<u>One variable and One action</u>

Example 1: Different actions for same range - Conflict

1: If 100<m<120 then output = 200
2: If 100<m<120 then output = 300

Code -

```
(define-fun range ((x Int) (lower Int) (upper Int)) Bool (and (< lower x) (< x upper)))

(declare-fun m () Int)

(define-fun rule1_applies () Bool ( and (range m 100 120)))

(declare-fun output0 () Int)

(define-fun output0_rule1 () Int (ite rule1_applies 200 output0))
(define-fun output1_rule2 () Int (ite rule1_applies 300 output0))

(define-fun violation_output0 () Bool (and rule1_applies (distinct output0_rule1
output1_rule2)))

(assert violation_output0)

(check-sat)
(get-value (m))
```

sat ((m 101))

Example 2: Same actions for same range - No conflict

1: If 100<m<120 then output = 200
2: If 100<m<120 then output = 200

```
(define-fun range ((x Int) (lower Int) (upper Int)) Bool (and (< lower x) (< x upper)))

(declare-fun m () Int)


(define-fun rule1_applies () Bool ( and (range m 100 120)))

(declare-fun output0 () Int)

(define-fun output0_rule1 () Int (ite rule1_applies 200 output0))
(define-fun output1_rule2 () Int (ite rule1_applies 200 output0))
```

```
(define-fun violation_output0 () Bool (and rule1_applies (distinct output0_rule1
output1_rule2)))
(assert violation_output0)

(check-sat)
(get-value (m))
```

Unsat

Example 3: Same actions for different ranges - No conflict

1: If 100<m<120 then output = 200
2: If 90<m<95 then output = 200

```
(define-fun range ((x Int) (lower Int) (upper Int)) Bool (and (< lower x) (< x upper)))

(declare-fun m () Int)


(define-fun rule1_applies () Bool ( and (range m 100 120)))
(define-fun rule2_applies () Bool ( and (range m 90 95)))

(declare-fun output0 () Int)

(define-fun output0_rule1 () Int (ite rule1_applies 200 output0))
(define-fun output1_rule2 () Int (ite rule1_applies 200 output0))


(define-fun violation_output0 () Bool (and rule1_applies (distinct output0_rule1
output1_rule2)))
(assert violation_output0)

(check-sat)
(get-value (m))
```

Unsat

Example 4: Different actions for different ranges - No conflict

1: If 100<m<120 then output = 200
2: If 90<m<95 then output = 220

```
(define-fun range ((x Int) (lower Int) (upper Int)) Bool (and (< lower x) (< x upper)))

(declare-fun m () Int)


(define-fun rule1_applies () Bool ( and (range m 100 120)))
(define-fun rule2_applies () Bool ( and (range m 90 95)))

(declare-fun output0 () Int)

(define-fun output0_rule1 () Int (ite rule1_applies 200 output0))
(define-fun output1_rule2 () Int (ite rule2_applies 220 output0))


(define-fun violation_output0 () Bool (and rule1_applies rule2_applies (distinct output0_rule1
output1_rule2)))
(assert violation_output0)

(check-sat)
(get-value (m))
```

unsat

<u>1 Variable - More than 1 action :</u>

Example 1 : Same range and same actions - No conflict

1: If 100<m<120 then output1 = 200 and output 2 = 500
2: If 100<m<120 then output1 = 200 and output2 = 500

```
(define-fun range ((x Int) (lower Int) (upper Int)) Bool (and (< lower x) (< x upper)))

(declare-fun m () Int)


(define-fun rule1_applies () Bool ( and (range m 100 120)))
(define-fun rule2_applies () Bool ( and (range m 100 120)))

(declare-fun output0 () Int)
```

```
(declare-fun output1 () Int)

(define-fun output0_rule1 () Int (ite rule1_applies 200 output0))
(define-fun output0_rule2 () Int (ite rule1_applies 500 output1))
(define-fun output1_rule1 () Int (ite rule2_applies 200 output0))
(define-fun output1_rule2 () Int (ite rule2_applies 500 output1))


(define-fun violation_output0 () Bool (and rule1_applies rule2_applies (distinct output0_rule1
output1_rule2 output1_rule1 output1_rule2)))
(assert violation_output0)

(check-sat)
(get-value (m))
```

