Questions assigned to the following page: 1 and 2

Peer Review ID: 300514266 — enter this when you fill out your peer evaluation via gradescope

Response: `[a-Z]*([aA][aA]|[bB][bB]|[cC][cC]|[dD][dD]|[eE][eE]|[fF][fF]|[gG][gG]|[hH][hH]|[iI][iI]|[jJ][jJ]|[kK][kK]|[lL][lL]|[mM][mM]|[nN][nN]|[oO][oO]|[pP][pP]|[qQ][qQ]|[rR][rR]|[sS][sS]|[tT][tT]|[uU][uU]|[vV][vV]|[wW][wW]|[xX][xX]|[yY][yY]|[zZ][zZ]){3}[a-Z]*`

Note: This avoids using constructs such as \n (backreferences) because they introduce memory and are thus non-regular.

Extended regular expressions are the same as deterministic finite automata (DFA). Any regex can be converted to a DFA and vice versa. This equivalence is guaranteed by Kleene's theorem, which states that both regular expressions and DFAs are valid ways to define and model regular languages.

Exercise 0F-2. Set Theory [5 points]. This answer should appear after the first page of your submission and may be shared during class peer review.

This exercise is meant to help you refresh your knowledge of set theory and functions. Let X and Y be sets. Let $\mathcal{P}(X)$ denote the powerset of X (the set of all subsets of X). There is a 1-1 correspondence (i.e., a bijection) between the sets A and B, where $A = X \to \mathcal{P}(Y)$ and $B = \mathcal{P}(X \times Y)$. Note that A is a set of functions and B is a (or can be viewed as a) set of relations. This correspondence will allow us to use functional notation for certain sets in class. This is Exercise 1.4 from page 8 of the Winskel textbook.

Demonstrate the correspondence between A and B by presenting an appropriate function and proving that it is a bijection. For example, you might construct a function $f : B \to A$ and prove that f is an injection and a surjection.

Response: We attempt to construct a bijective function between the sets A and B as follows:

It is given that $A = X \to \mathcal{P}(Y)$

It is given that $B = \mathcal{P}(X \times Y)$

Let us define the function $f : B \to A$ s.t. for any $r \in B$ (i.e. $r \subseteq (X \times Y)$), $f(r)$ maps each $x \in X$ to the set $\{y \in Y \mid (x, y) \in r\}$

- By definition, for every $r \in B$, $f(r)$ maps each $x \in X$ to a subset of Y

Thus $f(R)$ is indeed a valid function from X to $P(Y)$

- Assume we have $r_1, r_2 \in B$ and $r_1 \neq r_2$
- Without loss of generality, $\exists (x, y) \in r_1$ where $(x, y) \notin r_2$
- Then $y \in f(r_1),\ y \notin f(r_2)$

Therefore $r_1 \neq r_2 \to f(r_1) \neq f(r_2)$, by this contrapositive example, f is an **injection**

- Take some $g \in A$. We show that $\exists r \in B$, s.t. $f(r) = g$
- Define $r_g \subseteq (X \times Y)$ s.t. $r_g := \{(x, y) \in X \times Y \mid y \in g(x)\}$

Questions assigned to the following page: 2 and 3

Peer Review ID: 300514266 — enter this when you fill out your peer evaluation via gradescope

- It follows that since $r_g \subseteq (X \times Y)$, $r_g \in B$
- $\forall x \in X,\ f(r)(x) = \{y \in Y \mid (x, y) \in r\}$
- $= \{y \in Y \mid y \in g(x)\}$
- $= g(x)$

Therefore $\forall g \in A, \exists r \in B$ s.t. $f(r) = g$, and thus f is an **surjection**

We've shown that f is both an **injection** and **surjection**, thus proving f is a **bijection**, demonstrating the 1-1 correspondence between sets A and B

Exercise 0F-3. Model Checking [10 points]. This answer should appear after the first page of your submission and may be shared during class peer review.

Download the CPAChecker software model-checking tool using the instructions on the homework webpage. Read through enough of the manual to run the tool on the `tcas.i` testcase provided on the homework webpage. Check the three properties given. For each command, copy or screenshot the last ten non-empty lines of output from CPAChecker and include them as part of your answer to this question.

In this graduate-level class, it is your responsibility to get CPAChecker up and running properly. This may require you to set up a virtual machine. This level of systems programming experience is a prerequisite for the class and is intentionally part of an early assignment (i.e., before the drop deadline) to help students determine if they have the right incoming preparation.

Hint: if your output when checking `Property1a` does not indicate something like "Verification result: FALSE. Property violation (error label in line 1963) found by chosen configuration." then you may not have set things up correctly.

In at most three paragraphs, summarize your experience with the CPAChecker tool.

- What is going on when you run CPAChecker using the commands listed? What does `Property1a` mean? Is `tcas.i` a reasonable test suite? What has been proved? (This is the heart of the question. You may have to read ugly code, understand a legacy tool, and apply concepts from class. It is expected that answering this well will require time.)
- Did you find CPAChecker to be a usable tool? How easy is it to provide the inputs to CPAChecker? What information is present in the graphical (HTML) output?

