VERIFICATION OF CONCURRENT PROGRAM WITH PROBABILISTIC INSTRUCTION

A Project Report Submitted

by

SHUBH AGARWAL (190828)

HARSH TRIVEDI (200422)

Supervisor :- Dr. Subhajit Roy

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

16 NOVEMBER 2022

ACKNOWLEDGEMENTS

We would like to thank **Professor Subhajit Roy** for providing this wonderful opportunity of working on this project. We are thankful to him for providing us the guidance throughout the project and giving us his precious time.

ABSTRACT

Given a parallel program with probabilistic instructions, our program checks whether we can establish a proof in order to verify the program based on pre-condition and post-condition given by the user. For this the user has to encode the given code in a json format and then give it as input to our program with the pre-condition and the post-condition and our program finds the pre-condition(expectation) on its own following the set of rules and then it compares the given pre-condition with condition that it has found out and finally gives the answer.

In case of parallel program we check the interference free nature between any 2 instructions belonging to different threads, and for that we have to check whether the pre-condition of any instruction in one thread is affected by running any instruction in some other thread. In case of probabilistic programs we used the notion of weakest pre-expectation and not the condition because in these programs everything happens with certain probability. We used the rules of pGCL to find the weakest pre-expectation of the program and then compared the given pre-expectation with our weakest pre expectation for the verification of the proof.

TABLE OF CONTENTS

CHAPTER 1

Introduction

The current software industry suffers a huge loss due to errors in their programs, and most of the time the test suite is not able to cover the whole program. Now currently we have many tools that can check for the program correctness, but not all programs can be checked for correctness due to there underlying complexity.
In the current world most of the work is done on a multiprocessor CPU because it is highly efficient as compared to single processor. And also many algorithms used by the companies are approximate algorithm (that is they gave right answer most of the time) which introduce the probabilistic nature in programs. So keeping these 2 different nature in mind we developed an early stage tool which can check for the correctness of parallel programs with probabilistic instruction.

1.1 Hoare Logic

[1] The Hoare logic is the collection of some rules which can be used to proof the correctness of the program. By correctness we mean that whether the program does what is intended by the user, which can be expressed in terms of assertions on the variable before the execution and after the execution. So we represent the required condition in this form

$$P\{Q\}R$$

P = Pre-Condition (or initial assertion) on the variables
Q = Program which we have to check
R = Final Assertion after the completion of Q
And this whole can be interpretted as if P is true before running Q, then R will be true after completion of Q[**?**]. Now we go through the basic axioms which formed the basis of Hoare Logic:

1. Axiom of Assignment:
$$\{P[E/x]\}\ x = E\{P\}$$

where $P[E/x]$ is obtained by replacing all x with E in P.

2. Rule of Consequences

$$(\{P\}Q\{R\}, R \implies S) \implies \{P\}Q\{S\}$$

$$(\{P\}Q\{R\}, S \implies P) \implies \{S\}Q\{R\}$$

3. Rule of Composition

$$\{P\}Q_1\{R\}, \{R\}Q_2\{S\} \implies \{P\}Q_1; Q_2\{S\}$$

So these 3 can be used to proof simple programs but for both parallel and probabilistic instructions we have to introduce new rules.

1.2 Probabilistic Rules

[2] Firstly we will introduce 2 new instruction so that we can write probabilistic programs.

1. So the instruction allow us to perform P with probability p and Q with probability (1-p).

$$P[p]Q$$

2. This says that perform either P or Q but with some non deterministic probability. It is also known as demonic choice.

$$P[]Q$$

Now in case of probabilistic programs instead of relying on assertion we have to use expectation, because every variable takes certain value with some probability.

Weakest Pre Expectation :

$$wp(prog, post)$$

This is the maximum value of post that we can guarentee to be after running the program(prog). So we will introduce some rules related to weakest pre-expectation.

Table 1.1: Rules for weakest pre expectation[2]

syntax prog	semantics wp(prog,f)
skip	f
abort	0
x:=E	f[x/E]
P;Q	wp(P,wp(Q,f))
if(G) P else Q	[G]wp(P,f)+[-G]wp(Q,f)
P [] Q	min(wp(P,f),wp(Q,f))
P [p] Q	p.wp(P,f)+(1-p).wp(Q,f)

1.3 Parallel Programs

[3] Suppose we have programs running in parallel, $Q_1, Q_2, \ldots Q_n$ are programs running in parallel. If we want to show the correctness of the parallel programs we need to follow the following steps

- First we need to find out proofs for $\{P_i\}Q_i\{R_i\}$ without taking into the effects of the other program
- Second need to show that the proofs $\{P_i\}Q_i\{R_i\}$ and $\{P_j\}Q_j\{R_j\}$ are interference-free
  Let Q'_i be a statement from Q_i and Q'_j be a statement from Q_j, if we want to show that Q'_i and Q'_j are interference free, we need to show the following
  - $\{pre(Q'_i) \wedge pre(Q'_j)\}\ Q'_i\ \{pre(Q'_j)\}$
  - $\{pre(Q'_i) \wedge pre(Q'_j)\}\ Q'_j\ \{pre(Q'_i)\}$

