## 2 Set Theory

By definition, $A$ represents all of the possible functions with domain $X$ and codomain $\mathcal{P}(Y)$. Since each function maps all elements in the domain to some element in $Y$

**Theorem 1.** First, we observe that $b \in B = \mathcal{P}(X \times Y)$ is some subset of cartesian pairs of values in $X$ and $Y$ (i.e. $b = \{..., (x_i, y_j), ...\}$).

Let $f : B \to A$ be defined as follows:

$$\forall b \in B, \quad f(b)(x) = \{y : (x, y) \in b\}$$

*Proof.* To prove that $f$ is bijective, I will show it is both surjective and injective.

**Surjective:**

To show $f$ is surjective, I demonstrate that every element in the codomain is mapped to by some element in the domain.

$\forall a \in A$ where $a : X \to \mathcal{P}(Y)$, let $b = \{(x, y) : y \in a(x)\}$

Clearly, $b \in B$ as it is some collection of cartesian pairs of $X$ and $Y$. Furthermore,

$$\begin{aligned} f(b)(x) &= \{y : (x, y) \in b\} \\ &= \{y : y \in a(x)\} \\ &= a(x) \end{aligned}$$

**Injective:**

To show $f$ is injective, I demonstrate that $f$ is one-to-one (that is, if two elements in the domain map to the same value in the codomain, then the elements cannot be different).

Define $b_1, b_2 \in B$ such that $f(b_1) = f(b_2)$

$$\begin{aligned} f(b_1) = f(b_2) &\implies f(b_1)(x) = f(b_2)(x) \quad \forall x \in X \\ &\implies \{y : (x, y) \in b_1\} = \{y : (x, y) \in b_2\} \quad \forall x \in X \\ &\implies b_1 = b_2 \end{aligned}$$

□

Question assigned to the following page: 3

# 3 Model Checking

Figure 1: Output for checking Property1a

```
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.<in
it>, 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".
```

Figure 2: Output for checking Property1b

```
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.<in
it>, 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".
```

Figure 3: Output for checking Property2b

```
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.<in
it>, 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".
```

CPAChecker was able to parse the given C code, create a state machine space that represented the different variable/line states of the program, and find certain fail states. Each of the properties represented different possible fail states. Overall, I found CPAChecker to be quite usable. With only a limited amount of playing around, I was able to see complete graphs for the code's state structure (example 2) and analyze counterexamples that broke the code (examples 1 and 3). I personally felt that the graph output by valid examples was easy to understand, whereas the counterexamples were less intuitive to parse through. Even with an example as simple as *tcas.i*, I felt that the counterexamples were convoluted/obscure.