0F-2

Let f be the function that gives a mapping $a \in A$ over for some $b \in B$, such that $y \in a_x$ if and only if $(x, y) \in b$ for all $x \in X$, $y \in Y$.

An inverse function can be constructed that gives $b \in B$ for some $a \in A$ as $\{(x, y) \mid x \in X \ \wedge \ y \in Y \ \wedge \ y \in a_x\}$. This b contains (x, y) if and only if $y \in a_x$, so this function is a true inverse based on the definition of f above. Therefore, f must be a bijection.

0F-3

The program appears to be exploring the directed graph of execution between nodes representing lines of code. Opening the graphical representation shows on the left a path through this graph that serves as a counter example to the property's correctness. The CFA graph on the right shows full execution graph with the counter example path highlighted. Annotating the edges in the graph are conditions for that particular transition. For example, the first branch in the graph has two transitions with the annotations `[Alt_Layer_Value == 0]` and `[!(Alt_Layer_Value == 0)]` respectively representing the diverging execution paths resulting from the if statement on line 1864.

For Property1a.spc and Property2b.spc, the property is violated so the CPAChecker software provides a proof that the property does not hold. In this case, it does so by counter-example in the form of an execution path that arrives at the labels in question. Of note, this execution path also provides a set of pre-conditions for the input variables that would be required for that particular circumstance to occur. For Property1b.spc, the checker proves by exhaustion that no possible execution paths could arrive at the label in question.

I found the tool relatively easy to use in this case, but am concerned that applying the tool to an a real-life example would require a significant amount of overhead beyond standard programming practices to ensure that the tool is able to provide useful results. The output format I found very nice and easy to use. Once the code is prepared as necessary to be able to use the tool, it seems to me that the output would provide useful and actionable results.

In the case of Property1a.spc, the property in question is whether or not the case-insensitive label PROPERTY1a can possibly be reached. The counter example therefore shows an example of an execution path that arrives at this label.

Property1a:

```
Running CPAchecker with default heap size (1200M). Specify a larger value with -heap if you have more RAM.
Running CPAchecker with default stack size (1024k). Specify a larger value with -stack if needed.
Language C detected and set for analysis (CPAMain.detectFrontendLanguageIfNecessary, INFO)

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) "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".
```

In the case of Property1b.spc, the property in question is whether or not the case-insensitive label PROPERTY1b can possibly be reached. This property holds, so there is no counter example provided. The report still includes the CFA graph and ability to explore the execution graph of the program.

Property1b:

```
Running CPAchecker with default heap size (1200M). Specify a larger value with -heap if you have more RAM.
Running CPAchecker with default stack size (1024k). Specify a larger value with -stack if needed.
Language C detected and set for analysis (CPAMain.detectFrontendLanguageIfNecessary, INFO)

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) "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".
Graphical representation included in the file "./output/Report.html".
```

In the case of Property2b.spc, the property in question is whether or not the case-insensitive label PROPERTY2b can possibly be reached.

Property2b:

```
Running CPAchecker with default heap size (1200M). Specify a larger value with -heap if you have more RAM.
Running CPAchecker with default stack size (1024k). Specify a larger value with -stack if needed.
Language C detected and set for analysis (CPAMain.detectFrontendLanguageIfNecessary, INFO)

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) "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".
Graphical representation included in the file "./output/Counterexample.1.html".
```

1 HW0

- **0 pts** Correct