  Here pre(Q) means the conditions before running the statement Q

CHAPTER 2

Implementation

2.1 Input Format

The input to our program is a JSON file, which has the following key value pairs

- variables_used - These are the variables which are used in the input program. It should be given as a string. They should be all single letter characters.
- program_type - It is a list which should contain only one element "probabilistic"
- P - the weakest pre expectation given by the user
- Q - the details about the instructions in the program. This is again a dictionary, whose keys are the thread and the value for each thread is a list of the instructions to be run on each thread. Each instruction is represented in following manner
  - instruction_type - denotes the type of the instruction, can be assignment, condition_on_prob and demonic_choice
  - lhs - denotes the left hand side of an assignment, present only in case when instruction_type is assignment.
  - rhs - denotes the right hand side of an assignment, present only in case when instruction_type is assignment.
  - previous_possible_instructions - a list containing the indices of the program instructions which can be executed just before the current instruction. It is [-1] in case of first instruction of a thread.
  - probability - The probability with which the we execute the if condition. 1-probability will be the probability for executing the else part. It is present only in case when instruction_type is condition_on_prob.
- I - for each thread we ask for the post expectation expression . It is again a dictionary with key denoting the thread and the value containing the expression.
- R - post expectation given by the user

2.2 Proofs evaluated

In order to verify the program given by the user we do the following steps

- Firstly we calculate the pre-expectation for each of the thread using I for that thread without considering any effects from the other threads. Let the pre-expectation of threads be denoted as I' (for thread 1 let us denote it as I_1', for thread 2 as I_2'). Here let n is the number of threads.

- Next we need to prove that

  $$P \leq I_1' + I_2' + \ldots I_n'$$

  is valid.
  So we try to check whether

  $$P > I_1' + I_2' + \ldots I_n'$$

  is satisfiable or not. If it is not satisfiable then we say that $P \leq I_1' + I_2' + \ldots I_n'$ is valid else not.

- Next we need to prove that

  $$I_1 + I_2 + \ldots I_n \leq R$$

  is valid or not. Again we use the method shown in step above to establish the proof.

- Establish the proof of interference free properties.
  Let $\{P_i'\}Q_i'\{R_i'\}$ be the pre-expecation calculation of a single statement Q_i' which is taken from threads i step 1. Similarly let $\{P_j'\}Q_j'\{R_j'\}$ be the pre-expecation calculation of a single statement Q_j' which is taken from threads j step 1.
  Basically the above says

  $$P_i' \leq wpe(Q_i', R_i')$$
  $$P_j' \leq wpe(Q_j', R_j')$$

  So for interference free we need to show the validity of the following

  $$P_i' \leq wpe(Q_j', P_i')$$

  $$P_j' \leq wpe(Q_i', P_j')$$

  Again we use the standard technique used in step 2 to check for validity.

CHAPTER 3

Examples

1. Simple single threaded code

$$\{x + 4\}x = x + 5\{x\}$$

so we have the pre-expectation (P) given by the user as x+4 for the post expectation x (I_1) and the total post expectation given by the user was also x(R) and our program find the pre-expectation as x+5 (I_1') (by simple substitution) and then we have to show the validity of

$$(P \leq I_1') \wedge (I_1 \leq R)$$

$P \leq I_1'$ reduces to

$$x + 4 \leq x + 5$$

is valid or not and for that we check the satisfiablity of

$$x + 4 > x + 5$$

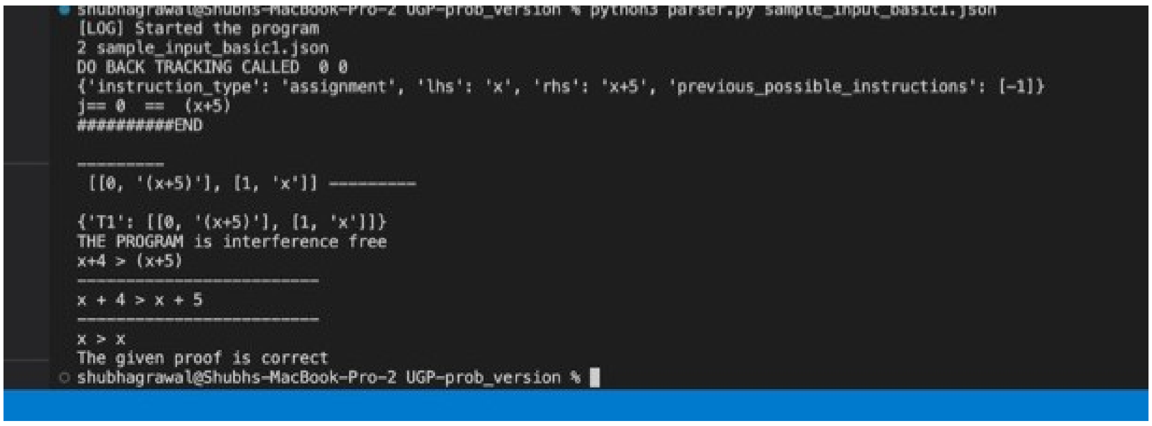
which is unsat and hence $P \leq I_1'$ is valid and now we have $I_1 \leq R$, which reduces to

