以mutualEx(3)为例，

不变式是：inv：n[i] = C -> n[j] != C

共四条规则，且每条规则都是只含一个变量，那么我们就以crit为例：

ruleset i : NODE

do rule "Crit"

n[i] = T & x = true

==>

begin

n[i] := C;

x := false;

endrule;endruleset;

可以有以下3种组合方式(所以共有12组f)：

1. Inv(1,2)+crit(1)

F1 = crit(1).guard & crit(1).action & assumption\_on\_2 & !inv

= n[1] = T & x=true &

n[1]’ = C & x’ = false &

n[2] = n[2]’ &

!(n[1] = C -> n[2] != C)

Inv(1,2)+crit(2)

F2 = crit(2).guard & crit(2).action & assumption\_on\_1 & !inv

= n[2] = T & x=true &

n[2]’ = C & x’ = false &

n[1] = n[1]’ &

!(n[1] = C -> n[2] != C)

Inv(1,2)+crit(3)

F2 = crit(3).guard & crit(3).action & assumption\_on\_1&2 & !inv

= n[3] = T & x=true &

n[3]’ = C & x’ = false &

n[1] = n[1]’ &

n[2] = n[2]’ &

!(n[1] = C -> n[2] != C)

第一步：先取到所有的f的式子（实例化后的）

怎么取？

——取到的先用Python语义表达吗？（Python->SMT-LIB,好处：Python已经有了大部分的节点，可以直接按节点来转）。或是直接用字符串表达式来表示？（好处：比较人眼可观；bad：转SMT-LIB会复杂得多吧，还得一个一个去匹配）

怎么给实例化方案？

——这周先直接给出好了。先遍历invPara的var，发现只需要两个参数i j，那么根据对称规约，我们给它实例化为1，2就行了；然后针对具体规则，比如Crit，发现Crit[var]只有一个参数i，那么我们需要分别实例化i为1，2，3。TIPS：还需要保证前面的i j，类型和后面的i是一致的。

怎么把带参的式子变为实例化的？

——直接正则项匹配上了就替换（n[i]->n[1],n[j]->n[2]，这样的话等协议变复杂了之后很容易出问题），这周先用这个吧；或者看看根据Python的节点去匹配到变量i，然后给替换成1。

怎么添加assumption项？

——对比规则的instances和不变式的，如果发现某个变量出现在了不变式中，但没有出现在规则中，那么此时就需要添加对该变量的assumption

✔构造f：按PPT上需要的项，合取起来就行了

✔在SMT-LIB2中怎么表示这个f？

怎么将Python节点转换为SMT-LIB2的表示？

——这一步估计得像转ivy那样写转换规则

第二步：把取到的式子写到SMT-LIB2的形式

第三步：调z3 solver，如果sat则表示有反例，获取反例的具体值

第四部：处理反例式子，得到新的不变式

假设req参数个数为m，r参数个数为k，则m与k可能有以下几种不同情况(注意使用对称规约)：

1. m=2,k=1。则r中实例化需要paras={[1],[2],[3]}，1/2分别对应req中的参数，3代表与req参数不重叠的情况；例如，将不变的 mutualInv(i1, i2) 实例化为 i1 = 1, i2 = 2 并将规则crit( j) 实例化为 j = 1、2 或 3 。

2. m=2,k=2。则r中实例化需要paras={[1,2],[2,1],[1,3],[2,3],[3,1],[3,2],[3,4]}，其中[3,4]代表与req参数完全不重叠情况。

{'mutualEx\_Try\_1':

{'guard': OpExpr(=, n[1], I),

'assign': [AssignCmd(ArrayIndex(VarExpr('n'), VarExpr('1')), EnumValExpr(EnumType('I', 'T', 'C', 'E'), 'T'))],

'assumption': [ArrayIndex(VarExpr('n'), VarExpr('2'))],

'!inv': NegExpr(1 != 2 -> ( n[1] = C -> n[2] != C))},

'mutualEx\_Try\_2':

{'guard': OpExpr(=, n[2], I),

'assign': [AssignCmd(ArrayIndex(VarExpr('n'), VarExpr('2')), EnumValExpr(EnumType('I', 'T', 'C', 'E'), 'T'))],

'assumption': [ArrayIndex(VarExpr('n'), VarExpr('1'))],

'!inv': NegExpr(1 != 2 -> ( n[1] = C -> n[2] != C))},

'mutualEx\_Crit\_1':

{'guard': OpExpr(&, n[1] = T, x = true),

'assign': [AssignCmd(ArrayIndex(VarExpr('n'), VarExpr('1')), EnumValExpr(EnumType('I', 'T', 'C', 'E'), 'C')), AssignCmd(VarExpr('x'), BooleanExpr(False))],

'assumption': [ArrayIndex(VarExpr('n'), VarExpr('2'))],

'!inv': NegExpr(1 != 2 -> ( n[1] = C -> n[2] != C))},

'mutualEx\_Crit\_2':

{'guard': OpExpr(&, n[2] = T, x = true),

'assign': [AssignCmd(ArrayIndex(VarExpr('n'), VarExpr('2')), EnumValExpr(EnumType('I', 'T', 'C', 'E'), 'C')), AssignCmd(VarExpr('x'), BooleanExpr(False))],

'assumption': [ArrayIndex(VarExpr('n'), VarExpr('1'))],

'!inv': NegExpr(1 != 2 -> ( n[1] = C -> n[2] != C))},

'mutualEx\_Exit\_1':

{'guard': OpExpr(=, n[1], C),

'assign': [AssignCmd(ArrayIndex(VarExpr('n'), VarExpr('1')), EnumValExpr(EnumType('I', 'T', 'C', 'E'), 'E'))],

'assumption': [ArrayIndex(VarExpr('n'), VarExpr('2'))],

'!inv': NegExpr(1 != 2 -> ( n[1] = C -> n[2] != C))},

'mutualEx\_Exit\_2':

{'guard': OpExpr(=, n[2], C),

'assign': [AssignCmd(ArrayIndex(VarExpr('n'), VarExpr('2')), EnumValExpr(EnumType('I', 'T', 'C', 'E'), 'E'))],

'assumption': [ArrayIndex(VarExpr('n'), VarExpr('1'))],

'!inv': NegExpr(1 != 2 -> ( n[1] = C -> n[2] != C))},

'mutualEx\_Idle\_1':

{'guard': OpExpr(=, n[1], E),

'assign': [AssignCmd(ArrayIndex(VarExpr('n'), VarExpr('1')), EnumValExpr(EnumType('I', 'T', 'C', 'E'), 'I')), AssignCmd(VarExpr('x'), BooleanExpr(True))],

'assumption': [ArrayIndex(VarExpr('n'), VarExpr('2'))],

'!inv': NegExpr(1 != 2 -> ( n[1] = C -> n[2] != C))},

'mutualEx\_Idle\_2': {'guard': OpExpr(=, n[2], E),

'assign': [AssignCmd(ArrayIndex(VarExpr('n'), VarExpr('2')), EnumValExpr(EnumType('I', 'T', 'C', 'E'), 'I')), AssignCmd(VarExpr('x'), BooleanExpr(True))],

'assumption': [ArrayIndex(VarExpr('n'), VarExpr('1'))],

'!inv': NegExpr(1 != 2 -> ( n[1] = C -> n[2] != C))}}