CPSC 406

Tyler Lewis
Chapman University

May 11, 2023

Abstract

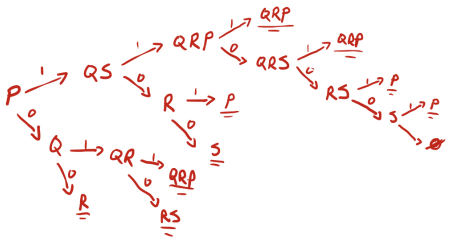
A very short introduction to typesetting in LaTeX for my courses "Programming Languages", "Compiler Construction" and "Algorithm Analysis".

Contents

1 Homework

1.1 HW 1

NFA2DFA In order to convert the provided NFA to DFA I considered each possible combination of P, Q, R, and S, and considered each possible combination its own state. The included figure details every possible state the NFA/DFA may find itself in.



State	0	1
P	Q	QS
R	S	P
Q	R	QR
S	∅	P
QS	R	QRP
QR	RS	QRP
RS	S	P
QRP	QRS	QRP
QRS	RS	QRP

1.2 HW 2

Question 1:

1. $f(X, f(X,Y)) \stackrel{?}{=} f(f(Y,a), f(U,b))$

$X \stackrel{?}{=} f(Y,a)$ $\quad f(X,Y) \stackrel{?}{=} f(U,b)$

$\sigma_1 = \frac{f(Y,a)}{X}$ $\quad X = U$ $\quad Y = b$

$\sigma_2 = \frac{U}{X}$ $\quad \sigma_3 = \frac{b}{Y}$

$\sigma = \left[\frac{f(Y,a)}{X}, \frac{U}{X}, \frac{b}{U}\right]$

2. $f(g(U), f(X,Y)) \stackrel{?}{=} f(X, f(Y,U))$

$g(U) \stackrel{?}{=} X$ $\quad f(X,Y) \stackrel{?}{=} f(Y,U)$

$X \stackrel{?}{=} Y$ $\quad Y \stackrel{?}{=} U$

$X \stackrel{?}{=} U$

$\sigma = \frac{g(X)}{X}$ Fail

3. $h(U, f(g(V), W), g(W)) \stackrel{?}{=} h(f(X,b), U, Z)$

$U \stackrel{?}{=} f(X,b)$ $\quad f(g(V), W) \stackrel{?}{=} U$ $\quad g(W) \stackrel{?}{=} Z$

$f(g(V), W) \stackrel{?}{=} f(X,b)$ $\quad \sigma_3 = \frac{g(W)}{Z}$

$g(V) \stackrel{?}{=} X$ $\quad W \stackrel{?}{=} b$

$\sigma_1 = \frac{g(V)}{X}$ $\quad \sigma_2 = \frac{b}{W}$

$\sigma = \sigma_1 \circ \sigma_2 \circ \sigma_3 = \left[\frac{g(V)}{X}, \frac{b}{W}, \frac{g(W)}{Z}\right]$

Question 2:

?- conn(W,a), conn(a,W)

?- addr(W,a), addr(a,Z), serv(Z), addr(Z,W) `?- twoway(W,a)`

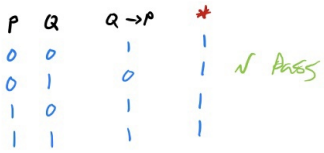
```
?- conn(W,a), conn(a,W)
?- addr(W,a), addr(a,Z), serv(Z), addr(Z,W)
```

1.3 HW 6

1. $P \vee \neg P$

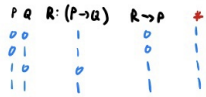
2. $(P \to Q) \to (\neg Q \to \neg P)$

3. $P \to (Q \to P)$



4. $(P \to Q) \vee (Q \to P)$

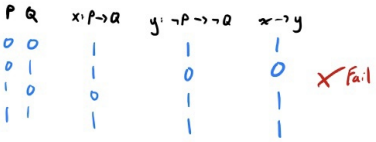
5. $((P \to Q) \to P) \to P$



6. $(P \vee Q) \wedge (\neg P \vee R) \to Q \vee R$

7. $(P \vee Q) \to (P \wedge Q)$

8. $(P \to Q) \to (\neg P \to \neg Q)$



1.4 HW 9

HackMD Page

Exercise 1:

- ‘success‘: This is a propositional variable that is true if both statusA and statusB are ok. This variable is used to check whether the protocol was executed successfully.
- ‘aliceBob‘: This is a propositional variable that is true if ‘partnerA‘ is ‘bob‘. This variable is used to check whether Alice successfully communicated with Bob.
- ‘bobAlice‘: This is a propositional variable that is true if ‘partnerB‘ is ‘alice‘. This variable is used to check whether Bob successfully communicated with Alice.
- ‘&&‘: This is the logical AND operator. It is used to combine two boolean expressions and evaluate to true if both expressions are true.
- ‘→‘: This is the implication operator. It is used to specify a condition that must be satisfied in order for an action to occur. For example, in the statement ‘(data.key == keyA) && (data.d1 == nonceA) →‘, the action on the right-hand side can only occur if the conditions on the left-hand side are true.
- ‘[]‘: This is the "always" operator. It is used to specify a condition that must always be true in order for a property to hold. For example, the property $[](success \rightarrow (aliceBob \wedge bobAlice))$ specifies that if the protocol was executed successfully, then Alice must have communicated with Bob and Bob must have communicated with Alice.
- ‘[]A‘: This is the "always eventually" operator. It is used to specify a condition that must eventually be true in order for a property to hold.

