## 2 Set Theory

1. Construct a function $f : B \to A$ as follows:

$$f(b) = (g(x) = \{p[2] | p \in b, p[1] = x\})$$

In other words, given a $b \in B$, return a function $g(x)$ which takes an $x \in X$ and returns the set consisting of the second element of all pairs $p \in b$ where the first element of $p$ satisfies $p[1] = x$.

- $g(x)$ is total since every $x \in X$ results in some set, even if it is the empty set (if $x$ does not occur in any pair contained in $b$).
- $g(x)$ is deterministic because there is only one unique way to express the sets returned by $g(x)$ given a single mapping provided by $b$.

2. $f(b)$ is a function because it is total—all valid lists of pairs can be turned into such a "lookup" function that gives sets in $Y$, and deterministic—one and only one unique function is obtained for each $b \in B$, which can be seen by inspection of the definition.

3. To prove this is a surjection (onto), show that $\forall a \in A, \exists b \in B \,.\, f(b) = a$

   (a) Choose an $a \in A$.

   (b) Create an empty set $b$

   (c) For each member $x$ of $X$:

   - Obtain $z = a(x)$. By definition, $z \in \mathcal{P}(Y)$
   - For each member $y$ of $z$:
     - Construct an ordered pair $(x, y)$ and add to $b$

   (d) In math, $b = \{(x, y) | x \in X, y \in a(x)\}$.

   (e) The resulting $b$ will be a set of pairs with some members $x \in X$ as their first element and some members $y \in Y$ as their second elements. Therefore, $b \in \mathcal{P}X \times Y$

4. To prove this is an injection (one-to-one), show that $\forall b, b' \in B \,.\, f(b) = f(b') \to b = b'$

   (a) Suppose $f(b) = f(b')$ for some $b, b' \in B$.

   (b) This means that $f(b) = g(x)$, where $g(x) \in A$ maps $X \to \mathcal{P}(Y)$ as described above, based on the set of pairs provided by $b$.

   (c) Likewise for $f(b') = g'(x)$ mapping based on pairs in $b'$.

   (d) Since $f(b) = f(b')$, $g(x)$ and $g'(x)$ are identically equal functions.

   (e) This means that $\forall x \in X$, they must map the same $x$ to the same $y \in \mathcal{P}(Y)$.

   (f) By nature of their construction, the sets of pairs defining the mappings for $g(x)$ and $g'(x)$ must be the same.

   (g) The sets that define these mappings are $b$ and $b'$. Therefore, $b = b'$

5. By showing that $f(b)$ is both onto and one-to-one, $f(b)$ is bijective. Therefore, a bijective correspondence exists between sets $A$ and $B$. □

# 3 Model Checking

## 3.1 Results

### 3.1.1 Property 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) "cpachecker/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".
```

### 3.1.2 Property 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) "cpachecker/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-1b".
Graphical representation included in the file "./output-1b/Report.html".
```

### 3.1.3 Property 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) "cpachecker/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 chosen configuration
More details about the verification run can be found in the directory "././output-2b".
Graphical representation included in the file "././output-2b/Counterexample.1.html".
```

### 3.2 Thoughts on CPAChecker

Based on my experience with the tools, I believe there is more work to be done in ascertaining the validity of CPAChecker before using it in a research project. According to the accompanying publication, CPAChecker's predicate analysis is essentially CEGAR under a different name. We have already seen the process of verifying properties of C code in this way (C→ CFA, reachability, and abstraction refinement). Specification files given via `-spec` appear to be telling CPAChecker what labels in the C code correspond to a property violation. In this sense, it appears that the C code (namely `tcas.i`) needs to be instrumented beforehand to contain the desired properties to be checked. To me, this seems a bit like the "chicken-and-egg" problem, where we must verify the code given verification conditions contained within the code itself, with the possibility of the verification conditions themselves being faulty.

There also arises the issue of actually creating "good" verification conditions that are strong enough to guarantee safety. I believe this to be a threat to validity for any verification tool, including CPAChecker. Property 1a appears to be checking if the TCAS wrongly issues a DESCEND advisory when there is not enough vertical separation below the aircraft. This seems like a useful safety property since it is usually safer for an airplane to climb, rather than descend, if it is at risk of collision from below. During verification, this property was violated because it was able to trigger this advisory despite having insufficient downward separation. In addition, it appears that the solver has found multiple paths to the counterexample. I am currently unsure of the validity of this property, as it seems to assume the two separation distances are uncorrelated and nondeterministic, which may not accurately reflect real-world dynamics, but they might be at least a reasonable over-approximation. Examining the traces proved to be challenging, but eventually, state variables at each stage of execution were found. The traceability of these states as evidence for a counterexample inspires some confidence in the solution.

The given test suite seems like a reasonable test for CPAChecker (with some caveats), as it uses several nondeterministic integer variables and has many branching paths for decisionmaking, all with different predicates. Although not all of these branches are verified with properties, the branches with such properties appear to be well-covered. Nevertheless, it would be a good idea to test this code on similar solvers, or perhaps to compare the results of those solvers to those of CPAChecker. This would determine whether the test cases are in fact valid, and therefore a good benchmark for these solvers. Additionally, it may be wise to closely examine the source code of CPAChecker to verify correct behavior (verifying the verifier, perhaps?). Therefore, although I find CPAChecker to be usable and its output promising, I also believe that more work has to be done to ensure that the conclusions drawn by CPAChecker actually say something substantial about the program it is verifying.

1 HW0

- **0 pts** Correct