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.

Answer to 0F-2

A is the set of all functions from X onto the powerset of Y, and B is the set which is the powerset of the cartesian product of X and Y. Let us define a function f which maps members of set B to members of set A, taking in some arbitrary member of B designated b and outputting some function a in A. f is defined as the following function: $f(b) = \{a(x) \to Y' | (x, y) \in b, y \in Y'\}$ where Y' is the set of members of the set Y for which a tuple (x,y) exists in b. We observe that if there are no such tuples, a({}) → {}; this is correct, as the empty set is a subset of Y. It is apparent that the output of f, a, maps some x in X to some subset of Y, which is a member of the power set of Y; therefore, a must be in A.
We can show that f is bijective by creating a function from A to B, g, and showing that g is an inverse for f. Let us define $g = \{\{(x, y_i), \dots\} | a(x) \to Y', y_i \in Y'\}$. The output of g is a set of tuples, each of which contains one item in X and one item in Y; this is a subset of the powerset of the cartesian product of X and Y; therefore, we know g takes an item in A and outputs an item in B.
Now, let us show that f and g are inverse functions. Without loss of generality, let us take b, an arbitrary member of B, and use $g(f(b))$ to show that for all B compositing the functions returns the original value. We observe that as B is the powerset of a set of tuples (x, y) for some x in X and y in Y, any b must consist of a set containing some quantity (—b— could be zero to uncountably infinite) of these tuples, each with one item from x and one item from y. Applying $f(b)$ gives us some a such that for all (x',y') ∈b, a(x') → Y where Y is a set containing y'. (This is true for b = {}; the resulting function is just {} → {}). We can take g(a). For all x' in the domain of a, we produce tuples (x",y") where y" is in Y, and we denote the returned set containing all of these b".
Now, we must show that b" must be equal to b. Let us say that they are not; either b" has at least one tuple which b lacks, or b" lacks a tuple in b (or both). If b" has a tuple not in b, then either in converting from b to a or in converting from a to b" a term was added. No terms are added when b is transformed into a; Y' is only a set containing $y \in b$. No terms are added when a is transformed into b" either; all y_i stem from Y'. So this is impossible. Similarly, as f and g transform all y_i and Y', no tuples can be lost. Therefore, b" and b are

sets which contain all the same tuples, and are therefore equivalent.
Intuitively, this g(f(b)) can be thought of as “factoring out” the X-term in each of the (x,y) pairs in b, then making a set of all the ys which had the x in that tuple and declaring a to be the function returning that. Then, g(a) “distributes” the x back into those pairs.

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.

Answer to 0F-3

```
1a)
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".
```

1b)

```
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".
```

2b)

```
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 ch
More details about the verification run can be found in the directory "./output".
Graphical representation included in the file "./output/Counterexample.1.html".
```

I did not find CPAchecker useful for the given program and predicates. When I run CPAchecker, the program takes in a specification (the .spc files) and the C file without #define or #includes (tcas.i) and instructions on how to apply them (the -predicateAnalysis flag). It transforms the tcas file into a simplified automaton according to the -predicateAnalysis flag and properties given, which specify that the checker should use adjustable-block encoding (modifiable-length tree depth before restarting) and search for blocks which satisfy the error condition. As the program tcas.i is relatively simple and the properties trivial without abstraction, it finds or does not find this quickly, and then outputs either a successful path to an error or a proof that it is impossible to ever have an error with that property. It finds error on 1a and 2b but not 1b.

Property 1a defines an automaton. It starts at an initial state and checks all subsequent states. If the CFA's label matches a regex (spelling out "Property1a", with capitalization optional) and the subsequent state is an error state, it returns a successful path to an error state. This is the case (in tcas.i, if down_separation is less than a threshold and up_separation is at or above it) and so Property1a has a violation in line 1963. The only thing which is proved is that given the constants, there is a way for the ALIM (altitude parameter) to lead to an improper height separation and an error. This is not a lot of information, and does not really help analyze the validity of the model used by the program.

I think that tcas.i and the predicates given are poor test cases for CPAchecker. These are very simple, just checking if a node's direct successor has the label Error. CPAchecker does not need to come up with difficult predicates or do refining; predmap.txt, the list of predicates which the program generates, is empty. The properties (1a, 1b, 2b, etc) are explicitly spelled out in the tcas.i file. Only one location in the problem is labeled with each of these properties, if we look at the CFA graph. CPAchecker would be better-tested by difficult predicates for which generating abstract predicate trees would be useful, instead of this small one where the tree depth parameter is never reached. Also, the actual code in the tcas.i file is very short–it looks long due to the fact that all the #includes are preprocessed and included by the compiler, but there really isn't much directly related to and tested by the program. For all these reasons, I did not really find CPAchecker useful for TCAS analysis, although it was in theory usable. I am sure it could be useful elsewhere, on more-complex programs where higher levels of abstraction are necessary for model checking. This model never needs to be refined, which is counter to the reason CEGAR is useful and valuable.

Submission. Turn in your assignment as a single PDF document via the `gradescope` website. Your name and Michigan email address must appear on the first page of your PDF submission but may not appear anywhere else.

1 HW0

- 0 pts Correct