$$x \leq x$$

is valid or not and for that we check the satisfiablity of

$$x > x$$

which is again unsat and $I_1 \leq R$ is valid and hence our proof is valid (here we don't check for interfernce free as the current code only has single thread).



```
shubhagrawal@Shubhs-MacBook-Pro-2 UGP-prob_version % python3 parser.py sample_input_basic1.json
[LOG] Started the program
2 sample_input_basic1.json
DO BACK TRACKING CALLED  0 0
{'instruction_type': 'assignment', 'lhs': 'x', 'rhs': 'x+5', 'previous_possible_instructions': [-1]}
j== 0  ==  (x+5)
###########END

---------
 [[0, '(x+5)'], [1, 'x']] ---------

{'T1': [[0, '(x+5)'], [1, 'x']]}
THE PROGRAM is interference free
x+4 > (x+5)
-------------------------
x + 4 > x + 5
-------------------------
x > x
The given proof is correct
shubhagrawal@Shubhs-MacBook-Pro-2 UGP-prob_version %
```

2. Double threaded probabilistic program
   $\{2.25 * y + 10.5 + 2.5 * x\}$
   Thread1:

```
u=3y+6[0.25]u=2y+4;
u=u+5;
```

   $\{u\}$

   Thread2:

```
v=3x+6[0.5]v=2x+4;
v=v+5;
```

   $\{v\}$
   $\{u + v\}$
   so in thread1 we choose between the 2 choices with 0.25 probability, followed by a simple assignment.
   And also since it is a double threaded program, we have to check for the interference free nature of the 2 threads. Now again we have $P = 2.25y + 10.5 + 2.5x$, $I_1 = u$, $I_2 = v$ and $R = u + v$. Now we find I'_1 and I'_2 for the respective threads.

$$I'_1 = 0.25 * ((3y + 11)) + 0.75 * (2y + 9)$$
$$I'_2 = 0.5 * (3x + 11) + 0.5 * (2x + 9)$$

   And now we have to check the validity of the following expression

$$(P \leq I'_1 + I'_2) \wedge (I_1 + I_2 \leq R)$$

   Now $(P \leq I'_1 + I'_2)$ reduces to

$$2.25y+10.5+2.5x \leq 0.25*(3y+11)+0.75*(2y+9)+0.5*(3x+11)+0.5*(2x+9)$$

   For this we check the satisfiablity of

$$2.25y+10.5+2.5x > 0.25*(3y+11)+0.75*(2y+9)+0.5*(3x+11)+0.5*(2x+9)$$

   which is UNSAT and hence $(P \leq I'_1 + I'_2)$ is correct
   Now $I_1 + I_2 \leq R$ reduces to

$$u + v \leq u + v$$

   and again to check it validity we check the satisfiablility of

$$u + v > u + v$$

   which is UNSAT and hence $I_1 + I_2 \leq R$ is also valid and now we check the interfernce free nature we do the checks as shown in step 4 of the proofs evaluated. Let us take an example of the same.
   Let Q'_i be $u = 3y + 6[0.25]u = 2y + 4$; and Q'_j be $v = v + 5$;.

Here from step 1 we had $P_i' = 0.25 * (3y + 11) + 0.75 * (2y + 9)$
Now we need to show that

$$0.25*(3y+11)+0.75*(2y+9) \leq wpe(v = v+5, 0.25*(3*y+11)+0.75*(2y+9))$$

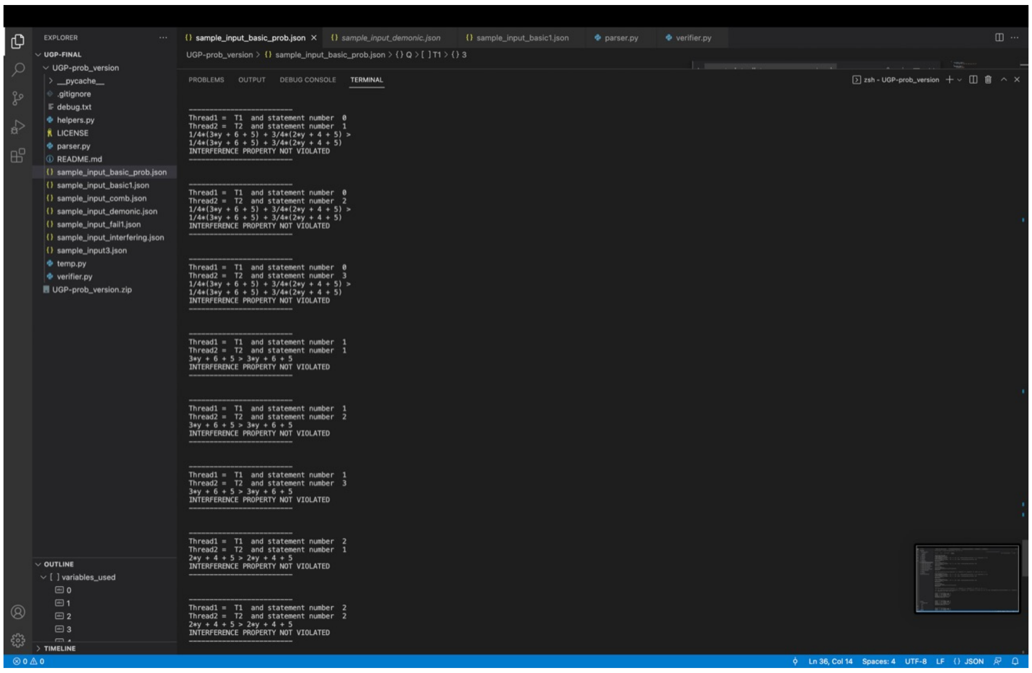
is valid. Using the rules of weakest pre-expectation calculation we have

