Exercise 0F-1. Bookkeeping

1. I am primarily interested in taking 590 because I had a good experience in Dr. Kamil's EECS 490 and I am attracted to PL's unique blend of theory and practice. I have also heard excellent things about Dr. Weimer and his teaching style.
2. I wish there were more opportunities to write programs that implement the PL concepts we will be reading about (either in the form of projects, homeworks, labs, etc.)
3. Despite being born without anterior cruciate ligaments in either knee, I am an avid runner and member of the Michigan Running Club.
4. What advice would you give to a recent grad who finds him/herself unhappy or unsatisfied in their first industry job? What principles should guide a job hunter when researching companies, areas of research, fields of CS, etc?
5. Woolloomooloo (an Australian City), sweettooth (technically two words.) After searching Google for the answer directly, it seems "Woolly" (double u, double o, double l) is a popular solution, despite making the original question seem a bit disingenuous.

Exercise 0F-2. Set Theory

The following function $f\colon A \to B$ satisfies the requirement where $\alpha \in A$ (i.e. $\alpha : X \to \mathcal{P}(Y)$)

$$f(\alpha) := \{\ (x, y)\ : y \in \alpha(x),\ \forall x \in X\ \}$$

First, I will prove that f is injective.

Proof. To prove injectivity, it is enough to show

$$f(a) = f(b) \implies a = b$$

for all $a, b \in A$. Immediately, $f(a) = f(b)$ implies $a(x) = b(x),\ \forall x \in X$ by the definition of f. Moreover, because a and b are both members of A they are defined to have the same domain and range. Thus, $a = b$. □

Next, I will prove that f is surjective (onto.)

Proof. To prove surjectivity, is is enough to show there exists $a\colon X \to \mathcal{P}(Y)$ s.t. $b = f(a)$ for some arbitrary $b \in B$. By the definition of B,

$$b = \{\ (x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\ \}$$

Where $0 \leq n \leq |X \times Y|$ and $(x_i, y_i) \in X \times Y$. Define a as follows:

$$a(x) := \{\ y\colon (x, y) \in b\ \}$$

for all $x \in X$. Notice $a(x)$ defines a subset of Y which satisfies the definition $a\colon X \to \mathcal{P}(Y)$ □

Having completed the proofs of injectivity and surjectivity, we conclude that f is a bijection. □

Exercise 0F-3. Model Checking

After a few hours using the command line interface and graphical components of CPA Checker 1.6.1, I am impressed by both its performance and usability. Aside from some difficulty installing Java/Docker (eventually abandoning a local installation and opting for v1.6.1 on CAEN), the tool was very easy to use and provided descriptive outputs about the successes and failures of the tests and how to generate graphical results files. Screenshots of my results for each property are included on the last page.

Per the instructions, I tested tcas.i for three separate properties (1a, 1b, 2b) using three invocations of CPA Checker. The property specifications themselves were very simple, simply defining a label in the target

program that should only be reached in the event of an error or correctness violation. In tcas.i, these property labels were located inside the bodies of if-statements testing for invalid program states (e.g. Up_Separation $\geq$ thresh && Down_Separaton $<$ thresh.) All things considered, I think tcas.i does an adequate job of testing the basic functionality of CPA Checker (i.e. reachability given basic if-statement control flow and a few direct function calls) but is too short and simplistic to be an exhaustive test suite. I would expect a better test suite to be longer, include loops, struct accesses, pointers, indirect function calls, etc. Moreover, the property specifications were incredibly short (especially given the sheer complexity of the CPA Checker specification grammar: doc/SpecificationAutomata.md.)

