CPAChecker 2.0 should find a violation for `Property1a`, verify that `Property1b` is safe, and find a violation for `Property2b`. If your output does not match that and you are using version 2.0 then you may not have not set things up correctly.

What is going on when you run CPAChecker using the commands listed? In at most three paragraphs, summarize your experience with the CPAChecker tool. What does `Property1a` mean? Is `tcas.i` a reasonable test suite? What has been proved? Did you find CPAChecker to be a usable tool? You may find the graphical reporting option of CPAChecker to be helpful here. For full credit, do not restate my lecture on counter-example guided abstraction refinement; instead, discuss your thoughts and experience using this tool. Focus on threats to validity (e.g., imagine that you were writing a paper and using this as an experiment) over usability.

Both your ideas and also the clarity with which they are expressed (i.e., your English prose) matter. A reader should be able to identify your main claim, the arguments you are making, and your conclusion.

Solution:

Last 10 lines of output 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 (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".
```

Last 10 lines of output 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 (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".
```

Last 10 lines of output for Property2a:

```
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. (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".
```

Summary of Experience with CPAChecker:

While CPAChecker is usable in terms of visualizing control flow automata (CFAs) and counterexample error paths, the ability to search CFAs and error paths could be improved. Additionally, its report functionality in the case of verification could provide clearer proof of verification. On the other hand, CPAChecker's Statistics feature is informative, and the ability to define and test properties with CPAChecker is simple.

CPAChecker's graphical HTML reports provide a usable interface for visualizing CFAs and error paths. For example, when the `ERROR` label was reached while checking Property1a, CPAChecker's counterexample report included a side-by-side comparison of the error path and its corresponding CFA edges, which was useful for understanding how the error was generated. Although CPAChecker's visualization tools are useful, the ability to search through paths and visualizations could be enhanced. Specifically, when searching for variable names in the error path, direct uses do not appear; it only shows the number of matches. Additionally, it would be useful if there were more explanation in the verification reports (for cases where no error path was found). It wasn't clear whether the CFA, ARG, and other information in the report constituted a proof of verification. While CPAChecker's usability and verification reports have flaws, CPAChecker does have informative statistics, and it is easy to add checks on new properties. CPAChecker provides a vast set of statistics in the Statistics section of its reports. These statistics include time to execute its algorithms, memory consumption, code coverage, and other relevant data. Software engineers can use these statistics to make assessments about the validity and quality of their code. For instance, a low line coverage could indicate that potential bugs outside of the checked properties may be missed. Likewise, long execution times and an abnormally high number of predicates discovered could raise safety concerns related to code structure, as poorly structured code may be harder to verify. In addition to statistics, it is also easy to add new property checks. For example, `Property1a.spc` spans less than 11 lines. When the `PROPERTY1A` label is encountered and the label of the successor node in the CFA is `ERROR`, Property1a is violated. A new property specification file could be similarly be created that matches a different label. Thus, although `tcas.i` is a minimal test suite consisting of a single test that does not check all properties in all cases (thus inhibiting the usefulness of CPAChecker), new property checks could easily be added as more tests are created.

In conclusion, CPAChecker is a usable tool but could be better, especially with respect to its path searching functionality and its clarity of verification proof. Though these pitfalls exist, CPAChecker provides an expansive set of statistics that developers can use to analyze their code quality, and new property checks can easily be added.

1 HW0

- **0 pts** Correct