Exercise 0F-2. Set Theory

We begin the proof with some true statements:

Let X and Y be sets.

Let A and B be sets s.t. $A = X \to \mathcal{P}(Y)$ and $B = \mathcal{P}(X \times Y)$

We know that for any two sets R and M, the cartesian product is:

$R \times M = \{(x, y) | x \in X, y \in Y\}$

Thus we can rewrite B as: $B = \mathcal{P}(\{(x, y) | x \in X, y \in Y\})$

Therefore a member of the set B is a set of tuple(s) of type (x,y).

One way of describing a tuple such as (x, y) is a function such as f(x) = y. We will use this immediately in our proof.

According to the (optional) course text, any function which can be shown to have a valid inverse is 1-1. I assume rather than prove this as it is trivial to see that any invertible function must have a 1-1 mapping otherwise the inverse would not be a function.

Let the proof proceed by example, proving two arbitrary functions of type $f : A \to B$ and $g : B \to A$ exist and they are inverses:

Elements of A are mapped to elements of B which we know to be of the form (x, y) via some function y = a(x).

Let f be a function s.t. $f(a) = \{(x, y) | y \in a(x)\}$

Elements of B, (x, y) must be mapped to single elements of A. To do this we accept tuples of type (x,y) as b, and return the y element under the assumption we are dealing with a bijection.

Let g be a function s.t. $g((x, y)) = y = g(b)$ where y is a set of function(s) because the type of g is a set of functions, so $g((x, y)) = y(x) = g(b)$.

We now compose these functions and prove that g is the inverse of f.

$$g(f(a)) = a = g(\{(x, y) | y \in a(x)\})$$

$$a = g((x, a(x)) = a(x) = a$$

Because $a \in A$ and $b \in B$ are some arbitrary members of A and B we have shown:

$$\forall a \in A, b \in B, \quad (f \circ g(a) = a)$$

By showing that $f \circ g(a) = a$ as constructed, we have proven a bijection $f : B \to A$.

Exercise 0F-3. Model Checking

Property1a CPAChecker Output

```
Language C detected and set for analysis (CPAMain.detectFrontendLanguageIfNecessary, INFO)

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

CPAchecker 2.0.1-svn / 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".
```

Property1b CPAChecker Output

```
Language C detected and set for analysis (CPAMain.detectFrontendLanguageIfNecessary, INFO)
```

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

CPAchecker 2.0.1-svn / 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".
```

Property2b CPAChecker Output

```
Language C detected and set for analysis (CPAMain.detectFrontendLanguageIfNecessary, INFO)

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

CPAchecker 2.0.1-svn / 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".
```

How CPAChecker Model Checks

CPAChecker appears to be model checking using some older methods. According to the literature they use MathSAT to do SMT solving, and JavaBDD. BDD's or binary decision diagrams are an interesting, but somewhat outdated method of model checking as they can be exponential in some cases, and even exponential in the best case under some functions [1]. The SMT solver MathSAT is a robust tool that historically focused on interpolation. While interpolation continues to be used in some model checking applications it has become outpaced by IC3/PDR style model checking tools [2]. CPAChecker is using interpolation to over-approximate the reachable set of states and checking if the property violation in reachable. If the over-approximation contains the violation but the actual reachable set does not, the abstraction is refined. As always, this assumes correctness of the underlying solver as well as the CPAChecker code, which may not always be a safe assumption.

My Thoughts and Experiences Using CPAChecker

Property1a.spec simply matches the location Property1a (being non-case sensative) and checks if the error state is a successor. The Property1a description in **tcas.i** describes an error occurring when the thresh-hold for up separation is exceeded and the thresh-hold for down separation is not exceeded. We can look at line **2118** for the statement "if (need_downward_RA)" and see that properties 1a - 5a are all concerned with a downward readjustment. To make sense of Property1a, assume our craft needs to readjust downward. We check if it's the case that our thresh-hold for downward separation is really not met, if the thresh-hold is met, our command to move down must cause an unsafe path and is therefore an error. Additionally, we check that we really need to move downwards, if the thresh-hold for upward separation is actually not met, then the downward adjustment may be erroneous. We check the condition is $P \rightarrow \neg[Up\_Separation \geq thresh \wedge Down\_Separation < thresh]$. **tcas.i** is a strong test suite as it is a "real world" C application which has life or death consequences. Further because some properties hold and others do not hold we can test the ability of CPAChecker to find bugs or provide proofs that the model is sound. The properties 1a and 2b are shown above to be untrue. This implies that the system may reach error states depending on the inputs (up/down separation), meaning the code is unsafe.

Because we found errors, it's possible to retrace our steps to find out what led to the unsafe execution. CPAChecker supplies a graphical error trace but I found the **Counterexample.1.assignment.txt** file to be the most helpful for the property1a violation. We can see in this file that the Up_Separation is set to approximately 2^{32}, thresh is set to 500, and Down_Separation is set to 88. Once we enter Property1a with those conditions it clearly errors out. The CPAChecker allows us to reason quickly over a specific property and see if it holds globally across all valid inputs across around two thousand lines of code. The tool allows a student like me to verify (or in our case falsify) the code in a matter of seconds. We as engineers build systems of increasing power and complexity, and as we do so we need to increase our ability to automate verification to keep up. The same way a civil engineer owes a promise that a structure will be safe, software engineers working in sensitive areas must verify that the code they've written is sound.

References

[Bra11] Aaron R. Bradley. "SAT-Based Model Checking without Unrolling". In: *Verification, Model Checking, and Abstract Interpretation*. Ed. by Ranjit Jhala and David Schmidt. Vol. 6538. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 70–87. isbn: 978-3-642-18274-7 978-3-642-18275-4. doi: `10.1007/978-3-642-18275-4_7`. url: `http://link.springer.com/10.1007/978-3-642-18275-4_7` (visited on 01/30/2021).

[EMB11] Niklas Een, Alan Mishchenko, and Robert Brayton. “Efficient Implementation of Property Directed Reachability”. In: (2011).

1 HW0

- 0 pts Correct

Peer Review ID: 62042345 — enter this when you fill out your peer evaluation via gradescope