Unsat

Example 2: Same range and 1 different action - conflict

1: If 100<m<120 then output1 = 200 and output 2 = 500
2: If 100<m<120 then output1 = 200 and output2 = 600

```
(define-fun range ((x Int) (lower Int) (upper Int)) Bool (and (< lower x) (< x upper)))

(declare-fun m () Int)


(define-fun rule1_applies () Bool ( and (range m 100 120)))
(define-fun rule2_applies () Bool ( and (range m 100 120)))

(declare-fun output0 () Int)
(declare-fun output1 () Int)

(define-fun output0_rule1 () Int (ite rule1_applies 200 output0))
(define-fun output0_rule2 () Int (ite rule1_applies 500 output1))
(define-fun output1_rule1 () Int (ite rule2_applies 200 output0))
(define-fun output1_rule2 () Int (ite rule2_applies 600 output1))


(define-fun violation_output0 () Bool (and rule1_applies rule2_applies (distinct output0_rule1
output0_rule2)))
(define-fun violation_output1 () Bool (and rule1_applies rule2_applies (distinct output1_rule1
output1_rule2)))
(define-fun violation () Bool (or violation_output0 violation_output1))

(assert violation)
```

```
(check-sat)
(get-value (m))
```

sat ((m 101))

Example 3 : Same actions for different ranges - No conflict

1: If 100<m<120 then output1 = 200 and output 2 = 500
2: If 90<m<95 then output1 = 200 and output2 = 500

```
(define-fun range ((x Int) (lower Int) (upper Int)) Bool (and (< lower x) (< x upper)))

(declare-fun m () Int)


(define-fun rule1_applies () Bool ( and (range m 100 120)))
(define-fun rule2_applies () Bool ( and (range m 90 95)))

(declare-fun output0 () Int)
(declare-fun output1 () Int)

(define-fun output0_rule1 () Int (ite rule1_applies 200 output0))
(define-fun output0_rule2 () Int (ite rule1_applies 500 output1))
(define-fun output1_rule1 () Int (ite rule2_applies 200 output0))
(define-fun output1_rule2 () Int (ite rule2_applies 500 output1))


(define-fun violation_output0 () Bool (and rule1_applies rule2_applies (distinct output0_rule1
output0_rule2)))
(define-fun violation_output1 () Bool (and rule1_applies rule2_applies (distinct output1_rule1
output1_rule2)))
(define-fun violation () Bool (or violation_output0 violation_output1))

(assert violation)

(check-sat)
(get-value (m))
```

unsat

Example 4: Both Different actions for different ranges - No conflict

1: If 100<m<120 then output1 = 200 and output 2 = 300

2: If 90<m<95 then output1 = 400 and output2 = 500

```
(define-fun range ((x Int) (lower Int) (upper Int)) Bool (and (< lower x) (< x upper)))

(declare-fun m () Int)


(define-fun rule1_applies () Bool ( and (range m 100 120)))
(define-fun rule2_applies () Bool ( and (range m 90 95)))

(declare-fun output0 () Int)
(declare-fun output1 () Int)

(define-fun output0_rule1 () Int (ite rule1_applies 200 output0))
(define-fun output0_rule2 () Int (ite rule1_applies 500 output1))
(define-fun output1_rule1 () Int (ite rule2_applies 400 output0))
(define-fun output1_rule2 () Int (ite rule2_applies 600 output1))


(define-fun violation_output0 () Bool (and rule1_applies rule2_applies (distinct output0_rule1
output0_rule2)))
(define-fun violation_output1 () Bool (and rule1_applies rule2_applies (distinct output1_rule1
output1_rule2)))
(define-fun violation () Bool (or violation_output0 violation_output1))

(assert violation)

(check-sat)
(get-value (m))
```

