Exercise 0F-2. Set Theory [5 points].

- First we write out the elements $b_{i,j}$ which belongs to set $B_i \in B$

$$\because\ B = \mathcal{P}(\{(x, y) \mid x \in X, y \in Y\})$$

$$let\ b_{i,j} = (x_{i,j}, y_{i,j}),\ where\ x_{i,j} \in X, y_{i,j} \in Y$$

- Next compose a mapping from B to A.

$$\forall B_i \in B, f(B_i) = g_i : \{X_i \rightarrow Y_i\}$$

$$where\ x_{i,j} \mapsto \{y_{i,k} \mid (x_{i,k}, y_{i,k}) \in B_i, x_{i,k} = x_{i,j}\}$$

- Now let's prove $g_i \in A : X \rightarrow \mathcal{P}(Y)$

  By definition $X_i = \bigcup_j x_{i,j}$ where $x_{i,j} \in X$, thus $X_i \subseteq X$

  $Y_i = \bigcup_j \{y_{i,k} \mid (x_{i,k}, y_{i,k}) \in B_i, x_{i,k} = x_{i,j}\}$ where $y_{i,k} \in Y$, thus $Y_i \subseteq \mathcal{P}(Y)$

  Having both $X_i \subseteq X$ and $Y_i \subseteq \mathcal{P}(Y)$, we proved g is also $X \rightarrow \mathcal{P}(Y)$

- Now we show the injection of function f

  Suppose set $P \neq Q, P \in B, Q \in B$; $\exists b_k = (x_k, y_k)$ where $b_k \in P$, $b_k \notin Q$

  Let $f(P) = g_p$, $f(Q) = g_q$

  Since $(x_k, y_k) \in P$, $y_k \in g_p(x_k)$; while $(x_k, y_k) \notin Q$, thus $y_k \notin g_q(x_k)$

  Thus $g_p \neq g_q$, the injection is proved

- Now we prove subjection of function f

  For any function $h \in A$, let's write out its mapping:

