2. 0F-2

Solution: From the problem definition: $A = X \to P(Y)$; $B = P(X \times Y)$.

Let function $f : B \to A$ be defined as for any $R \in B$, $f(R) = X \to P(Y)$ s.t. $f(R)(x) = \{y \in Y \mid (x, y) \in R\}$. So for every $x \in X$, the function is the set of all y that is paired with x in R.

(a) Prove that f is an injection (every element in B is mapped to by at most one element of A).
Let $R1, R2 \in B$ s.t. $f(R1) = f(R2)$. This means that $f(R1)(x) = f(R2)(x)$. Want to show R1 = R2.
For any $(x, y) \in X \times Y$, $(x, y) \in R1 \iff y \in f(R1)(x) \iff y \in f(R2)(x) \iff (x, y) \in R2$. Therefore, $R1 = R2$.

(b) Prove that f is a surjection (every element in B is mapped to at least one element of A).
Let function $g \in A$ that maps X to P(Y). Want to show there exists a $R \in B$ s.t. $f(R) = g$.
Let $R = \{(x, y) \in X \times Y \mid y \in g\}$. For all $x \in X$, $f(R)(x) = \{y \in Y \mid (x, y) \in R\} = \{y \in Y \mid y \in g\} = g$. Therefore, $f(R) = g$.

3. 0F-3

Solution:
Property 1a:

```
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".
```

Property 1b:

```
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".
```

Question assigned to the following page: 3

Property 2b:

```
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".
```

The CPAChecker takes in 2 arguments: the .spc file which represents the property to check, and the source file. First, it converts the source file (`tcas.i` in our case) to Control Flow Automation (CFA). Then it creates an abstraction of the program and looks for spurious counterexamples if it reaches the property's error state. Once it has analyzed the whole program, it ouputs whether the verification is true (no errors), or false with the line number of where the error was found. It also creates an output folder, which contains visualizations of the CFA and counterexample.

Property1a found a violation on line 1963, Property1b did not have any violations, and Property2b had an error on line 1997. The Property1a.spc file specifies the checker to detect "PROPERTY1A" labels as error locations. In the source code, the property1a function checks if the `need_downward_RA` predicate is true and `Up_Separation >= thresh && Down_Separation < thresh`. If true, it branches to the PROPERTY1A label and errors. `tcas.i` may not be a reasonable test suite, because it only statically checks a few properties and is a verification method rather than a comprehensive test suite. The results prove that there are some invariants that are violated in some paths and the programmer should review the source code.

I found the CPAChecker to be a useful tool, but has a steep learning curve for setup and usability. Providing inputs to it is simple, and the ouput visualizations are useful. The HTML output displays a control flow graph of the source with descriptions of function calls and the predicate values at each node. Some paths are red, which I assume show the error path.