Unsat

<u>2 variables with 2 actions - which are different</u>

1: If 100<m<120 then output1 = 200 and output 2 = 300
2: If 90<n<95 then output1 = 400 and output2 = 500

```
(define-fun range ((x Int) (lower Int) (upper Int)) Bool (and (< lower x) (< x upper)))

(declare-fun m () Int)
(declare-fun n () Int)
```

```
(define-fun rule1_applies () Bool ( and (range m 100 120)))
(define-fun rule2_applies () Bool ( and (range n 90 95)))

(declare-fun output0 () Int)
(declare-fun output1 () Int)

(define-fun output0_rule1 () Int (ite rule1_applies 200 output0))
(define-fun output0_rule2 () Int (ite rule1_applies 500 output1))
(define-fun output1_rule1 () Int (ite rule2_applies 400 output0))
(define-fun output1_rule2 () Int (ite rule2_applies 600 output1))


(define-fun violation () Bool (and rule1_applies rule2_applies (distinct output0_rule1
output0_rule2 output1_rule1 output1_rule2)))

(assert violation)

(check-sat)
(get-value (m n))
```

sat
M 101
N 91

<u>2 variables with 2 actions - which are same</u>

1: If 100<m<120 then output1 = 200 and output 2 = 300
2: If 90<n<95 then output1 = 200 and output2 = 300

```
(define-fun range ((x Int) (lower Int) (upper Int)) Bool (and (< lower x) (< x upper)))

(declare-fun m () Int)
(declare-fun n () Int)


(define-fun rule1_applies () Bool ( and (range m 100 120)))
(define-fun rule2_applies () Bool ( and (range n 90 95)))

(declare-fun output0 () Int)
(declare-fun output1 () Int)

(define-fun output0_rule1 () Int (ite rule1_applies 200 output0))
(define-fun output0_rule2 () Int (ite rule1_applies 300 output1))
(define-fun output1_rule1 () Int (ite rule2_applies 200 output0))
(define-fun output1_rule2 () Int (ite rule2_applies 300 output1))
```

```
(define-fun violation () Bool (and rule1_applies rule2_applies (distinct output0_rule1
output0_rule2 output1_rule1 output1_rule2)))

(assert violation)

(check-sat)
(get-value (m n))

unsat
```

2 Variables connected and have 1 action - which is different

1: If (100 < m < 120) and (200 < n < 220) then output = 200
2: If (110 < m < 120) and (200 < n <210) then output =220

```
(define-fun range ((x Int) (lower Int) (upper Int)) Bool (and (< lower x) (< x upper)))

(declare-fun m () Int)
(declare-fun n () Int)


(define-fun rule1_applies () Bool ( and (range m 100 120) (range n 200 220)))
(define-fun rule2_applies () Bool ( and (range m 110 120) (range n 200 210 )))


(declare-fun output0 () Int)

(define-fun output0_rule1 () Int (ite rule1_applies 200 output0))
(define-fun output0_rule2 () Int (ite rule2_applies 500 output0))


(define-fun violation () Bool (and rule1_applies rule2_applies (distinct output0_rule1
output0_rule2 )))

(assert violation)

(check-sat)
(get-value (m n))

sat ((m 111) (n 201))
```

2 Variables unconnected and have 1 action - which is different

1: If (100 < m < 120) and (200 < n < 220) then output = 200
2: If (110 < m < 120) and (300 < n < 400) then output =220

```
(define-fun range ((x Int) (lower Int) (upper Int)) Bool (and (< lower x) (< x upper)))

(declare-fun m () Int)
(declare-fun n () Int)


(define-fun rule1_applies () Bool ( and (range m 100 120) (range n 200 220)))
(define-fun rule2_applies () Bool ( and (range m 110 120) (range n 300 400 )))


(declare-fun output0 () Int)

(define-fun output0_rule1 () Int (ite rule1_applies 200 output0))
(define-fun output0_rule2 () Int (ite rule2_applies 500 output0))


(define-fun violation () Bool (and rule1_applies rule2_applies (distinct output0_rule1
output0_rule2 )))

(assert violation)

(check-sat)
(get-value (m n))
```