  $x_0 \mapsto \{y_{0,0}, y_{0,1}, \ldots, y_{0,n_0}\}$

  $\vdots$

  $x_m \mapsto \{y_{m,0}, y_{m,1}, \ldots, y_{m,n_m}\}$

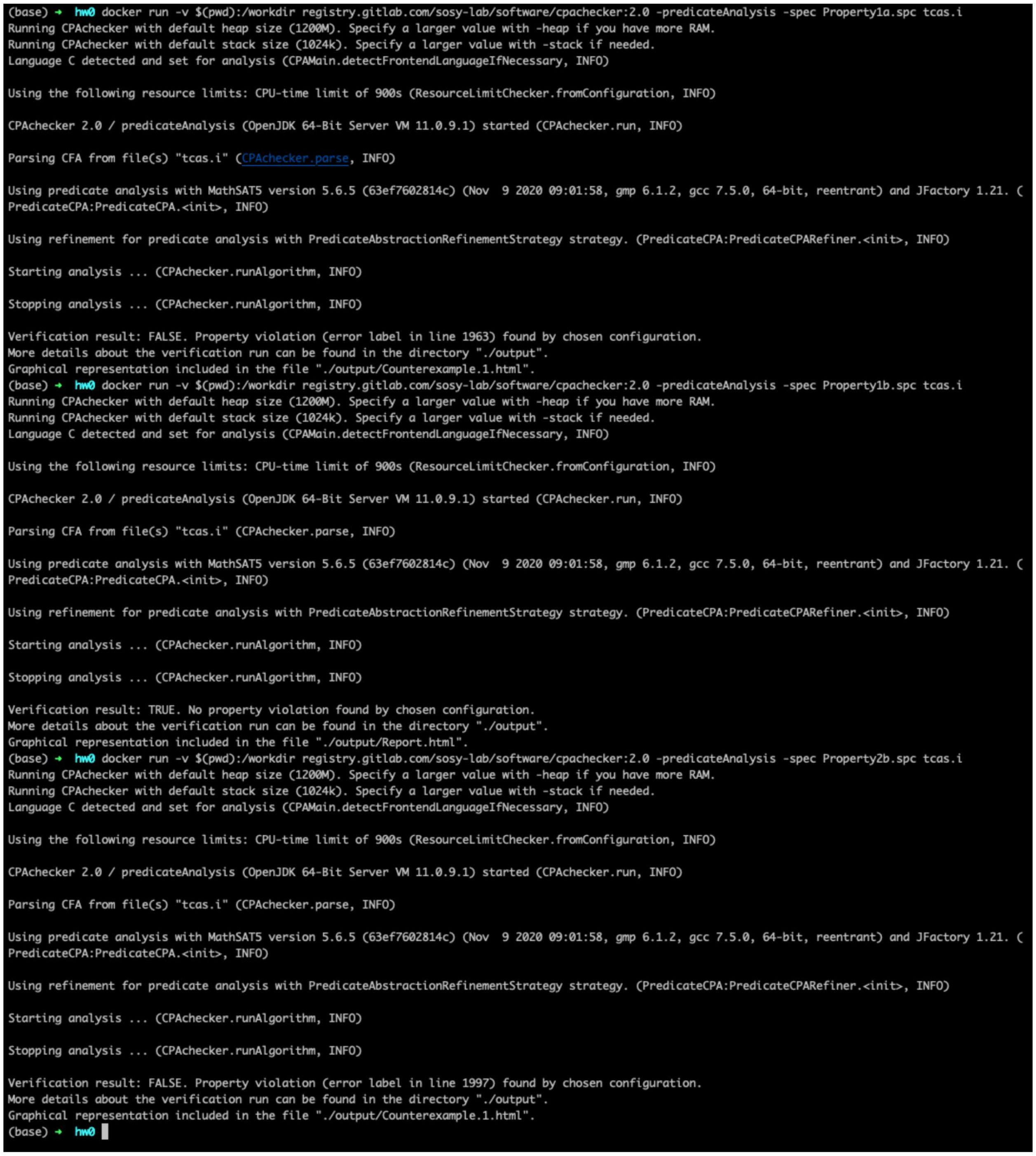
  Construct set $P = \bigcup_{i=0}^{m} \{(x_i, y_{i,0}), (x_i, y_{i,1}), \ldots, (x_i, y_{i,n_i})\}$

  It is trivial to check that $f(P) = h$ and $P \subseteq B$, thus we have proved subjection

  By combining injection and subjection f is a bijection between sets A and B

Exercise 0F-3. Model Checking [10 points].

- Outputs



```
(base) ➜  hw0 docker run -v $(pwd):/workdir registry.gitlab.com/sosy-lab/software/cpachecker:2.0 -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.
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".
(base) ➜  hw0 docker run -v $(pwd):/workdir registry.gitlab.com/sosy-lab/software/cpachecker:2.0 -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.
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".
(base) ➜  hw0 docker run -v $(pwd):/workdir registry.gitlab.com/sosy-lab/software/cpachecker:2.0 -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.
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".
(base) ➜  hw0 ▌
```

- My thoughts

Property1a.spec means any reachable labels in the program that matches $[Pp\backslash][Rr\backslash][Oo\backslash][Pp\backslash][Ee\backslash][Rr\backslash][Tt\backslash][Yy\backslash][1\backslash][Aa\backslash]]$ means the program is in *USEFIRST* state and should print out a error message of "error label in [?] location"

In the case of *tcas.i*, it is a reasonable proof-of-concept test case. On one side, it has a reasonable complex branches that shows CPAChecher is able to distinguish reachable and non-reachable branches; on the other side, it is not too complicated so that we can manually check the result. In the specific case of *Property1a.spec*, CPAChecker proves *PROPERTY1A* is reachable at line 1963, thus print out the corresponding message. But *tcas.i* can definitely be extended with more complicated branches along with more dynamic variables, to show more power from CPAChecker.

From my experience, CPAChecker is definitely useful. I can conveniently check whether an undesired state can be reached and what is the exact execution path. The spec I need to write is straight forward and easy to write. The command-line output message is clear and concise. But it would be better to specify which property it violates. In the graph report it show detailed execution path in both graphical way and textual way. The graphical CFA facilitates the understanding of the control flow, and the "path section" makes it easier to track the critical variables. The report is super awesome.

1 HW0

- **0 pts** Correct