$$wpe(v = v+5, 0.25*(3*y+11)+0.75*(2y+9)) = 0.25*(3*y+11)+0.75*(2y+9)$$

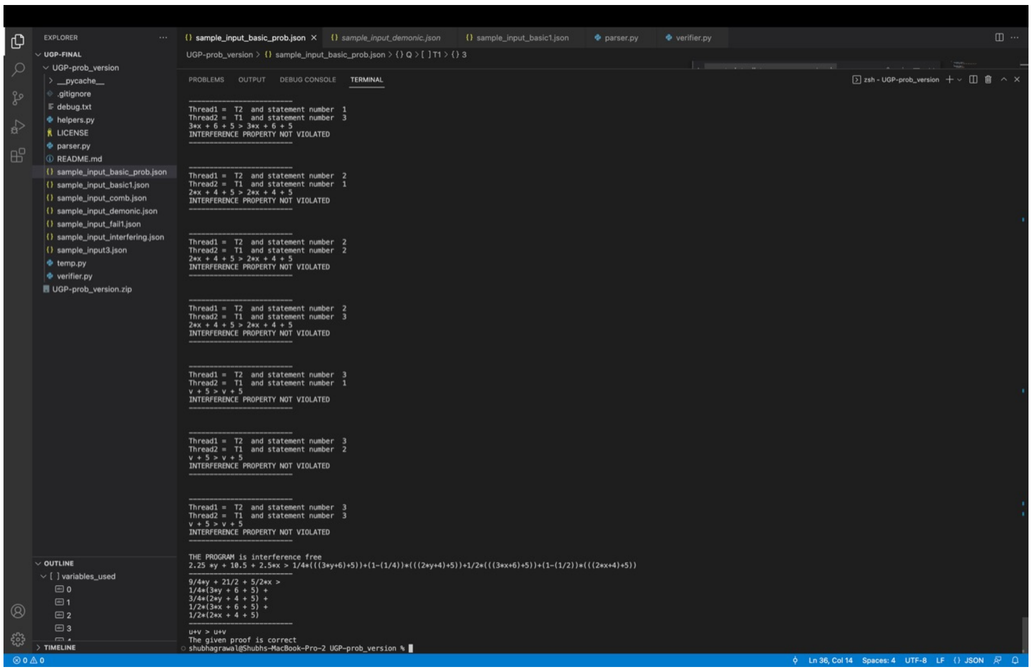
So we now check for satisfiability of

$$0.25 * (3y + 11) + 0.75 * (2y + 9) > 0.25 * (3y + 11) + 0.75 * (2y + 9)$$

which is UNSAT and hence one of interference free has been established, we can do the similar analysis for the reverse was as well.
Hence our proof is correct
Here are some screenshots for the interfernce free and the final check on the pre-condtion



```
--------------------------
Thread1 =  T1  and statement number  0
Thread2 =  T2  and statement number  1
1/4*(3*y + 6 + 5) + 3/4*(2*y + 4 + 5) >
1/4*(3*y + 6 + 5) + 3/4*(2*y + 4 + 5)
INTERFERENCE PROPERTY NOT VIOLATED
--------------------------


--------------------------
Thread1 =  T1  and statement number  0
Thread2 =  T2  and statement number  2
1/4*(3*y + 6 + 5) + 3/4*(2*y + 4 + 5) >
1/4*(3*y + 6 + 5) + 3/4*(2*y + 4 + 5)
INTERFERENCE PROPERTY NOT VIOLATED
--------------------------


--------------------------
Thread1 =  T1  and statement number  0
Thread2 =  T2  and statement number  3
1/4*(3*y + 6 + 5) + 3/4*(2*y + 4 + 5) >
1/4*(3*y + 6 + 5) + 3/4*(2*y + 4 + 5)
INTERFERENCE PROPERTY NOT VIOLATED
--------------------------


--------------------------
Thread1 =  T1  and statement number  1
Thread2 =  T2  and statement number  1
3*y + 6 + 5 > 3*y + 6 + 5
INTERFERENCE PROPERTY NOT VIOLATED
--------------------------


--------------------------
Thread1 =  T1  and statement number  1
Thread2 =  T2  and statement number  2
3*y + 6 + 5 > 3*y + 6 + 5
INTERFERENCE PROPERTY NOT VIOLATED
--------------------------


--------------------------
Thread1 =  T1  and statement number  1
Thread2 =  T2  and statement number  3
3*y + 6 + 5 > 3*y + 6 + 5
INTERFERENCE PROPERTY NOT VIOLATED
--------------------------


--------------------------
Thread1 =  T1  and statement number  2
Thread2 =  T2  and statement number  1
2*y + 4 + 5 > 2*y + 4 + 5
INTERFERENCE PROPERTY NOT VIOLATED
--------------------------


--------------------------
Thread1 =  T1  and statement number  2
Thread2 =  T2  and statement number  2
2*y + 4 + 5 > 2*y + 4 + 5
INTERFERENCE PROPERTY NOT VIOLATED
--------------------------
```



```
-------------------------
Thread1 =  T2  and statement number  1
Thread2 =  T1  and statement number  3
3*x + 6 + 5 > 3*x + 6 + 5
INTERFERENCE PROPERTY NOT VIOLATED
-------------------------


-------------------------
Thread1 =  T2  and statement number  2
Thread2 =  T1  and statement number  1
