Exercise 0F-2. Set Theory [5 points]. This answer should appear after the first page of your submission and may be shared during class peer review.

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) bewteen 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.

We define our function f(b)(x) = a(x) = $\{y|(x, y) \in b\}$ on $B \to A$. Conceptually, given b, an element of B, we map it to a, an element of A, which is a function that outputs some power set of Y for every input x in X. Since B is the power set of $X \times Y$, each b is simply a set of (x,y) pairs. a on input x' will look at all the (x,y) pairs in b and will output all y such that (x',y) is a pair in b. If there are no pairs that contain x', then it outputs the empty set, which is also an element of P(y). We need to show that this is injective and surjective to prove that is a bijection.

For injectivity, we must show that given $b_1, b_2 \in B$ such that $f(b_1)(x) = f(b_2)(x)$, then $b_1 = b_2$. We do this as follows. If $f(b_1)(x) = f(b_2)(x)$, then for any $x' \in X$, $f(b_1)(x') = f(b_2)(x')$. Call this output py where py is an element in the powerset of Y. This means, by definition of f, for each y' in py, (x', y') is a pair in b_1 and b_2 as otherwise that y' would not be in the output of the function for x'. Additionally, for every y' in Y but not in py, (x', y') is not a pair in b_1 and b_2 as otherwise that y' would be in the output of the function for x'. This means that since the functions $f(b_1)(x)$ and $f(b_2)(x)$ are the same, b_1 and b_2 consist of the same pairs which means $b_1 = b_2$ and we have shown injectivity.

For surjectivity, we must show that for any $a(x) \in A$, there exists $b \in B$ such that $f(b)(x) = a(x)$. We show this as follows. For any $a(x) \in A$, we construct b as a set of pairs. To do this, we iterate through each $x' \in X$. For each x', we run a(x')=py, where py is an element of the power set of Y. Then, for each $y' \in py$, we add (x',y') to b. Once this is done, we see that b contains all the pairs (x,y) such that $y \in a(x)$ by definition, which means f(b)(x)=a(x). This means we have show that f(b)(x) is surjective.

Since we have shown that f(b)(x) mapping B to A is injective and surjective, this means it is bijective, and A and B are bijections of each other.

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.

When running these commands, CPAChecker uses the specification provided (property1a, 1b, 2b) to check the program and make sure the spec is not violated; here the program in use is the tcas.i Traffic Collision and Avoidance System. While running, CPAChecker will look to see if there is an execution path that results in the specification being violated. If it doesn't find one, the program passes the specification. If it does find one, it will look for variable assignments that form a counter-example which show how the specification can be violated. In the given specifications, it looks like CPAChecker is looking for a path that leads to a specifically labelled error. For property1a, this is the error on line 1965 of tcas.i which has the label "property1a". Similarly, the spec for property1b will look to see if the error labelled "property1b" is reachable and for property2b, it will look to see if the error labelled "property2b" is reachable. Specifically, the error labelled property1a is reached if $up\_separation >= thresh$ and $down\_separation < thresh$ and $need\_downward\_RA$ is true.

Based on the results using CPAChecker 1.6.1, it looks like property1a and 1b are not violated while 2b is violated. The tool has proven that there is a set of arguments that lead to property1b being violated, i.e. there is a sequence of arguments that leads to the execution path ending at the error labelled "property1b". The system also claims that property1a and

property1b are not violated, that is, the errors with those labels are never reached. Now, based on discussion on the Piazza forum, it seems CPAChecker 2.0 DOES find a path that violated property1a, so saying that the system has "proven" property1a cannot be violated is a bit of a strong term. I am personally a bit concerned about this, since based on what we learned in lecture and in the papers so far, it seems false positives are more of a concern and false negatives don't usually exist. This means that either property1a is a false negative in CPAChecker 1.6.1 which is bad as this is something that would slip past a programmer, or in CPAChecker 2.0 it is a false positive. As a false positive, it is something the programmer can manually check, but its concerning that a newer version of the tool would have more false positives. Based on this, I think tcas.i is a good test case (though it's nowhere near comprehensive as we're only checking a very limited set of features using the three specifications). The fact that over different versions of CPAChecker property1a acts differently indicates that its a good test case to analyze the behaviour and performance of CPAChecker. I would like to compare the output from version 2.0 to see if this is a false positive or negative.

