Q2

Define function $f : B \to A$.

- The input R , where $R \in B = P(X \times Y)$, represents a set of relations, i.e. $R \subseteq X \times Y$.
- The output F, where $F = f(R)$ and $F \in A = X \to P(Y)$, is also a function that maps an element $x \in X$ to a set of $y \in Y$. Let F defined as: $F(x) = f(R)(x) = \{y \in Y | (x, y) \in R\}$.
- In physical meaning, given a few (x, y) tuples as the input relation R, the function f returns a mapping that aggregates the tuples, mapping each x to its related y's.

Proof of f being injective:

- To prove f being injective, we show $\forall F_1, F_2 \in A$, where $F_1 = f(R_1)$ and $F_2 = f(R_2)$, if $F_1 = F_2$, then $R_1 = R_2$.
- $F_1 = F_2$

  $\Rightarrow \forall x, F_1(x) = F_2(x)$
  $\Leftrightarrow \forall x, \{y \in Y | (x, y) \in R_1\} = \{y \in Y | (x, y) \in R_2\}$

  $\Rightarrow R_1 = R_2$

Proof of f being surjective:

- To prove f being surjective, we show $\forall g \in A, \exists R \in B$ such that $f(R) = g$.
- $g \in A \Rightarrow g : X \to P(Y)$, and note that $g(x) = \{y \in Y | y \in g(x)\}$ $(\star)$
- we can always construct $R = \{(x, y) \in X \times Y | y \in g(x)\}$ $(\star\star)$
- such that $\forall x, f(R)(x) = \{y \in Y | (x, y) \in R\}$
  by $(\star\star)$
  $= \{y \in Y | y \in g(x)\}$
  by $(\star)$
  $= g(x)$
  $\Rightarrow f(R) = g$

Since $f : B \to A$ is both injective and surjective, A and B are bijective.

Q3

Property1a

```
docker run -v $(pwd):/workdir -u $UID:$GID \
  sosylab/cpachecker -predicateAnalysis -spec Property1a.spc tcas.i
```

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

CPAchecker 4.0 / predicateAnalysis (OpenJDK 64-Bit Server VM 17.0.13) started (CPAchecker.run, INFO)

Parsing CFA from file(s) "tcas.i" (CPAchecker.parse, INFO)

Using predicate analysis with MathSAT5 version 5.6.10 (9293adc746be) (May 31 2023 12:38:06, gmp 6.2.0, 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".
```

Property1b

```
docker run -v $(pwd):/workdir -u $UID:$GID \
  sosylab/cpachecker -predicateAnalysis -spec Property1b.spc tcas.i
```

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

CPAchecker 4.0 / predicateAnalysis (OpenJDK 64-Bit Server VM 17.0.13) started (CPAchecker.run, INFO)

Parsing CFA from file(s) "tcas.i" (CPAchecker.parse, INFO)

Using predicate analysis with MathSAT5 version 5.6.10 (9293adc746be) (May 31 2023 12:38:06, gmp 6.2.0, 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".
```

Property2b

```
docker run -v $(pwd):/workdir -u $UID:$GID \
  sosylab/cpachecker -predicateAnalysis -spec Property2b.spc tcas.i
```

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

CPAchecker 4.0 / predicateAnalysis (OpenJDK 64-Bit Server VM 17.0.13) started (CPAchecker.run, INFO)

Parsing CFA from file(s) "tcas.i" (CPAchecker.parse, INFO)

Using predicate analysis with MathSAT5 version 5.6.10 (9293adc746be) (May 31 2023 12:38:06, gmp 6.2.0, 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".
```

What is going on when you run CPAChecker using the commands listed?

The command asks cpachecker to perform a predicate analysis which checks whether `tcas.i` complies with or violates properties specified in `Property*.spc`. It parses the `CFA`, control flow analysis, for the provided C program. It uses `MathSAT5` and `JFactory` for boolean abstraction and predicate analysis. It iteratively refine predicate till property violation is found, or safety is verified.

What does Property1a mean?

The CPAcheker's documentation of specification automata syntax shows:

> MATCH LABEL [expr] matches if the CFA edge is a label with name, which matches regular expression expr

The `Property1a.spc` defines observation to a state transition (a CFA edge) which has label containing the string "property1a" with case-insensitive letters. Look into the source code `tcas.i`, the property specifies the code should never run into the line labeled with "property1a", case-insensitive.

Is `tcas.i` a reasonable test suite?

Yes, because it is code with non-trivial loc (2161), and there are many functions, variables and condition checking in it. The code is for traffic collisition avoidance, so property verification is desired, while checking properties by hand is tedious. It is a good use case of model checking.

What has been proved?

It is proved possible in the given source code that there exists a path leading to an undesired condition `Up_Separation >= thresh && Down_Separation < thresh` being evaluated true, where `thresh` is from `ALIM()` and `need_downward_RA` holds so that `property1a()` is called. The actual value of `thresh` and the related conditions are dependent on constant values of `Positive_RA_Alt_Thresh__*` and random values from `__VERIFIER_nondet_unsigned`.

Did you find CPAChecker to be a usable tool?

It is useful as it gives concrete counter examples when a property is verified False. It also provides results of control flow analysis and reachability graph so that bug location is made easy.

How easy is it to provide the inputs to CPAChecker?

To run the checker, it only requires a single line of code which is easy. To write the property specs, user need to have automaton in mind, and write state transitions. It may still need some efforts, especially it requires domain knowledge for the syntax. If there are off-the-shelf specs, or the spec can be written in more generic and popular language, then preparation of inputs would be even easier.

What information is present in the graphical (HTML) output?

It demonstrates detailed execution path for the counter example 1 that leads to the violation of property1a. Specifically, it assigns values to related variables e.g. `thresh=500U`, `Up_Separation=4294967284U`, `Down_Separation=88U`, etc, and shows how each branching condition is evaluated. A user can step by step go through the path, with reference to CFA, ARG, or source code listed on the right of the page.