Name: Akash Shivaji Varude

Roll Number: 231110006

Assignment 4

Implementation Details:

I have considered range i.e. [min val, max val] for abstract representations. Further I have considered rotation angle as theta. Clockwise rotation is considered to be as negative direction rotation and anticlockwise rotation as positive direction rotation.

Initialization function:

if the node is start node, then inintialize it as x=0, y=0 and theta as 0. This initialize is done using dictionary. However this iniaialization is done like x: IntervalDomain(0, 0) where intervalDomain is class where Lattice is implemented

Transfer Function Details:

- Transfer function calculates here OUT value of each Basic block.

1. For Mov type Commands(forward,backward,left,right):

- First, values from command are extracted. For e.g. ‘forward 30’ command, here 30 is extracted and represented in abstract format [30,30]. To represent this in abstract format, I have used **IntervalDomain(Lattice)** class. So mov.L and mov.R stores lower bound and upper bound respectively.
- If command is ‘left’ then just add rotation angle to theta and if command is ‘right’ then subtract rotation angle value from theta.
- When kachua s moving, i.e. for forward and backward commands: I have calculated lower bounds and upper bounds of both x and y axis.

  Below is example for forward: (if backward command then just subtract)
  - Lb of x = current x.L + mov.L * cos(theta)
  - Ub of x = current x.R + mov.R * cos(theta)
  - Lb of y = current y.L + mov.L * cos(theta)
  - Ub of y = current y.R + mov.R * cos(theta)

2. For condition type commands: (i.e. for branches in cfg):

For these type of commands, implementation of **correct lattice** is necessary. We have variable, operator and value.

Let operator is less than '<', so in 'if' condition, if the condition is true, then range of the variable will be:

Lb of variable = minimum till now

Ub of variable = value-1

if the condition is false, then range of the variable will be:

Lb of variable = value

Ub of variable = maximum till now

Similar operations are also performed for '<=', '>', '>=', '==' with corresponding logics

Meet Function:

Created separate lists for lower and upper bounds of x and y both. inserted x lower bounds of predecessors in xlb, and upper bounds of x in xub. Similarl for y.

Now final lower bound is just minimum of all lower bounds and upper bound is maximum of all upper bounds. So I have calculated meetvalue i.e. abstract values of x and y after meet operation

Interval Domain(lattice) :

Class interval domain is initialized with abstract values as interval, self.L means lower bound and self.R means upper bounds

1. isBot function: As bottom is defined as 'None'. This function checks the parameter passed is None or not
2. __str__ : is just to convert into strings.
3. isTop function: As top is defined as (int('-inf'), int('inf')). This function is to check whether abstract value is top or not
4. sub function: To carry out subtraction operation for two intervals(abstract values). So, lb of resultant interval is (lb of first – ub of second) and ub of resultant interval is (ub of first - lb of second)
5. add function: To carry out addition operation for two intervals(abstract values). So, lb of resultant interval is (lb of first + lb of second) and ub of resultant interval is (ub of first + ub of second)
6. join function: if either of two abstract values is top then their join is Top element itself. Otherwise just finding the union of the intervals
7. meet function: if either of two abstract values is bottom then their join is bot element itself. Otherwise just finding the intersection of the intervals
8. __le__: just comparing first abstract value is less than or equal to other abstract value or not. So if ub of first abstract value is <= lb of second abstract value then return true
9. __eq__: compare both lb and ub of both abstract values. Is same then return true
10. __lt__: just comparing first abstract value is less than other abstract value or not. So if ub of first abstract value is < lb of second abstract value then return true

11. __gt__: comparing first abstract value is greater than other abstract value or not. So if lb of first abstract value is > ub of second abstract value then return true
12. __ge__: to check greater than equal to. If lb of first abstract value >= ub of second abstract value then return true

Limitations:

Implementation is done for all if, else, forward, backward, left, right commands of chiron. However, code doesn't work for repeat statements.

Sample Output:

Running on test.tl and test2.json : Kachua is unsafe

```
D:\Mtech IITK\First SEM\PAVT\Assignment_4_231110006\Chiron-Framework-master\ChironCore>chiron.py --control_flow -ai examples/test1.tl
Chiron v5.3
```

```
:a = 10
if :a <= 10[
        right 90
        forward 30
]
else[
        forward 40
]

 Possibility of kachua to halt in the region : X from  0  TO  40  and y from  -30  to  0

 Magarmach region : X from  -6  TO  20  and y from  -5  to  -40
Hence there is overlap between possibile region where kachua stops and magarmach region

",{VERIFIED} Kachua can be inside magarmach region. Hence Unsafe
== Abstract Interpretation ==
```

Running on test3.tl and test3.json: Kachua is safe