For full credit, do not restate the lecture on counter-example guided abstraction refinement; instead, discuss your thoughts and experience using this tool (including its input requirements, output guarantees, and context). Focus on threats to validity (e.g., imagine that you were writing a paper and using this as an experiment) over usability.

Both your ideas and also the clarity with which they are expressed (i.e., your English prose) matter. You can use a tool like ChatGPT to refine your prose, but be wary of false claims. A reader should be able to identify your main points, the arguments you are making, and your conclusion.

Peer Review ID: 300514266 — enter this when you fill out your peer evaluation via gradescope

Question assigned to the following page: 3

Response: When using the commands listed to run CPAChecker, we initiate Predicate Analysis with the -spec flag pointing to a .spc file (which contains the automaton specification) and the .i file, a preprocessed C program with expanded macros and includes. CPAChecker exhaustively checks all paths of a boolean abstraction of the C program. The automaton embedded in the specification is activated to verify whether the defined properties (like PROPERTY1a) are reached in the abstracted program. The tool performs symbolic execution, analyzing all possible paths in the abstracted program. If an error is detected, the model checker provides a counterexample, indicating the point of violation, and performs counterexample refinement as discussed in lecture (not repeating here to avoid repeating lecture).

For Property1a, CPAChecker identifies execution paths that lead to the PROPERTY1A label, confirming the program's potential to reach this state. This occurs when the alt_sep_test() function calls property1a, passing alim as a variable. The error(); line in the program is not to be confused with what CPAChecker is actually searching for. While it's true this line would trigger an error at runtime, CPAChecker's focus comes solely via the automaton, which identifies the violation by reaching the PROPERTY1A label based on the abstracted execution path. This process showcases how CPAChecker ensures that the program maintains certain safety properties, and its automated verification mechanism can detect errors that may be overlooked in traditional testing.

In terms of usability, CPAChecker offers a powerful tool for formal verification, however it requires a solid foundation of how the tool works and how its system is setup. While gathering inputs such as the preprocessed C program can be relatively straightforward, setup challenges can still arise. Firstly, creating the specification itself often requires notable human time and effort (Ball & Rajamani 2002), and while projects like SLAM aim to alleviate this burden, valid concerns still arise in the accuracy of such specs (the specifications themselves may have inaccuracies/bugs/faults) and how wholistic the specifications are (do they cover all potential errors that the program may result in?). Thus, it remains that CPAChecker suffers from many of the potential pitfalls present by such a human-in-the-loop process. Secondly, this is a relatively small program, with only around 2000 lines of code. The scalability of predicate analysis tools, such as CPAChecker, are bottlenecked by their ability to efficiently convert source programs into boolean abstractions (Ball & Rajamani 2002), thus as programs scale in size, the computational power required by the CPAChecker scales exponentially. A third critique to note, is that while the tool offers a robust graphical display of the execution path taken to reach an error state, interpretability of CPAChecker's output is often limited to those with adequate knowledge and understanding of the tool. While the CPAChecker does offer the line number of where an error label is reached (if one was), this information risks being made obsolete as the complexity of a bug/error path increases in magnitude. UI design could help ease this debugging process, as well as external programs that take as input CPAChecker's output and convert it into something more useful, descriptive, and human-readable.

Question assigned to the following page: 3

Outputs (Last 10 Lines):

Property1a

```
Using the following resource limits:  CPU-time limit of 900s
(ResourceLimitChecker.fromConfiguration, INFO) CPAchecker 4.0 /
predicateAnalysis (OpenJDK 64-Bit Server VM 17.0.13) started
(CPAchecker.run, INFO) Parsing CFA from file(s) "tcas.i" (CPAchecker.parse,
INFO) Using predicate analysis with MathSAT5 version 5.6.10 (9293adc746be)
(May 31 2023 12:38:06, gmp 6.2.0, 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

```
Using the following resource limits:  CPU-time limit of 900s
(ResourceLimitChecker.fromConfiguration, INFO) CPAchecker 4.0 /
predicateAnalysis (OpenJDK 64-Bit Server VM 17.0.13) started
(CPAchecker.run, INFO) Parsing CFA from file(s) "tcas.i" (CPAchecker.parse,
INFO) Using predicate analysis with MathSAT5 version 5.6.10 (9293adc746be)
(May 31 2023 12:38:06, gmp 6.2.0, 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".
```

Property2a

```
Using the following resource limits:  CPU-time limit of 900s
(ResourceLimitChecker.fromConfiguration, INFO) CPAchecker 4.0 /
predicateAnalysis (OpenJDK 64-Bit Server VM 17.0.13) started
(CPAchecker.run, INFO) Parsing CFA from file(s) "tcas.i" (CPAchecker.parse,
INFO) Using predicate analysis with MathSAT5 version 5.6.10 (9293adc746be)
(May 31 2023 12:38:06, gmp 6.2.0, gcc 7.5.0, 64-bit, reentrant) and JFactory
1.21.  (PredicateCPA:PredicateCPA.<init>, INFO) Using refinement for
```

Question assigned to the following page: 3

```
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".
```