Unsat

2 Variables with 2 actions having overlapping regions and different actions

1: If (100 < m < 120) and (200 < n < 220) then output1= 200 and output2 = 300
2: If (110 < m < 120) and (200 < n < 400) then output1 =200 and output2 = 400

```
(define-fun range ((x Int) (lower Int) (upper Int)) Bool (and (< lower x) (< x upper)))

(declare-fun m () Int)
(declare-fun n () Int)


(define-fun rule1_applies () Bool ( and (range m 100 120) (range n 200 220)))
(define-fun rule2_applies () Bool ( and (range m 110 120) (range n 200 400 )))


(declare-fun output0 () Int)
(declare-fun output1 () Int)
```

```
(define-fun output0_rule1 () Int (ite rule1_applies 200 output0))
(define-fun output0_rule2 () Int (ite rule1_applies 300 output1))
(define-fun output1_rule1 () Int (ite rule2_applies 200 output0))
(define-fun output1_rule2 () Int (ite rule2_applies 400 output1))


(define-fun violation_output0 () Bool (and rule1_applies rule2_applies (distinct output0_rule1
output0_rule2)))
(define-fun violation_output1 () Bool (and rule1_applies rule2_applies (distinct output1_rule1
output1_rule2)))
(define-fun violation () Bool (or violation_output0 violation_output1))


(assert violation)

(check-sat)
(get-value (m n))
```

sat ((m 111) (n 201))

2 Variables with 2 actions having overlapping regions and same actions - <u>ask</u>

1: If (100 < m < 120) and (200 < n < 220) then output1= 200 and output2 = 300
2: If (110 < m < 120) and (200 < n < 400) then output1 =200 and output2 = 300

```
(define-fun range ((x Int) (lower Int) (upper Int)) Bool (and (< lower x) (< x upper)))

(declare-fun m () Int)
(declare-fun n () Int)


(define-fun rule1_applies () Bool ( and (range m 100 120) (range n 200 220)))
(define-fun rule2_applies () Bool ( and (range m 110 120) (range n 200 400 )))


(declare-fun output0 () Int)
(declare-fun output1 () Int)


(define-fun output0_rule1 () Int (ite rule1_applies 200 output0))
(define-fun output0_rule2 () Int (ite rule1_applies 300 output1))
(define-fun output1_rule1 () Int (ite rule2_applies 200 output0))
(define-fun output1_rule2 () Int (ite rule2_applies 300 output1))
```

```
(define-fun violation_output0 () Bool (and rule1_applies rule2_applies (distinct output0_rule1
output0_rule2)))
(define-fun violation_output1 () Bool (and rule1_applies rule2_applies (distinct output1_rule1
output1_rule2)))
(define-fun violation () Bool (or violation_output0 violation_output1))


(assert violation)

(check-sat)
(get-value (m n))

sat ((m 111) (n 201)
)
```

2 Variables with 2 actions having no overlapping regions and same actions

1: If (100 < m < 120) and (200 < n < 220) then output1= 200 and output2 = 300
2: If (110 < m < 120) and (300 < n < 400) then output1 =200 and output2 = 300

```
(define-fun range ((x Int) (lower Int) (upper Int)) Bool (and (< lower x) (< x upper)))

(declare-fun m () Int)
(declare-fun n () Int)


(define-fun rule1_applies () Bool ( and (range m 100 120) (range n 200 220)))
(define-fun rule2_applies () Bool ( and (range m 110 120) (range n 300 400 )))


(declare-fun output0 () Int)
(declare-fun output1 () Int)


(define-fun output0_rule1 () Int (ite rule1_applies 200 output0))
(define-fun output0_rule2 () Int (ite rule1_applies 300 output1))
(define-fun output1_rule1 () Int (ite rule2_applies 200 output0))
(define-fun output1_rule2 () Int (ite rule2_applies 300 output1))


(define-fun violation_output0 () Bool (and rule1_applies rule2_applies  (distinct output0_rule1
output0_rule2)))
(define-fun violation_output1 () Bool (and rule1_applies rule2_applies (distinct output1_rule1
output1_rule2)))
(define-fun violation () Bool (or violation_output0 violation_output1))
```

(assert violation)

(check-sat)
(get-value (m n))

Unsat

Complex Rules -

No overlapping region and different actions -
1 overlapping region and different actions
Same overlapping regions and different actions
Same overlapping regions and same actions

