2 OF-2

For $b \in B$ and $x \in X$, let $F : B \to A = \lambda b.\lambda x.\{y \mid (x, y) \in b\}$. We'll show F is injective and surjective, and hence is a bijection.

Now, for any $a \in A$, let $b = \{(x, y) \in X \times Y \mid y \in a(x)\}$. Then, for any $x \in X$ we have that $F(b)(x) = \{y \mid y \in a(x)\} = a(x)$. Thus by extensionality, we have that for all $a \in A$, there is a $b \in B$ such that $F(b) = a$, i.e. F is surjective.

Finally, say that for all $x \in X$ and $b_1, b_2 \in B$, we know $F(b_1)(x) = F(b_2)(x)$. Then if $(x_1, y_1) \in b_1$ we have that $y_1 \in \{y \mid (x, y) \in b_1\} = F(b_1)(x) = F(b_2)(x)$, so $(x_1, y_1) \in b_2$. Similarly, $(x_2, y_2) \in b_2$ implies that $(x_2, y_2) \in b_1$, so again by extensionality, $b_1 = b_2$. Thus F is injective, and hence is a bijection.

3 OF-3

Output for property 1a

```
./scripts/cpa.sh -predicateAnalysis -spec ../Property1a.spc ../tcas.i
Parsing CFA from file(s) "../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, g
Using refinement for predicate analysis with PredicateAbstractionRefinementStrategy strategy. (Predicat
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".
```

Output for property 1b

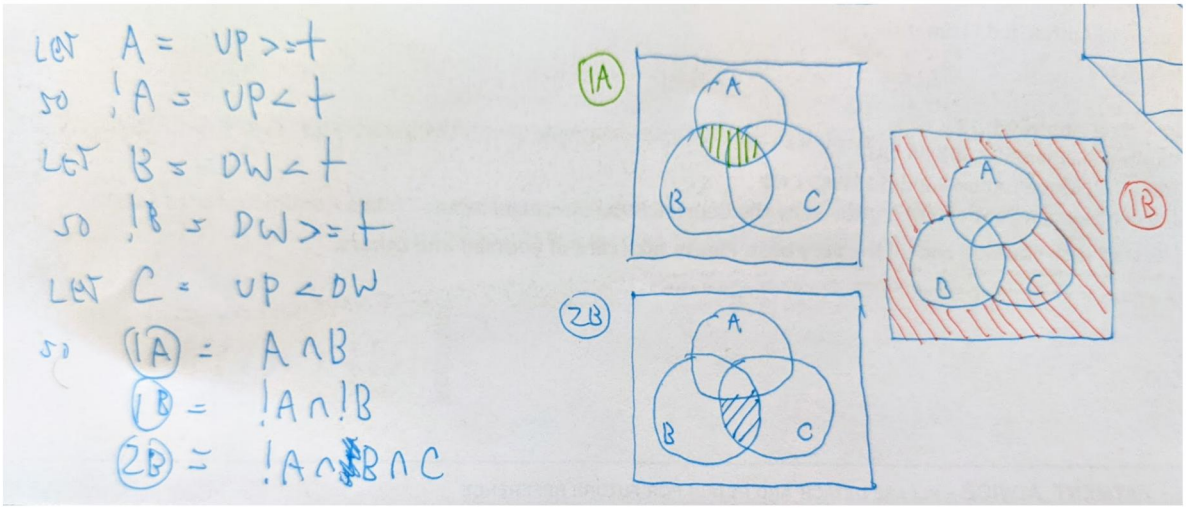
```
./scripts/cpa.sh -predicateAnalysis -spec ../Property1b.spc ../tcas.i
Parsing CFA from file(s) "../tcas.i" (CPAchecker.parse, INFO)
Using predicate analysis with MathSAT5 version 5.6.5 (63ef7602814c) (Nov  9 2020 09:01:58, gmp 6.1.2, g
Using refinement for predicate analysis with PredicateAbstractionRefinementStrategy strategy. (Predicat
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".
```

Output for property 2b

```
 ./scripts/cpa.sh -predicateAnalysis -spec ../Property2b.spc ../tcas.i
 Parsing CFA from file(s) "../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, g
Using refinement for predicate analysis with PredicateAbstractionRefinementStrategy strategy. (Predicat
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".
```

When executing these commands, CPAChecker executes (something analogous to) the SLAM/BLAST strategy for software model-checking. The provided *tcas.i* file represents a program, apparently already pre-instrumented with a specification. This specification consists of 5 properties, each given by a proposition defined on both local and global program variables. For example, Property1a asserts that we have an error if provided parameter *thresh* is between global *DownSeparation* (non-inclusive) and global *UpSeparation* (inclusive). Internally, CPAChecker abstracts the provided program into a boolean program over a set of predicates, reducing the program to a labelled transition system between states defined in terms of these predicates. It then does model checking against the boolean program, resulting in either demonstrating that none of the error conditions can occur, or it generates a path to an error in the boolean program. That path is then checked for reachability in the original program. If it's a valid path, the error is reported. Otherwise the predicate set is refined to preclude this path, and the analysis begins again.

Examining the output produced in *Counterexample.1.html*, we see that CPAChecker has found a concrete path through the program terminating in $PROPERTY1A$ violation label on line 1963 (insert JFK assassination conspiracy theory here). Specifically, we see that this run, we have an error due to proposition 1a being fulfilled; *thresh* is 500 where *UpSeparation* is 4294967284 and *DownSeparation* is 88. The *tcas.i* test suite seems somewhat suspect as a test case for the collision avoidance program. If $1b$ holds, then we know logically that both $1a$ and $2b$ must fail (see figure). So given the result for the first, the latter two aren't really telling us more about the implementation than pure propositional logic can tell us. These tests do provide weak evidence for at least the consistency of CPAChecker itself, as the results do not logically contradict each other.



I also noticed that these tests are non-deterministic; practically all globals are set nondeterministically. From what I've read this is (understandably) an issue for the basic CPAChecker. A paper I found; 'Symbolic Execution in CPAchecker' by Lemberger, seems to suggest that this has been dealt with, and the program seems to be appropriately instrumented. Test-case validity aside, I found CPAChecker to be reasonably usable. The command line interface is clean, and the output produced is easy to understand. It's easy to skim the generated CFA, tracing the counterexample path which is highlighted in red. Presumably this would make it easy to see where execution went wrong, if the program under analysis was one I knew anything about. The provided txt files clearly indicate the variable assignments along the path and at the time of failure.

1 HW0

- **0 pts** Correct