2*x + 4 + 5 > 2*x + 4 + 5
INTERFERENCE PROPERTY NOT VIOLATED
-------------------------


-------------------------
Thread1 =  T2  and statement number  2
Thread2 =  T1  and statement number  2
2*x + 4 + 5 > 2*x + 4 + 5
INTERFERENCE PROPERTY NOT VIOLATED
-------------------------


-------------------------
Thread1 =  T2  and statement number  2
Thread2 =  T1  and statement number  3
2*x + 4 + 5 > 2*x + 4 + 5
INTERFERENCE PROPERTY NOT VIOLATED
-------------------------


-------------------------
Thread1 =  T2  and statement number  3
Thread2 =  T1  and statement number  1
v + 5 > v + 5
INTERFERENCE PROPERTY NOT VIOLATED
-------------------------


-------------------------
Thread1 =  T2  and statement number  3
Thread2 =  T1  and statement number  2
v + 5 > v + 5
INTERFERENCE PROPERTY NOT VIOLATED
-------------------------


-------------------------
Thread1 =  T2  and statement number  3
Thread2 =  T1  and statement number  3
v + 5 > v + 5
INTERFERENCE PROPERTY NOT VIOLATED
-------------------------

THE PROGRAM is interference free
2.25 *y + 10.5 + 2.5*x > 1/4*(((3*y+6)+5))+(1-(1/4))*(((2*y+4)+5))+1/2*(((3*x+6)+5))+(1-(1/2))*(((2*x+4)+5))
-------------------------
9/4*y + 21/2 + 5/2*x >
1/4*(3*y + 6 + 5) +
3/4*(2*y + 4 + 5) +
1/2*(3*x + 6 + 5) +
1/2*(2*x + 4 + 5)
-------------------------
u+v > u+v
The given proof is correct
shubhagrawal@Shubhs-MacBook-Pro-2 UGP-prob_version %
```

3. Introducing Demonic choice with simple double threaded program
   $\{7\}$
   Thread1:

   ```
   u=1[]u=0;
   u=u+5;
   ```

   $\{u\}$

   Thread2:

   ```
   v=1[]v=2;
   v=v+4;
   ```

   $\{v\}$
   $\{u+v\}$
   so in thread1 we choose between the 2 choices with some non determisninstic probability, followed by a simple assignment and same for thread2.
   And also since it is a double threaded program, we have to check for the interference free nature of the 2 threads. Now again we have $P = 7$, $I_1 = u$, $I_2 = v$ and $R = u + v$. Now we find I_1' and I_2' for the respective threads.

$$I_1' = min(1 + 5, 0 + 5)$$
$$I_2' = min(1 + 4, 2 + 4)$$

   And now we have to check the validity of the following expression

$$(P \leq I_1' + I_2') \wedge (I_1 + I_2 \leq R)$$

Now $(P \leq I_1' + I_2')$ reduces to

$$7 \leq min(1+5, 0+5) + min(1+4, 2+4)$$

For this we check the satisfiablity of

$$7 > min(1+5, 0+5) + min(1+4, 2+4)$$

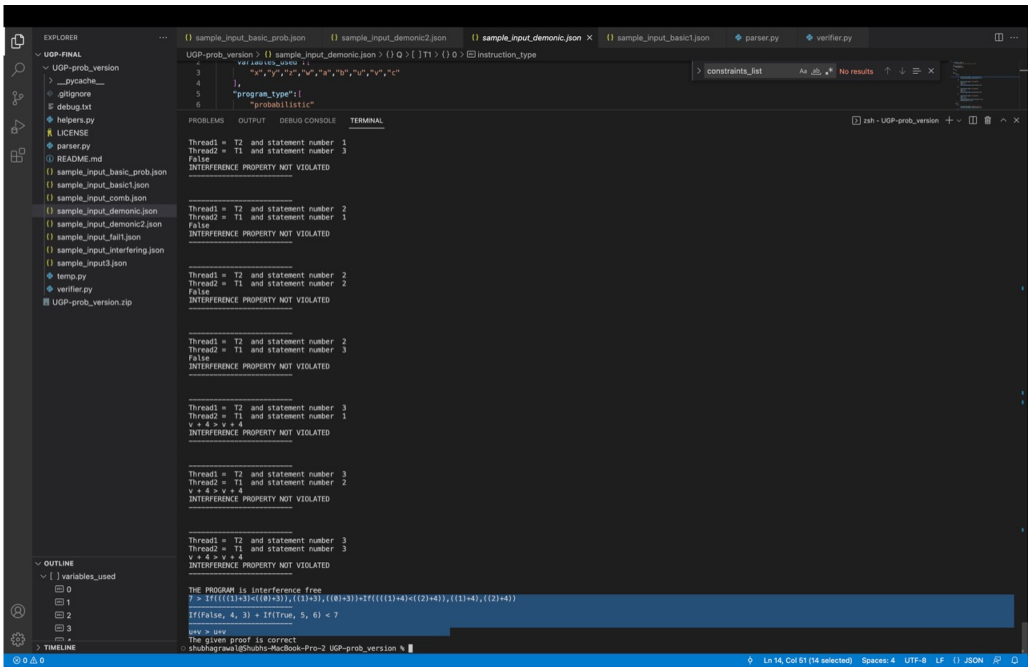
which is UNSAT and hence $(P \leq I_1' + I_2')$ is correct
Now $I_1 + I_2 \leq R$ reduces to

