Exercise 0F-2: Set Theory:

SET THEORY

0F-2 Defining the function: $f: B \to A$

for any relation $R \in B$ (where $R \subseteq X \times Y$):

$f(R)$ is the function that maps each $x \in X$ to the set $\{y \in Y \mid (x,y) \in R\}$

* To Prove: Injective:

Assume, $f(R_1) = f(R_2)$ for some $R_1, R_2 \in B$.

Thus, $\forall x \in X \; f(R_1)(x) = f(R_2)(x)$

Thus by the definition of f, this implies that the set of y values associated with x in R_1 is same as the set of y values associated with x in R_2 for all $x \in X$.

Thus, R_1 and R_2 must contain the same ordered pairs,

Thus, $R_1 = R_2$ (Hence f is injective)(one to one)

* To Prove: Surjective

let $g \in A$ be an arbitrary function from X to $P(Y)$.

Thus, $R = \{(x,y) \mid x \in X \text{ and } y \in g(x)\}$

Hence, by the definition of f,

$$f(R)(x) = \{y \in Y \mid (x,y) \in R\} = \{y \in Y \mid y \in g(x)\} = g(x) \text{ for all } x \in X.$$

This shows that for any function $g \in A$, there exists a relation $R \in B$ such that $f(R) = g$.

Hence f is surjective. (onto)

Hence injective and sujective, means it is bijection!

Exercise 0F-3: Model Checking

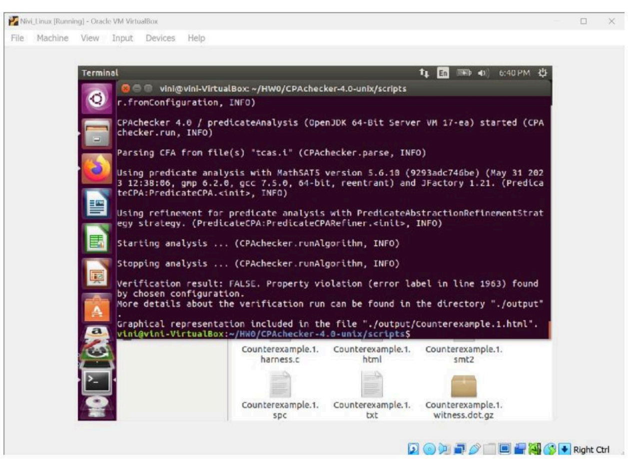
My experience with CPAChecker was both challenging and insightful. When running the commands, the tool uses predicate abstraction to analyze the program tcas.i against the specified properties, like Property1a. These properties define safety conditions that the program needs to meet, such as avoiding errors or unsafe states. The program tcas.i is a simple model of a Traffic Collision Avoidance System, which works well as an example but is very specific and can be hard to understand without extra documentation. The tool either proves that the property holds or shows a counterexample if it doesn't.

Using CPAChecker felt powerful but not very beginner-friendly. Setting it up required carefully organizing property files and the program code, and running commands meant understanding their exact syntax. The HTML output provided detailed results, including error traces and counterexamples, but interpreting this information required some familiarity with the tool and the code being analyzed, which made the learning curve steep for new users.

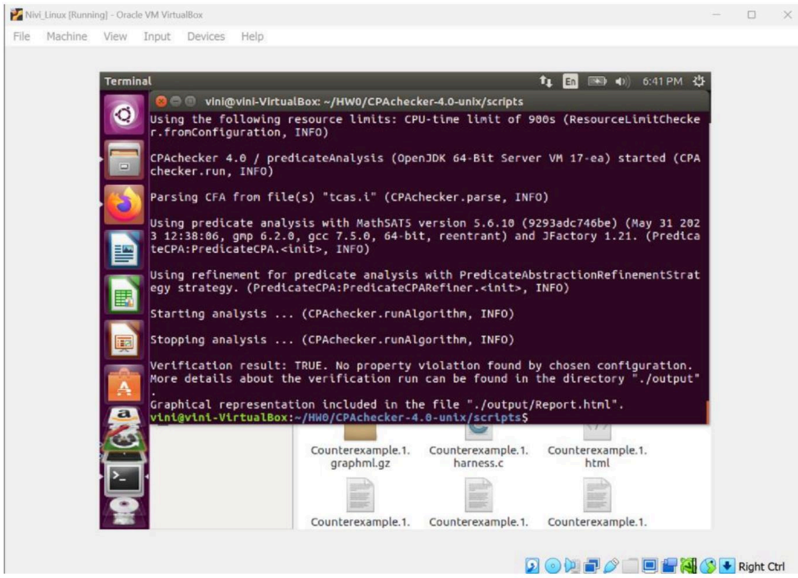
From an experimental point of view, the results depend on having correctly defined properties and a valid program. Even small mistakes in these inputs could lead to incorrect conclusions. While CPAChecker gives reliable verification for the specified properties, it struggles with larger or more complex programs. This experience showed me the strengths of tools like CPAChecker in verifying properties but also highlighted their limitations, especially when applied to real-world scenarios or more extensive systems.

Commands and their Screenshots:

1. **/path/to/CPAChecker/scripts/cpa.sh -predicateAnalysis -spec Property1a.spc tcas.i**



2. **/path/to/CPAChecker/scripts/cpa.sh -predicateAnalysis -spec Property1b.spc tcas.i**



3. **/path/to/CPAChecker/scripts/cpa.sh -predicateAnalysis -spec Property2b.spc tcas.i**