I think CPAChecker is good tool considering it evaluated these properties very quickly and for the violation of property2b, it created a counterexample that clearly states the variable values that lead to the error. Even if this is a false positive, the developers can test these values out and follow the execution path that CPAChecker has drawn out to confirm whether or not it is a false positive. The drawn out CFGs are very nice as well for following along, though I feel that it may be hard to follow at times for very large programs. I think it is a usable tool that does its job well, though in the future, perhaps providing a GUI option as well allowing for quicker analysis and incorporation of the CFGs generated will be extremely helpful. However, some of the issues mentioned earlier regarding property1a concern me about the tool, but it does seem to prove a good amount of detail to reproduce its findings for counterexamples and is an efficient model checker.

Below are the screenshots of the output from CPAChecker 1.6.1

```
Using the following resource limits: CPU-time limit of 900s (ResourceLimitChecke
r.fromConfiguration, INFO)

CPAchecker 1.6.1 (OpenJDK 64-Bit Server VM 1.8.0_282) started (CPAchecker.run, I
NFO)

Using predicate analysis with SMTInterpol 2.1-238-g1f06d6a-comp and JFactory 1.2
1. (PredicateCPA:PredicateCPA.<init>, INFO)

No invariants are computed (PredicateCPA:InvariantsManager.<init>, INFO)

Using refinement for predicate analysis with PredicateAbstractionRefinementStrat
egy 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"
.
Run /afs/umich.edu/user/s/a        eecs590/HW0/CPAchecker-1.6.1-unix/scripts/rep
ort-generator.py to show graphical report.
```

Figure 1: CPAChecker output for property1a shows no violations

```
Using the following resource limits: CPU-time limit of 900s (ResourceLimitChecker.fromConfiguration, INFO)

CPAchecker 1.6.1 (OpenJDK 64-Bit Server VM 1.8.0_282) started (CPAchecker.run, INFO)

Using predicate analysis with SMTInterpol 2.1-238-g1f06d6a-comp and JFactory 1.21. (PredicateCPA:PredicateCPA.<init>, INFO)

No invariants are computed (PredicateCPA:InvariantsManager.<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".
Run /afs/umich.edu/user/s/a/        ecs590/HW0/CPAchecker-1.6.1-unix/scripts/report-generator.py to show graphical report.
```

Figure 2: CPAChecker output for property1b shows no violations

```
No invariants are computed (PredicateCPA:InvariantsManager.<init>, INFO)

Using refinement for predicate analysis with PredicateAbstractionRefinementStrategy strategy. (PredicateCPA:PredicateCPARefiner.<init>, INFO)

Starting analysis ... (CPAchecker.runAlgorithm, INFO)

Error path found, starting counterexample check with CPACHECKER. (CounterexampleCheckAlgorithm.checkCounterexample, INFO)

Using the following resource limits: CPU-time limit of 900s (CounterexampleCheck:ResourceLimitChecker.fromConfiguration, INFO)

Repeated loading of Eclipse source parser (CounterexampleCheck:EclipseParsers.getClassLoader, INFO)

Error path found and confirmed by counterexample check with CPACHECKER. (CounterexampleCheckAlgorithm.checkCounterexample, INFO)

Stopping analysis ... (CPAchecker.runAlgorithm, INFO)

Verification result: FALSE. Property violation (error label in tcas.i, line 1997) found by chosen configuration.
More details about the verification run can be found in the directory "./output".
Run /afs/umich.edu/user/s/a/        ecs590/HW0/CPAchecker-1.6.1-unix/scripts/report-generator.py to show graphical report.
```

Figure 3: CPAChecker output for property2b shows there is a violation

1 HW0

- **0 pts** Correct