$$u + v \leq u + v$$

and again to check it validity we check the satisfiablility of

$$u + v > u + v$$

which is UNSAT and hence $I_1 + I_2 \leq R$ is also valid and now we check the interfernce free nature of the program by using rule 4 which all evaluates to true. Hence our proof is correct
Here are some screenshots for the interfernce free and the final check on the precondtion



```
        "x","y","z","w","a","b","u","v","c"
    ],
    "program_type":[
        "probabilistic"
```

```
Thread1 =  T2  and statement number  1
Thread2 =  T1  and statement number  3
False
INTERFERENCE PROPERTY NOT VIOLATED
-------------------------


-------------------------
Thread1 =  T2  and statement number  2
Thread2 =  T1  and statement number  1
False
INTERFERENCE PROPERTY NOT VIOLATED
-------------------------


-------------------------
Thread1 =  T2  and statement number  2
Thread2 =  T1  and statement number  2
False
INTERFERENCE PROPERTY NOT VIOLATED
-------------------------


-------------------------
Thread1 =  T2  and statement number  2
Thread2 =  T1  and statement number  3
False
INTERFERENCE PROPERTY NOT VIOLATED
-------------------------


-------------------------
Thread1 =  T2  and statement number  3
Thread2 =  T1  and statement number  1
v + 4 > v + 4
INTERFERENCE PROPERTY NOT VIOLATED
-------------------------


-------------------------
Thread1 =  T2  and statement number  3
Thread2 =  T1  and statement number  2
v + 4 > v + 4
INTERFERENCE PROPERTY NOT VIOLATED
-------------------------


-------------------------
Thread1 =  T2  and statement number  3
Thread2 =  T1  and statement number  3
v + 4 > v + 4
INTERFERENCE PROPERTY NOT VIOLATED
-------------------------

THE PROGRAM is interference free
7 > If((((1)+3)<((0)+3)),((1)+3),((0)+3))+If((((1)+4)<((2)+4)),((1)+4),((2)+4))
-------------------------
If(False, 4, 3) + If(True, 5, 6) < 7
-------------------------
u+v > u+v
The given proof is correct
shubhagrawal@Shubhs-MacBook-Pro-2 UGP-prob_version %
```

4. Found the counter Example
   Now we used the same example as example 3 but now instead we used the demonic choice.
   $\{2.25 * y + 10.5 + 2.5 * x\}$
   Thread1:

```
u=3y+6[]u=2y+4;
u=u+5;
```

$\{u\}$

Thread2:

```
v=3x+6[]v=2x+4;
v=v+5;
```

$\{v\}$

$\{u+v\}$

Now again we have $P = 2.25y + 10.5 + 2.5x$, $I_1 = u$, $I_2 = v$ and $R = u + v$. Now we find I'_1 and I'_2 for the respective threads.

