

作业1
N皇后
SMT
SAT
对比
作业2
加法
减法
代码使用
实验结果

## 作业1

### N皇后

#### SMT

以下代码内容来自老师PPT，基本流程就是先约束每个Queen坐标的列坐标取值范围 `val_c`，再通过 `col_c` 约束每个列上不会有两个Queen，最后通过 `diag_c` 约束对角线。

```
1 def n_queen_smt(n = 8):
2     Q = [Int('Q_%i' % (i + 1)) for i in range(n)]
3     val_c = [And(1 <= Q[i], Q[i] <= n) for i in range(n)]
4     col_c = [Distinct(Q)]
5     diag_c = [If(i == j, True, And(i+Q[j] != j+Q[i], i-Q[j] != j-Q[i])) for i
6               in range(n) for j in range(i)]
7
8     time_start=time.perf_counter()
9     solve(val_c + col_c + diag_c)
10    time_end=time.perf_counter()
11    print('time cost for smt:',time_end-time_start,'s')
```

#### SAT

以下代码基本流程

1. 对N queen问题中， $n * n$ 棋盘的每个格子定义一个变量 $Q_{n,n}$
2. 先约束每行有且仅有一个格子为1，例如一行有 $a_1, a_2, a_3$ 三个格子通过 $(\neg(\neg a_1 \vee a_2 \vee a_3) \vee \neg(a_1 \vee \neg a_2 \vee a_3) \vee \neg(a_1 \vee a_2 \vee \neg a_3))$ 来保证约束成立
3. 再通过与上述方法相同的思路约束列 `col_c`
4. 最后类似思路通过 `diag_c` 约束对角线没有冲突，但是由于可能一条对角线上没有皇后，所以需要增加一个 $\neg(\vee Q_{ij})$ 约束，其中 $i, j$ 对应对角线元素。

```
1 def n_queen_sat(n = 8):
2     Q = [Bool('Q_%i%i' % (i + 1, j + 1)) for i in range(n) for j in
3           range(n)]
4     row_c = And([Or([Not(Or([If(i==k, Not(Q[j*n+i]), Q[j*n+i]) for i in
5                             range(n)))] for k in range(n)))] for j in range(n)) #每一行有且仅有一个为True
6     col_c = And([Or([Not(Or([If(j==k, Not(Q[j*n+i]), Q[j*n+i]) for j in
7                             range(n)))] for k in range(n)))] for i in range(n)) #每一列有且仅有一个为True
8     outer1 = []
```

```

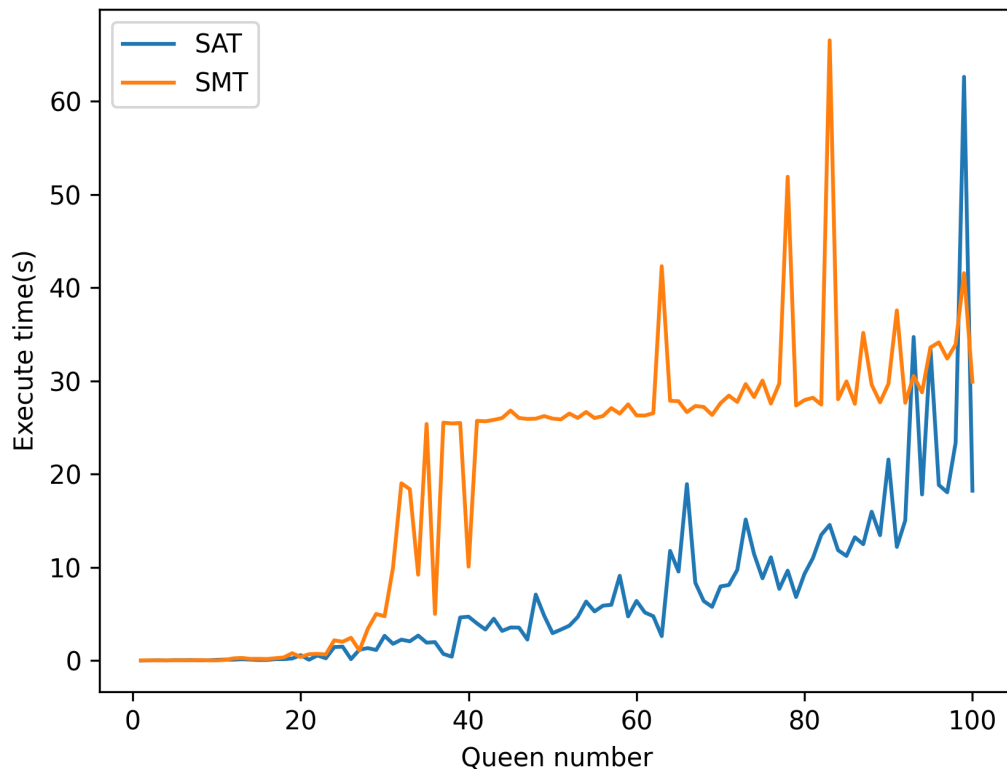
6     outer2 = []
7     for j in range(2*n-1):#0-6
8         inner = []
9         literal = []
10        for i in range(n):
11            if(j-i < n and j-i >=0):
12                literal += [Q[(j-i)*n+i]]
13        outer_list = [Not(Or(literal))]
14        for a in range(len(literal)):
15            inner_list = []
16            for b in range(len(literal)):
17                if(b == a):
18                    inner_list += [Not(literal[b])]
19                else:
20                    inner_list += [literal[b]]
21            outer_list += [Not(Or(inner_list))]
22        outer1 += [Or(outer_list)]
23    for j in range(-n+1,n):#0-3
24        inner = []
25        literal = []
26        for i in range(n):
27            if(j+i < n and j+i >= 0):
28                literal += [Q[(j+i)*n+i]]
29        outer_list = [Not(Or(literal))]
30        # print(literal)
31        for a in range(len(literal)):
32            inner_list = []
33            for b in range(len(literal)):
34                if(b == a):
35                    inner_list += [Not(literal[b])]
36                else:
37                    inner_list += [literal[b]]
38            outer_list += [Not(Or(inner_list))]
39        outer2 += [Or(outer_list)]
40    diag_c = And(And(outer1), And(outer2))
41
42    time_start=time.perf_counter()
43    solve(row_c + col_c + diag_c)
44    time_end=time.perf_counter()
45    print('time cost for smt:',time_end-time_start,'s')