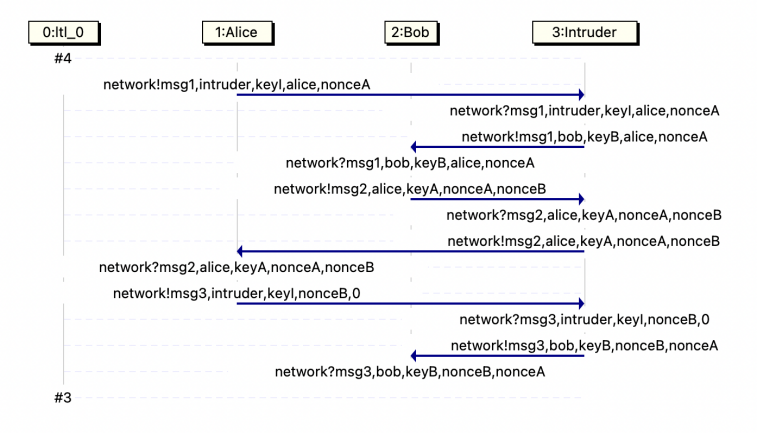
Exercise 2:

The first formula, ’ltl [] (success && aliceBob -> bobAlice) is not violated’, is verified as correct, meaning that the property holds for all possible executions of the protocol. However, the second formula, ’ltl [] (success && bobAlice -> aliceBob) is violated’, is not correct, meaning that there exists at least one execution of the protocol where the property does not hold.

Exercise 3:

The property that is violated produces an execution sequence where the specified property does not hold. Based on the trail file, the length of the execution sequence that violated the property is 84 steps. This is the total number of steps taken during the execution of the protocol before the assertion was violated.

Exercise 4:



The attack was successful because the intruder was able to intercept the messages exchanged between Alice and Bob and alter them in such a way that Bob believed she was communicating with Alice when in fact he was communicating with the intruder. Specifically, the intruder was able to send messages to Bob and sign them with Alice's key, thereby convincing Bob that he was receiving messages from Alice when in fact they were coming from the intruder.

1.5 HW 11

$$A_1|A_2$$

$$A_1;A_2$$

$$A*$$

1.6 HW 12

HackMD Page

Exercise 1: *Have a look at peterson.py. What program behavior do you expect? Is your expectation confirmed when you run the program?*

peterson.py uses a busy waiting lock, my expectation that this is adequate to ensure a result of 20,000 is confirmed by running the program

Exercise 2: *Analyse the Java program peterson in the same way as you analysed the Python program in the previous exercise. Make sure to run the Java program on your local machine. What observations do you make?*

peterson in Java follows the same busy waiting lock scheme as in the python code.

Exercise 3: *Explain why the outcome a = 0, b = 0 is not sequentially consistent, but the other three outcomes are.*

The way the program is written, this outcome would only arise if one of the threads failed or the order of operations was modified within the threads. Assuming the threads execute synchronously, and memory is properly shared between threads, then x and/or y will always be set to 1 before the assignment of a and/or b.

Exercise 4: *Report the results you get from running memoryModelWithStats on your local machine. Include the specs of our processor, in particular the number of cores. If you can find out something about the cashes, add this as well.*

```
Outcomes after 1000 iterations:
(0, 0): 4
(0, 1): 487
(1, 0): 509https://www.overleaf.com/project/63f2ff20adaf2530bdc3f4fa
(1, 1): 0

Hardware Overview:

  Model Name: MacBook Pro
  Chip: Apple M2 Pro
  Total Number of Cores: 12 (8 performance and 4 efficiency)
  Memory: 32 GB
```

Exercise 5: *You can force sequential consistency of memoryModelWithStats by declaring certain variables volatile. In general, declaring variables as volatile comes at cost in execution time, so we want to use this sparingly. Which variables must be declare as volatile to ensure sequential consistency? Measure and report the effect that volatile has on your run time (I use java Main | gnomon).*

Declaring variables (x, y, a, and b) as volatile will achieve sequential consistency.

I was unable to get gnomon running but chatGPT indicates this may come at a cost in execution time, as accesses to volatile variables may be slower than non-volatile variables.

2 Conclusions

In this document, to help you getting started, I gave a first succinct example of typesetting in Latex.

References

[ALG] Algorithm Analysis, Chapman University, 2023.