Case 1 - All parameters overlapping - and actions are different

1: If (((100 < m < 120) or (200 < n < 220)) and (((110<m<130) or(210<n<230))) then output1 = 200 and output 2 = 300
2: If (((100 < m < 140) or (200 < n < 220)) and (((110<m<130) or(210<n<230))) then output1= 300 and output 2 =300 .

```
(declare-fun m () Int)
(declare-fun n () Int)

(define-fun range ((x Int) (lower Int) (upper Int)) Bool (and (< lower x) (< x upper)))

(define-fun rule1_applies () Bool (and (or (range m 100 120) (range n 200 220))
                                  (or (range m 110 130) (range n 210 230))))

(define-fun rule2_applies () Bool (and (or (range m 100 140) (range n 200 220))
                                  (or (range m 110 130) (range n 210 230))))


(declare-fun output0 () Int)
(declare-fun output1 () Int)

(define-fun output0_rule1 () Int (ite rule1_applies 200 output0))
(define-fun output1_rule1 () Int (ite rule1_applies 300 output1))
(define-fun output0_rule2 () Int (ite rule2_applies 300 output0))
(define-fun output1_rule2 () Int (ite rule2_applies 300 output1))


(define-fun violation_output0 () Bool (and rule1_applies rule2_applies (distinct
output0_rule1 output0_rule2)))
```

```
(define-fun violation_output1 () Bool (and rule1_applies rule2_applies (distinct
output1_rule1 output1_rule2)))
(define-fun violation () Bool (or violation_output0 violation_output1))

(assert violation)

(check-sat)
(get-value (m n))

sat ((m 111) (n 211))
```

Case 2 - All parameters overlapping - and actions are same -

1: If (((100 < m < 120) or (200 < n < 220)) and (((110<m<130) or(210<n<230))) then output1 = 200 and output 2 = 300
2: If (((100 < m < 140) or (200 < n < 220)) and (((110<m<130) or(210<n<230))) then output1= 300 and output 2 =300 .

```
(declare-fun m () Int)
(declare-fun n () Int)

(define-fun range ((x Int) (lower Int) (upper Int)) Bool (and (< lower x) (< x upper)))

(define-fun rule1_applies () Bool (and (or (range m 100 120) (range n 200 220))
                                  (or (range m 110 130) (range n 210 230))))

(define-fun rule2_applies () Bool (and (or (range m 100 140) (range n 200 220))
                                  (or (range m 110 130) (range n 210 230))))


(declare-fun output0 () Int)
(declare-fun output1 () Int)

(define-fun output0_rule1 () Int (ite rule1_applies 200 output0))
(define-fun output1_rule1 () Int (ite rule1_applies 300 output1))
(define-fun output0_rule2 () Int (ite rule2_applies 200 output0))
(define-fun output1_rule2 () Int (ite rule2_applies 300 output1))


(define-fun violation_output0 () Bool (and rule1_applies rule2_applies (distinct output0_rule1
output0_rule2)))
(define-fun violation_output1 () Bool (and rule1_applies rule2_applies (distinct output1_rule1
output1_rule2)))
(define-fun violation () Bool (or violation_output0 violation_output1))

(assert violation)
```

```
(check-sat)
(get-value (m n))
```

```
unsat
```

Case 3 - only 1 of the parameters overlapping (say parameter n) and actions are different

1: If (((100 < m < 120) or (200 < n < 220)) and (((400<m<500) or(210<n<230))) then output1 = 200 and output 2 = 300
2: If (((100 < m < 140) or (200 < n < 220)) and (((110<m<130) or(210<n<230))) then output1= 400 and output 2 =500 .

```
(declare-fun m () Int)
(declare-fun n () Int)

(define-fun range ((x Int) (lower Int) (upper Int)) Bool (and (< lower x) (< x upper)))

(define-fun rule1_applies () Bool (and (or (range m 100 200) (range n 200 220))
                              (or (range m 400 500) (range n 210 230))))

(define-fun rule2_applies () Bool (and (or (range m 100 140) (range n 200 220))
                              (or (range m 110 130) (range n 210 230))))


(declare-fun output0 () Int)
(declare-fun output1 () Int)

(define-fun output0_rule1 () Int (ite rule1_applies 200 output0))
(define-fun output1_rule1 () Int (ite rule1_applies 300 output1))
(define-fun output0_rule2 () Int (ite rule2_applies 400 output0))
(define-fun output1_rule2 () Int (ite rule2_applies 500 output1))


(define-fun violation_output0 () Bool (and rule1_applies rule2_applies (distinct output0_rule1
output0_rule2)))
(define-fun violation_output1 () Bool (and rule1_applies rule2_applies (distinct output1_rule1
output1_rule2)))
(define-fun violation () Bool (or violation_output0 violation_output1))

(assert violation)

(check-sat)
(get-value (m n))
```

sat ((m 111) (n 211))

