**Liveness Verifier**

This code generates a file in the .smv format , which can then be used to verify properties using the model checker NUSMV. The file is loaded on the checker and the boolean property to be verified is the specified .,then the checker verifies if the property holds by generating a transition system and running simulations or gives a contradiction , when the property does not hold

**Input :**

1: Network table in a particular format

2: Property to be verified as text

Example, for a Firewall the input will be two files firewall.txt which has the rules table and atomic proposition txt:

\*\*Please follow the below format with spacing for the files

Firewall.txt --->

rule active pro match action

R0 false 100 src=I drop()

R1 false 50 src=I,dst=D drop(),add(R0)

R2 false 50 src=I,dst=F send(),add(R1)

R3 true 40 dst=I,port=2222 send(),add(R2)

R4 true 10 \* send()

property.txt ---> lets call this as Q

src=I,dst=E;send(IDPS)

**Command to run: python3 pysmvc.py Firewall.txt property.txt**

We get this Boolean formulae for the above proposition .

(r0.active=TRUE&TRUE)&r2.active=TRUE&TRUE&FALSE|!(r2.active=TRUE&TRUE)&r5.active=TRUE&TRUE&FALSE|!(r5.active=TRUE&TRUE)&TRUE

Now for the property.txt ---> lets call this as P

src=E,dst=I;send(IDPS)

Boolean formulae :

!(r1.active=TRUE&TRUE)&!(r3.active=TRUE&TRUE)&r4.active=TRUE&TRUE&FALSE|!(r4.active=TRUE&TRUE)&r5.active=TRUE&TRUE&FALSE|!(r5.active=TRUE&TRUE)&TRUE

The whole property in text ---->

a stateful firewall at the periphery of an enterprise that allows endpoint E (e.g., as an external host) to talk to I (e.g., as an internal host) only if I send a request to E first.

In term of boolean formulae form property P and Q from above we get :

(¬p∪q)∧(q→Circle Square p)

Circle : User operator for next , Square ;User operator for always

(¬p∪q)∧(q→next always p)

The whole property now in terms of boolean formulae:

"((!(!(r1.active=TRUE&TRUE)&!(r3.active=TRUE&TRUE)&r4.active=TRUE&TRUE&FALSE|!(r4.active=TRUE&TRUE)&r5.active=TRUE&TRUE&FALSE|!(r5.active=TRUE&TRUE)&TRUE)U(!(r0.active=TRUE&TRUE)&r2.active=TRUE&TRUE&FALSE|!(r2.active=TRUE&TRUE)&r5.active=TRUE&TRUE&FALSE|!(r5.active=TRUE&TRUE)&TRUE))&((!(r0.active=TRUE&TRUE)&r2.active=TRUE&TRUE&FALSE|!(r2.active=TRUE&TRUE)&r5.active=TRUE&TRUE&FALSE|!(r5.active=TRUE&TRUE)&TRUE)->(X(G(!(r1.active=TRUE&TRUE)&!(r3.active=TRUE&TRUE)&r4.active=TRUE&TRUE&FALSE|!(r4.active=TRUE&TRUE)&r5.active=TRUE&TRUE&FALSE|!(r5.active=TRUE&TRUE)&TRUE)))))"

The .smv file is loaded , the model is built and then the above property is simulated , to check if it holds in the network or a counter example exists where the property does not hold.

The generated .smv fie is then loaded on the NUSMV checker and the above property is verified by the command :

check\_ltlspec -p "((!(!(r1.active=TRUE&TRUE)&!(r3.active=TRUE&TRUE)&r4.active=TRUE&TRUE&FALSE|!(r4.active=TRUE&TRUE)&r5.active=TRUE&TRUE&FALSE|!(r5.active=TRUE&TRUE)&TRUE)U(!(r0.active=TRUE&TRUE)&r2.active=TRUE&TRUE&FALSE|!(r2.active=TRUE&TRUE)&r5.active=TRUE&TRUE&FALSE|!(r5.active=TRUE&TRUE)&TRUE))&((!(r0.active=TRUE&TRUE)&r2.active=TRUE&TRUE&FALSE|!(r2.active=TRUE&TRUE)&r5.active=TRUE&TRUE&FALSE|!(r5.active=TRUE&TRUE)&TRUE)->(X(G(!(r1.active=TRUE&TRUE)&!(r3.active=TRUE&TRUE)&r4.active=TRUE&TRUE&FALSE|!(r4.active=TRUE&TRUE)&r5.active=TRUE&TRUE&FALSE|!(r5.active=TRUE&TRUE)&TRUE)))))"

Similarly, the whole property is realized first , atomic propositions are generated and the model is loaded in NUSMV . Then the atomic propositions are converted into boolean formulae and the complete boolean property is obtain dwhoch is then verified on NUSMV by using NUSMV operators