$$I'_1 = min(3y + 11, 2y + 9)$$
$$I'_2 = min(3x + 11, 2x + 9)$$

And now we have to check the validity of the following expression

$$(P \leq I'_1 + I'_2) \wedge (I_1 + I_2 \leq R)$$

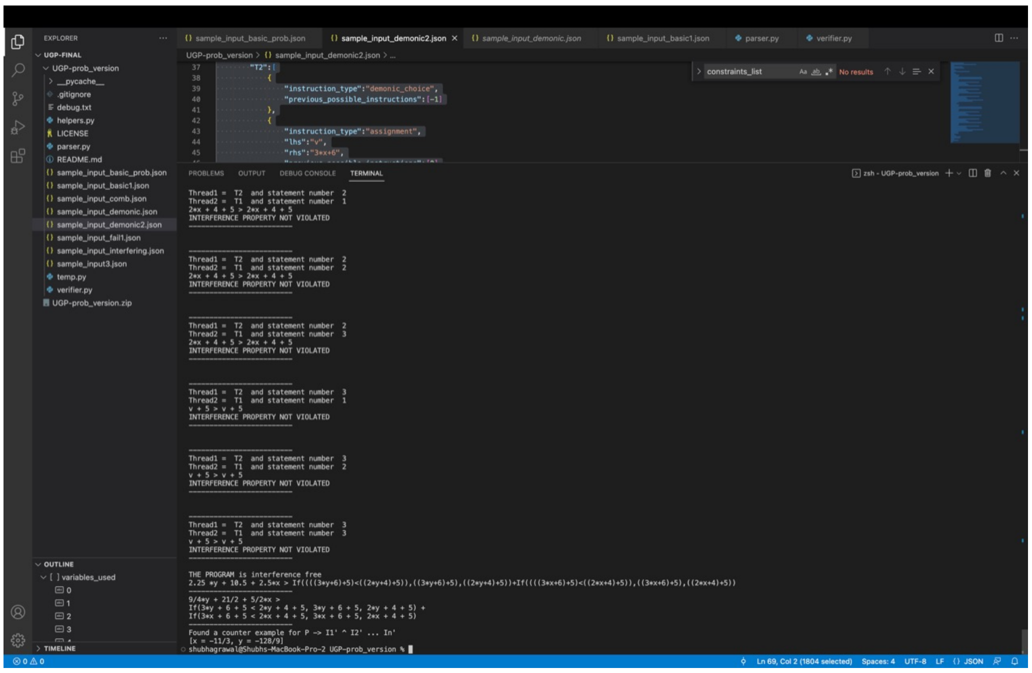
Now $(P \leq I'_1 + I'_2)$ reduces to

$$2.25y + 19.5 + 2.5x \leq min(3y + 11, 2y + 9) + min(3x + 11, 2x + 9)$$

For this we check the satisfiablity of

$$2.25y + 19.5 + 2.5x > min(3y + 11, 2y + 9) + min(3x + 11, 2x + 9)$$

which is SAT with x=-11/3, y=-128/9 and hence $(P \leq I'_1 + I'_2)$ is not valid and we stop here.



```
            "T2":[
                {
                    "instruction_type":"demonic_choice",
                    "previous_possible_instructions":[-1]
                },
                {
                    "instruction_type":"assignment",
                    "lhs":"v",
                    "rhs":"3*x+6",
```

```
Thread1 =  T2  and statement number  2
Thread2 =  T1  and statement number  1
2*x + 4 + 5 > 2*x + 4 + 5
INTERFERENCE PROPERTY NOT VIOLATED
-------------------------


-------------------------
Thread1 =  T2  and statement number  2
Thread2 =  T1  and statement number  2
2*x + 4 + 5 > 2*x + 4 + 5
INTERFERENCE PROPERTY NOT VIOLATED
-------------------------


-------------------------
Thread1 =  T2  and statement number  2
Thread2 =  T1  and statement number  3
2*x + 4 + 5 > 2*x + 4 + 5
INTERFERENCE PROPERTY NOT VIOLATED
-------------------------


-------------------------
Thread1 =  T2  and statement number  3
Thread2 =  T1  and statement number  1
v + 5 > v + 5
INTERFERENCE PROPERTY NOT VIOLATED
-------------------------


-------------------------
Thread1 =  T2  and statement number  3
Thread2 =  T1  and statement number  2
v + 5 > v + 5
INTERFERENCE PROPERTY NOT VIOLATED
-------------------------


-------------------------
Thread1 =  T2  and statement number  3
Thread2 =  T1  and statement number  3
v + 5 > v + 5
INTERFERENCE PROPERTY NOT VIOLATED
-------------------------

THE PROGRAM is interference free
2.25 *y + 10.5 + 2.5*x > If((((3*y+6)+5)<((2*y+4)+5)),((3*y+6)+5),((2*y+4)+5))+If((((3*x+6)+5)<((2*x+4)+5)),((3*x+6)+5),((2*x+4)+5))
-------------------------
9/4*y + 21/2 + 5/2*x >
If(3*y + 6 + 5 < 2*y + 4 + 5, 3*y + 6 + 5, 2*y + 4 + 5) +
If(3*x + 6 + 5 < 2*x + 4 + 5, 3*x + 6 + 5, 2*x + 4 + 5)
-------------------------
Found a counter example for P -> I1' ^ I2' ... In'
[x = -11/3, y = -128/9]
shubhagrawal@Shubhs-MacBook-Pro-2 UGP-prob_version % 
```

5. Example failing on interference check
   $\{2.25 * y + 10.5 + 2.5 * x\}$
   Thread1:

```
u=3y+6[0.25]u=2y+4;
u=u+5;
x=x-5;
```

$\{u\}$

Thread2:

```
v=3x+6[0.5]v=2x+4;
v=v+5;
```

$\{v\}$
$\{u + v\}$

Let us consider the statement $x = x - 5$; from thread 1 and statement $v = 3x + 6[0.5]v = 2x+4$; from the thread2. Here $P_2' = 1/2*(3*x+6+5)+1/2*(2*x+4+5)$. Now we need to show that