Case 4 : 1 of the parameters overlapping and actions are same

1: If (((100 < m < 120) or (200 < n < 220)) and (((400<m<500) or(210<n<230))) then output1 = 200 and output 2 = 300
2: If (((100 < m < 140) or (200 < n < 220)) and (((110<m<130) or(210<n<230))) then output1= 200 and output 2 =300 .

```
(declare-fun m () Int)
(declare-fun n () Int)

(define-fun range ((x Int) (lower Int) (upper Int)) Bool (and (< lower x) (< x upper)))

(define-fun rule1_applies () Bool (and (or (range m 100 200) (range n 200 220))
                              (or (range m 400 500) (range n 210 230))))

(define-fun rule2_applies () Bool (and (or (range m 100 140) (range n 200 220))
                              (or (range m 110 130) (range n 210 230))))


(declare-fun output0 () Int)
(declare-fun output1 () Int)

(define-fun output0_rule1 () Int (ite rule1_applies 200 output0))
(define-fun output1_rule1 () Int (ite rule1_applies 300 output1))
(define-fun output0_rule2 () Int (ite rule2_applies 200 output0))
(define-fun output1_rule2 () Int (ite rule2_applies 300 output1))


(define-fun violation_output0 () Bool (and rule1_applies rule2_applies (distinct output0_rule1
output0_rule2)))
(define-fun violation_output1 () Bool (and rule1_applies rule2_applies (distinct output1_rule1
output1_rule2)))
(define-fun violation () Bool (or violation_output0 violation_output1))

(assert violation)

(check-sat)
(get-value (m n))
```

unsat

Case 5 : No overlap and different actions -

```
1:        If (((100 < m < 200) or (100 < n < 200)) and (((300<m<400)  or(300<n<400)))
          then output1 = 200 and output 2 = 300
 2:      If (((400 < m < 500) or (400 < n < 500)) and (((500<m<600)  or(500<n<600)))
          then output1= 400 and output 2 =500
```

```
(declare-fun m () Int)
(declare-fun n () Int)

(define-fun range ((x Int) (lower Int) (upper Int)) Bool (and (< lower x) (< x upper)))

(define-fun rule1_applies () Bool (and (or (range m 100 200) (range n 100 200))
                           (or (range m 300 400) (range n 300 400))))

(define-fun rule2_applies () Bool (and (or (range m 400 500) (range n 400 500))
                           (or (range m 500 600) (range n 500 600))))


(declare-fun output0 () Int)
(declare-fun output1 () Int)

(define-fun output0_rule1 () Int (ite rule1_applies 200 output0))
(define-fun output1_rule1 () Int (ite rule1_applies 300 output1))
(define-fun output0_rule2 () Int (ite rule2_applies 400 output0))
(define-fun output1_rule2 () Int (ite rule2_applies 500 output1))


(define-fun violation_output0 () Bool (and rule1_applies rule2_applies (distinct
output0_rule1 output0_rule2)))
(define-fun violation_output1 () Bool (and rule1_applies rule2_applies (distinct
output1_rule1 output1_rule2)))
(define-fun violation () Bool (or violation_output0 violation_output1))

(assert violation)

(check-sat)
(get-value (m n))
```

```
unsat
```

