These link is vital to understand the smt-lib format -

1. **https://stackoverflow.com/questions/50822934/using-the-pure-smt-lib2-in-z3-to-check-for-consistency-in-rules**
2. **https://rise4fun.com/z3/tutorial**

Example 1 -

1. 10<abc<20 then outp = 100
2. 10<abc<20 then outp = 100

Code - https://rise4fun.com/Z3/fsMX

```
;Helper function declaration
(define-fun range ((x Int) (lower Int) (upper Int)) Bool (and (< lower x) (< x upper)))

;Declaration of input and output variables
(declare-fun abc () Int)
(declare-fun outp () Int)

;Declaration of range for input variables
(define-fun rule1_applies () Bool (and (range abc 10 20)))
(define-fun rule2_applies () Bool (and (range abc 10 20)))

;Declaration of rules for output variables
(define-fun output0_rule1 () Int (ite rule1_applies 100 outp))

(define-fun output0_rule2 () Int (ite rule2_applies 100 outp))


;Define a helper function
(define-fun atleast_two_rules_fire () Bool ((_ at-least 2) rule1_applies rule2_applies ))

;Define the violation for the output variables
(define-fun violation_output0 () Bool (and atleast_two_rules_fire  (not (= output0_rule1
output0_rule2  ))))

;Define the final violation constraint
(define-fun violation () Bool (or violation_output0 ))

(assert violation)

(check-sat)
```

Example 2-

1. 10<abc<20 then outp = 100
2. 10<abc<20 then outp = 100
3. 10<abc<20 then outp = 200

Code - https://rise4fun.com/Z3/9zLaJ

```
;Helper function declaration
(define-fun range ((x Int) (lower Int) (upper Int)) Bool (and (< lower x) (< x upper)))

;Declaration of input and output variables
(declare-fun abc () Int)
(declare-fun outp () Int)

;Declaration of range for input variables
(define-fun rule1_applies () Bool (and (range abc 10 20)))
(define-fun rule2_applies () Bool (and (range abc 10 20)))
(define-fun rule3_applies () Bool (and (range abc 10 20)))

;Declaration of rules for output variables
(define-fun output0_rule1 () Int (ite rule1_applies 100 outp))

(define-fun output0_rule2 () Int (ite rule2_applies 100 outp))

(define-fun output0_rule3 () Int (ite rule3_applies 200 outp))


;Define a helper function
(define-fun atleast_two_rules_fire () Bool ((_ at-least 2) rule1_applies rule2_applies
rule3_applies ))

;Define the violation for the output variables
(define-fun violation_output0 () Bool (and atleast_two_rules_fire  (not (= output0_rule1
output0_rule2  output0_rule3  ))))

;Define the final violation constraint
(define-fun violation () Bool (or violation_output0 ))
```

```
(assert violation)

(check-sat)
```