Assumption : We are aware of the inputs for which we are collecting the dynamic traces from the original binary for essence extraction.

Path directed symbolic execution (PD-SE) : We will add those input constraints while collecting the symbolic execution using angr tool.

For now, we are assuming that we know the corresponding register which is containing the input values and directly adding those constraints in the code file ***logicalSummary.py***.

Function wise symbolic execution : Moreover, for this PD-SE, we have chosen one function rather than the whole program for running symbolic execution. We also have to put the starting address of the function as the starting address in the ***logicalSummary.py*** file.

The function which is changed for the purpose of PD-SE or function wise symbolic execution is the ***runAndFile()*** function in the ***logicalSummary.py*** file.

Test case 1: calculator and its essences (will find the ***Calculator.c*** in the **testSource** folder)

1. The original binary : calculator
2. The extracted essence case 1 : calculator.essence.case1
3. The extracted essence case 2 : calculator.essence.case2

We are going to analyze only the ***doCalculation*** function in the ***Calculator.c*** code. The address of that function in the binary is **0x401189**. So, the exploration will start from this address to collect the symbolic path information for all the cases.

The original ***doCalculation*** will handle all the cases for ***operation*** 1, 2, 3, 4, 5 or not any one of them.
The ***doCalculation*** of the extracted essence case 1 will only operate for ***operation*** 3.
The ***doCalculation*** of the extracted essence case 2 will only operate for ***operation*** 2 or 3.

So, these are the input domain knowledge we have before running the symbolic execution on the programs. We will add the respective input domain before checking the equivalence between extracted and the original program.

Extracted essence case 1 : `state.add_constraints(state.regs.rdx ==0x3)`
Extracted essence case 2 : `state.add_constraints(claripy.Or(state.regs.rdx ==0x3, state.regs.rdx ==0x2))`

After adding these constraints if we collect the path constraints and corresponding return values, then they will result the same.

If we do not add these constraints in the ***logicalSummary.py*** file in the ***runAndFind*** function, they will not be equivalent.

Test case2 : calc2 and its essences(will find the ***calc2.cpp*** in the **testSource** folder)

1. The original binary : calc2
2. The extracted essence case 1 : calc2.essence.case1
3. The extracted essence case 2 : calc2.essence.case2
4. The extracted essence case 3 : calc2.essence.case3

We are going to analyze only the ***calculation*** function in the ***calc2.c*** code. The address of that function in the binary is **0x401289**. So, the exploration will start from this address to collect the symbolic path information for all the cases.

The original ***calculation*** will handle all the cases for ***oper*** character '+', '-', '*', '/', '^', 's' or not any one of them.
The ***calculation*** of the extracted essence case 1 will only operate for ***oper*** 's'.
The ***calculation*** of the extracted essence case 2 will only operate for ***oper*** 's' or '^'.
The ***calculation*** of the extracted essence case 3 will only operate for ***oper*** 's' or '^' or '/'.

So, these are the input domain knowledge we have before running the symbolic execution on the programs. We will add the respective input domain before checking the equivalence between extracted and the original program.

Extracted essence case 1 : `state.add_constraints(state.regs.rdi ==115)`
Extracted essence case 2 : `state.add_constraints(claripy.Or(state.regs.rdi ==115, state.regs.rdi == 94))`
Extracted essence case 2 : `state.add_constraints(claripy.Or(state.regs.rdi ==115, state.regs.rdi == 94, state.regs.rdi == 47))`

After adding these constraints if we collect the path constraints and corresponding return values, then they will result the same.