1 overlapping region and different actions

```
1:        If (((100 < m < 200) or (100 < n < 200)) and (((300<m<400)  or(300<n<400)))
          then output1 = 200 and output 2 = 300
 2:       If (((100 < m < 200) or (400 < n < 500)) and (((300<m<400)  or(500<n<600)))
          then output1= 400 and output 2 =500
```

```
(declare-fun m () Int)
(declare-fun n () Int)

(define-fun range ((x Int) (lower Int) (upper Int)) Bool (and (< lower x) (< x upper)))

(define-fun rule1_applies () Bool (and (or (range m 100 200) (range n 100 200))
                            (or (range m 300 400) (range n 300 400))))

(define-fun rule2_applies () Bool (and (or (range m 100 200) (range n 400 500))
                            (or (range m 300 400) (range n 500 600))))


(declare-fun output0 () Int)
(declare-fun output1 () Int)

(define-fun output0_rule1 () Int (ite rule1_applies 200 output0))
(define-fun output1_rule1 () Int (ite rule1_applies 300 output1))
(define-fun output0_rule2 () Int (ite rule2_applies 400 output0))
(define-fun output1_rule2 () Int (ite rule2_applies 500 output1))


(define-fun violation_output0 () Bool (and rule1_applies rule2_applies (distinct
output0_rule1 output0_rule2)))
(define-fun violation_output1 () Bool (and rule1_applies rule2_applies (distinct
output1_rule1 output1_rule2)))
(define-fun violation () Bool (or violation_output0 violation_output1))

(assert violation)

(check-sat)
(get-value (m n))
```

```
sat
```

2 Variables with 1 conflicting action

1: If 100<m<120 then output1 = 200
2: If 90<n<95 then output1 = 400

Code-

```
(define-fun range ((x Int) (lower Int) (upper Int)) Bool (and (< lower x) (< x upper)))

(declare-fun m () Int)
(declare-fun n () Int)

(define-fun rule1_applies () Bool ( and (range m 100 120)))
(define-fun rule2_applies () Bool ( and (range n 90 95)))

(declare-fun output0 () Int)

(define-fun output0_rule1 () Int (ite rule1_applies 200 output0))
(define-fun output1_rule1 () Int (ite rule2_applies 400 output0))


(define-fun violation_output0 () Bool (and rule1_applies  (distinct output0_rule1)))
(define-fun violation_output1 () Bool (and rule2_applies (distinct output1_rule1 )))
(define-fun violation () Bool (or violation_output0 violation_output1))

(assert violation)

(check-sat)
(get-value (m n))
```

sat ((m 101) (n 91))

2 variables with different actions - no conflict - ask

1: If 100<m<120 then output1 = 200
2: If 90<n<95 then output2 = 400

```
(define-fun range ((x Int) (lower Int) (upper Int)) Bool (and (< lower x) (< x upper)))
```

```
(declare-fun m () Int)
(declare-fun n () Int)

(define-fun rule1_applies () Bool ( and (range m 100 120)))
(define-fun rule2_applies () Bool ( and (range n 90 95)))

(declare-fun output0 () Int)
(declare-fun output1 () Int)

(define-fun output0_rule1 () Int (ite rule1_applies 200 output0))
(define-fun output1_rule1 () Int (ite rule2_applies 400 output1))


(define-fun violation_output0 () Bool (and rule1_applies false))
(define-fun violation_output1 () Bool (and rule2_applies false))
(define-fun violation () Bool (or violation_output0 violation_output1))

(assert violation)

(check-sat)
(get-value (m n))
```

Unsat

1: If (((100 < m < 200) or (100 < n < 200)) and (((100<m<150))) then output1 = 200 and output 2 = 300
2: If (((100 < m < 500) or (100 < n < 500)) and (((100<m<120)) then output1= 200 and output 2 =300 .

```
(declare-fun m () Int)
(declare-fun n () Int)

(define-fun range ((x Int) (lower Int) (upper Int)) Bool (and (< lower x) (< x upper)))

(define-fun rule1_applies () Bool (and (or (range m 100 200) (range n 100 200))
                                  (range m 100 150) ))

(define-fun rule2_applies () Bool (and (or (range m 100 500) (range n 100 500))
                          (or (range m 100 120))))


(declare-fun output0 () Int)
(declare-fun output1 () Int)

(define-fun output0_rule1 () Int (ite rule1_applies 200 output0))
```

```
(define-fun output1_rule1 () Int (ite rule1_applies 300 output1))
(define-fun output0_rule2 () Int (ite rule2_applies 300 output0))
(define-fun output1_rule2 () Int (ite rule2_applies 300 output1))


(define-fun violation_output0 () Bool (and rule1_applies rule2_applies (distinct
output0_rule1 output0_rule2)))
(define-fun violation_output1 () Bool (and rule1_applies rule2_applies (distinct
output1_rule1 output1_rule2)))
(define-fun violation () Bool (or violation_output0 violation_output1))

(assert violation)

(check-sat)
(get-value (m n))
```