In general, my experience with CPA Checker was positive. Upon executing each of the three sample commands on the EECS 590 website, CPA Checker generated an abstract boolean representation of tcas.i and tested whether or not there exists a feasible program state that reaches the specified property label. If so, CPA Checker provides a detailed graphical counterexample that illustrates the exact state and control flow path that led to the error. My testing resulted in TRUE, TRUE, FALSE for properties 1a, 1b, and 2b respectively. This proves that there is a reachable error state violating property2b, but not so for properties 1a and 1b (although property 1a may be a false positive given the Piazza discussion on CPA Checker 2.0.) The graphical results (and counterexample generated as a result of the Property2b failure) were especially impressive. As discussed in class and in the reading, a major barrier to the adoption of model checking software is whether or not such software can produce meaningful and easily-understood bug reports. Judging by the interactive HTML pages and automatically color-coded CFGs, I would argue that CPA Checker has met that goal.

```
owebb@CAEN > CPAchecker-1.6.1-unix/scripts/cpa.sh -predicateAnalysis -spec Property1a.spc tcas.i
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.
Using the following resource limits: CPU-time limit of 900s (ResourceLimitChecker.fromConfiguration, INFO)

CPAchecker 1.6.1 (OpenJDK 64-Bit Server VM 1.8.0_282) started (CPAchecker.run, INFO)

Using predicate analysis with SMTInterpol 2.1-238-g1f06d6a-comp and JFactory 1.21. (PredicateCPA:PredicateCPA.<init>, INFO)

No invariants are computed (PredicateCPA:InvariantsManager.<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".
Run /afs/umich.edu/user/o/w/owebb/eecs590/CPAchecker-1.6.1-unix/scripts/report-generator.py to show graphical report.
owebb@CAEN >
```

Figure 1: Property 1a Output

```
owebb@CAEN > CPAchecker-1.6.1-unix/scripts/cpa.sh -predicateAnalysis -spec Property1b.spc tcas.i
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.
Using the following resource limits: CPU-time limit of 900s (ResourceLimitChecker.fromConfiguration, INFO)

CPAchecker 1.6.1 (OpenJDK 64-Bit Server VM 1.8.0_282) started (CPAchecker.run, INFO)

Using predicate analysis with SMTInterpol 2.1-238-g1f06d6a-comp and JFactory 1.21. (PredicateCPA:PredicateCPA.<init>, INFO)

No invariants are computed (PredicateCPA:InvariantsManager.<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".
Run /afs/umich.edu/user/o/w/owebb/eecs590/CPAchecker-1.6.1-unix/scripts/report-generator.py to show graphical report.
owebb@CAEN >
```

Figure 2: Property 1b Output

```
owebb@CAEN > CPAchecker-1.6.1-unix/scripts/cpa.sh -predicateAnalysis -spec Property2b.spc tcas.i
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.
Using the following resource limits: CPU-time limit of 900s (ResourceLimitChecker.fromConfiguration, INFO)

CPAchecker 1.6.1 (OpenJDK 64-Bit Server VM 1.8.0_282) started (CPAchecker.run, INFO)

Using predicate analysis with SMTInterpol 2.1-238-g1f06d6a-comp and JFactory 1.21. (PredicateCPA:PredicateCPA.<init>, INFO)

No invariants are computed (PredicateCPA:InvariantsManager.<init>, INFO)

Using refinement for predicate analysis with PredicateAbstractionRefinementStrategy strategy. (PredicateCPA:PredicateCPARefiner.<init>, INFO)

Starting analysis ... (CPAchecker.runAlgorithm, INFO)

Error path found, starting counterexample check with CPACHECKER. (CounterexampleCheckAlgorithm.checkCounterexample, INFO)

Using the following resource limits: CPU-time limit of 900s (CounterexampleCheck:ResourceLimitChecker.fromConfiguration, INFO)

Repeated loading of Eclipse source parser (CounterexampleCheck:EclipseParsers.getClassLoader, INFO)

Error path found and confirmed by counterexample check with CPACHECKER. (CounterexampleCheckAlgorithm.checkCounterexample, INFO)

Stopping analysis ... (CPAchecker.runAlgorithm, INFO)

Verification result: FALSE. Property violation (error label in tcas.i, line 1997) found by chosen configuration.
More details about the verification run can be found in the directory "./output".
Run /afs/umich.edu/user/o/w/owebb/eecs590/CPAchecker-1.6.1-unix/scripts/report-generator.py to show graphical report.
owebb@CAEN >
```

Figure 3: Property 2b Output

Peer Review ID: 61859269 — enter this when you fill out your peer evaluation via gradescope

1 HW0

- **0 pts** Correct