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: All $b \in B$ has the form $\{(x, y), ..., (x_i, y_j) | x \in X, y \in Y)\}$. And since A is a set of functions of type $X \to \mathcal{P}(Y)$, all $a \in A$ has the form $\{(x, \{y,..., y_j\}), (x_i, \{y,..., y_j\}) | x \in X, y \in Y\}$. In other words, $f(b)(x) = \lambda x.\{b'[1] \in b | b'[0] = x\}$. This can be stated as a set union instead of a lambda via the definition of functions as a form of relations over sets, something like $f(b) = \forall_{t \in b}\{(t[0], \{t[1] | t[0] = t\})\}$.

The function f is injective. By contradiction.

Proof. For the function $f : B \to A$ to be injective, the following must hold: $\forall_{b,b' \in B}, f(b) = f(b') \implies b = b'$. Towards contradiction, assume our f is not injective, i.e. $\exists_{b,b' \in B} f(b) = f(b') \wedge b \neq b'$. This means some b and b' differ in at least one element, $b = \{..., (x_i, y_j), ...\}, b' = \{..., (x'_i, y'_j), ...\}$ but both $f(b)$ and $f(b')$ produced $\{..., (x_i, \{..., y_j, ...\}), ...\}$. If $x_i \neq x'_i$ or/and $y_j \neq y'_j$ then the resulting partition would also be different in f, and thus $f(b) \neq f(b')$, which is a contradiction. □

The function f is surjective. By contradiction.

Proof. For the function $f : B \to A$ to be surjective, the following must hold: $\forall_{a \in A} \exists_{b \in B} f(b) = a$. Towards contradiction, assume $\exists_{a \in A} \forall_{b \in B} f(b) \neq a$. This means a is such $\{...(x_i, \{y, ..., y_j\})...\}$ such that there is no b that has $\{...(x_i, y), ..., (x_i, y_j), ...\}$. We can take this a and form a set of form B by "distributing" the x_i into the corresponding y_j set, to create pairs. Then given $B = \mathcal{P}(X \times Y)$ this b is in B. □

Thus f is bijective.

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.

```
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. (PredicateC
PA:PredicateCPA.<init>, INFO)

Using refinement for predicate analysis with PredicateAbstractionRefinementStrate
gy 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 b
y 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".
```

Figure 1: Spec 1a

```
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. (PredicateC
PA:PredicateCPA.<init>, INFO)

Using refinement for predicate analysis with PredicateAbstractionRefinementStrate
gy 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".
```

Figure 2: Spec 1b

Answer: CPAChecker performers verification of properties of software artifacts by using a combination of data-flow analysis and verification techniques. These techniques can be configured by the user depending on the precision and scalability desired, as well as dynamically by the tool itself. The base analysis technique is a dataflow algorithm used for reachability analysis. Additionally, CEGAR is enabled by default. The outputs above are produced by

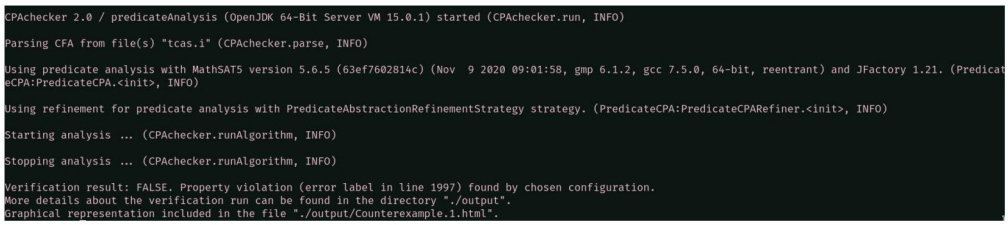

Figure 3: Spec 2b

running `CPA` with the `predicateAnalysis` flag enabled as well. The command passes the specification and the program to be verified under this configuration.

`Property1a` is the property that `UP_separation` is above the threshold and the `Down_separation` is under the same threshold. The program is a traffic collision avoidance system, so it can be safely inferred that `Property1a` is ensuring the distance between vehicles, in the up and down direction, is large enough, and specifically under the threshold distance. CPAChecker found a violation for this property, so it's proven by counter-example that the traffic control system does not always prevent vehicles from getting too close. Although CPAChecker was able to easily verify all the three properties here, I don't think the program `tcas.i` is a good test suite for CPAChecker because the program state size is too small. This can be observed from the statistics reported by CPAChecker: only two abstractions used, one predicate discovered, and the BDD constructed had only 202 nodes. The program also doesn't have any loops as can be seen from the source code and the report from CPAChecker. A reasonable test suite should emulate industry applications, both in scale and characteristics, since the goal is to provide useful tools that software engineers can utilize to build correct applications.

However, CPAChecker was a fairly usable tool in general. The documentation was comprehensive and well organized. The tool itself was straightforward to use thanks to the automating scripts provided. Although we did not have to do it, writing the properties in the specification language seems trivially easy as well. However, by the same token I am concerned about the usability of the specification language if the property was not as easily expressible, as is the case in most industry-scale applications. Additionally, CPAChecker offers a lot of configuration options with various trade-offs, such as precision and scalability. Configuring the tool for a specific program thus becomes the software engineer's responsibility. This additional workload on the software engineer seems too high and likely to be misused without further guidance from the tool, either through automatic configuration or thorough documentation.

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