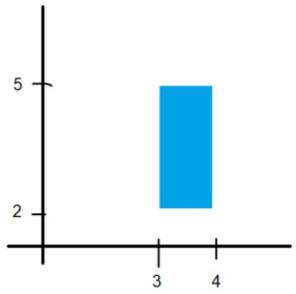
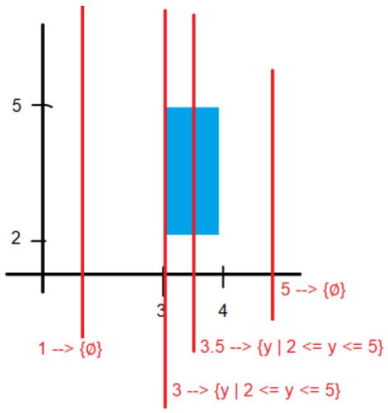
Exercise 0F-2

Take the function f: B → A as follows: B is a shape on a 2D-graph, and A is the set of vertical slices of said shape. For this, let both X and Y be the set of real numbers. Then, if we define both A and B on a 2D graph, with X and Y being depicted as the x-coordinate and y-coordinate respectively, A will be depicted as the sets of vertical slices of a given shape on the graph, while B will be depicted as the sets of shapes possible on the graph.

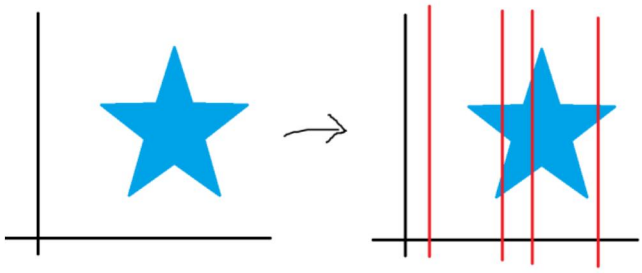
For example, a particular element of B, call it "b" for now, could be the set of (x,y) with x from 3 to 4 and y from 2 to 5. Then on the graph, b will look like this:



If we put this b through the function f, the corresponding result a (an element from the set A) will be the set of vertical slices of the diagram from above:



Different elements from B will result in different shapes being formed on the graph, which will in return result in different sets of vertical slices of the graph, i.e. different elements from A.



What's more, different (unique) collections of vertical slices will result in different (unique) shapes on the graph, which means there is a function g: A → B that is an inverse of f. Thus f is bijective.

Exercise 0F-3

Note: I'm using Ver 1.6.1 on a CAEN machine.

Property1a: TRUE

```
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.
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[redacted]EECS590/CPAchecker-1.6.1-unix/scripts/report-generator.py to show graphical report.
```

Property1b: TRUE

```
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.
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[redacted]EECS590/CPAchecker-1.6.1-unix/scripts/report-generator.py to show graphical report.
```

Property2b: FALSE

```
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/use[redacted]CPAchecker-1.6.1-unix/scripts/report-generator.py to show graphical report.
```

When CPAchecker is run with the listed commands, it analyzes the program tcas.i, checking to see if there are any paths leading to an error label listed in the .spc file fed into the program through its arguments. Property1a.spc checks for labels named "Property1a", Property1b.spc checks for "Property1b", and Property2b checks for "Property2b". Reading through the tcas.i file, it seems that the label "Property1a" is encountered when Up_Separation >= thresh and Down_Separation < thresh. Since testing for this property passed through CPAchecker, I can assume that that situation will never occur

realistically. Same for “Property1b”, which occurs when Up_Separation < thresh && Down_Separation >= thresh. Meanwhile, “Property2b”, which is encountered when Up_Separation < thresh && Down_Separation < thresh && Up_Separation < Down_Separation returns FALSE, which signals that there is a counterexample where the above situation can occur realistically. The graphical report might have given me some more insight, but the report was never generated due to encoding issues in the python files.

My experience with this tool has been really harrowing, to be honest. I am using a Windows 10 laptop at home, and the program really doesn’t like it for some reason. Trying to run the binary on my laptop on the BASH for Windows terminal did not work, nor did cloning the repository and trying to compile it by myself. Docker did not want to be installed in my laptop as well somehow, so that was out of the question as well. I went on CAEN in the end, but even that proved difficult – I wasn’t able to change the Java version used in the machines, so version 2.0 did not work. Only by going onto Piazza would I learn that version 1.6 would work on CAEN. And even after all that, the graphical report (which is mentioned in hw0.pdf to be helpful) did not compile, due to some encoding issues in the python files. So overall, I found CPAchecker to be almost unusable.

1 HW0

- **0 pts** Correct