$$P_2' \leq wpe(Q_1', P_2')$$

is valid. This reduces to show that

$$P_2' > wpe(Q_1', P_2')$$

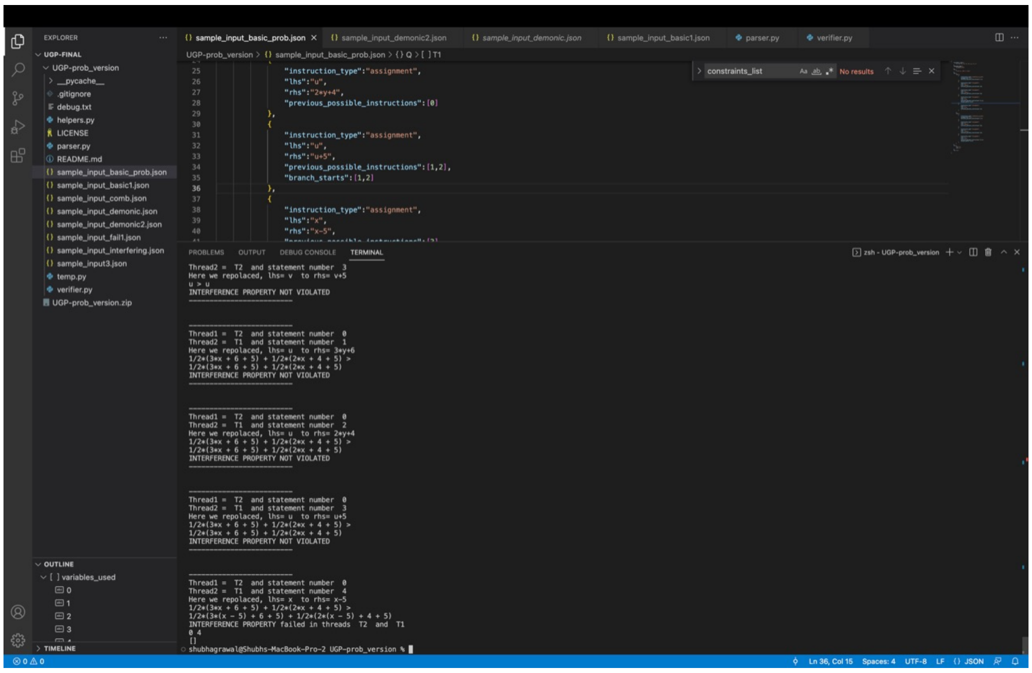
is not satisfiable. We know

$$wpe(Q_1', P_2') = 1/2 * (3 * (x - 5) + 6 + 5) + 1/2 * (2 * (x - 5) + 4 + 5)$$

and equation reduces to

$$1/2*(3*x+6+5)+1/2*(2*x+4+5) > 1/2*(3*(x-5)+6+5)+1/2*(2*(x-5)+4+5)$$

which is satisfiable. Now we can see that the above program is not interfernce free as $x = x - 5$ will affect the result of 2nd thread and our program find the same.



```
                    "instruction_type":"assignment",
                    "lhs":"u",
                    "rhs":"2*y+4",
                    "previous_possible_instructions":[0]
                },
                {
                    "instruction_type":"assignment",
                    "lhs":"u",
                    "rhs":"u+5",
                    "previous_possible_instructions":[1,2],
                    "branch_starts":[1,2]
                },
                {
                    "instruction_type":"assignment",
                    "lhs":"x",
                    "rhs":"x-5",
```

```
Thread2 =  T2  and statement number  3
Here we repolaced, lhs= v  to rhs= v+5
u > u
INTERFERENCE PROPERTY NOT VIOLATED
-------------------------



-------------------------
Thread1 =  T2  and statement number  0
Thread2 =  T1  and statement number  1
Here we repolaced, lhs= u  to rhs= 3*y+6
1/2*(3*x + 6 + 5) + 1/2*(2*x + 4 + 5) >
1/2*(3*x + 6 + 5) + 1/2*(2*x + 4 + 5)
INTERFERENCE PROPERTY NOT VIOLATED
-------------------------



-------------------------
Thread1 =  T2  and statement number  0
Thread2 =  T1  and statement number  2
Here we repolaced, lhs= u  to rhs= 2*y+4
1/2*(3*x + 6 + 5) + 1/2*(2*x + 4 + 5) >
1/2*(3*x + 6 + 5) + 1/2*(2*x + 4 + 5)
INTERFERENCE PROPERTY NOT VIOLATED
-------------------------



-------------------------
Thread1 =  T2  and statement number  0
Thread2 =  T1  and statement number  3
Here we repolaced, lhs= u  to rhs= u+5
1/2*(3*x + 6 + 5) + 1/2*(2*x + 4 + 5) >
1/2*(3*x + 6 + 5) + 1/2*(2*x + 4 + 5)
INTERFERENCE PROPERTY NOT VIOLATED
-------------------------



-------------------------
Thread1 =  T2  and statement number  0
Thread2 =  T1  and statement number  4
Here we repolaced, lhs= x  to rhs= x-5
1/2*(3*x + 6 + 5) + 1/2*(2*x + 4 + 5) >
1/2*(3*(x - 5) + 6 + 5) + 1/2*(2*(x - 5) + 4 + 5)
INTERFERENCE PROPERTY failed in threads  T2  and  T1
0 4
[]
shubhagrawal@Shubhs-MacBook-Pro-2 UGP-prob_version % 
```

CHAPTER 4

Future Development

1. We can create a parser so that user don't have to make the json file for this program.
2. We would like to increase the domain of instruction, also allow nested instructions and locks and various other stuffs.
3. Also if possible we can only ask for pre and post expectation of the whole program with mutlithread and find out local post expectation on our own (finding out I by ourself).

REFERENCES

[1] C. A. R. Hoare, “An axiomatic basis for computer programming,” 1969.

[2] J.-P. K. . A. M. Friedrich Gretz, “On a quest for probabilistic loop invariants,” 2013.

[3] S. Owicki and D. Gries, “An axiomatic proof technique for parallel programs,” 1975.