```

## 对比

可以看到在大多数情况下，pure SAT的效率要高于SMT，但是有时候SAT求解会出现较大抖动。总体来说**SAT**求解的效率要高于**SMT**，这可能是由于SMT转换为SAT时需要花费一定时间，而且SMT转换为SAT后可能变量个数要多于pure SAT，所以效率低于pure SAT。

不过图中的时间仅仅包含 `solve` 函数的执行时间。如果将前面的数据预处理部分一起加入，pure SAT花费的时间要多于SMT，这大概是因为我自己写的pure SAT约束生成部分效率较低，用了很多 `for` 循环，所以花费时间比SMT中自带的转换器慢。



## 作业2

由于观察到减法  $a-b=d$  可以转化为  $a=b+d$ ，所以加法和减法实际上是一个东西，所以我就先实现了加法，后在加法基础上进行修改得到减法

### 加法

由于a, b的二进制位数可能不相同，所以需要按照最长位数补齐。

其它关于代码内容在注释中

```

1  def add(a = 20, b = 7):
2      max_len = max(len("{0:b}".format(a)), len("{0:b}".format(b)))+1
3      format_str = '{0:0'+str(max_len)+'b}'
4      a_bin = format_str.format(a)
5      b_bin = format_str.format(b)
6      # 数据预处理，将A B补足到相同位数
7      A = [Bool('a_%i' % (i + 1)) for i in range (max_len)]
8      B = [Bool('b_%i' % (i + 1)) for i in range (max_len)]
9      C = [Bool('c_%i' % i) for i in range (max_len + 1)]
10     D = [Bool('d_%i' % (i + 1)) for i in range (max_len)]
11     # 采用舍去D的最高位的方式计算
12     # 变量定义
13     A_c = And([If(a_bin[i] == '0', Not(A[i]), A[i]) for i in
range(max_len)]) # 将A的二进制数字形式化为表达式
14     B_c = And([If(b_bin[i] == '0', Not(B[i]), B[i]) for i in
range(max_len)]) # 将B的二进制数字形式化为表达式
15     D_c = And([D[i]==(A[i]==(B[i]==C[i+1])) for i in range(max_len)]) # 约
束A[i],B[i],C[i],D[i]间关系
16     Carry_c = And([C[i]==Or(And(A[i], B[i]), And(A[i], C[i+1]), And(B[i],
C[i+1])) for i in range(max_len)]) # 约束进位关系

```

```

17     s = Solver()
18     s.add(A_c, B_c, D_c, Carry_c, Not(C[0]), Not(C[max_len]))
19     result = s.check()
20     # 输出计算结果
21     if result == sat:
22         d = ""
23         for i in range(max_len):
24             if s.model()[D[i]] == True:
25                 d += '1'
26             else:
27                 d += '0'
28             print(int(d, 2))
29             return int(d, 2)
30     return None

```

## 减法

大部分代码思路和加法一致，只不过在  $A[i], B[i], C[i], D[i]$  间关系的约束和进位的约束需要稍作修改

```

1  def minus(a = 20, b = 7):
2      # a - b = d implies a = b + d
3      max_len = max(len("{0:b}".format(a)), len("{0:b}".format(b)))
4      format_str = '{0:0'+str(max_len)+'b}'
5      a_bin = format_str.format(a)
6      b_bin = format_str.format(b)
7      A = [Bool('a_%i' % (i + 1)) for i in range (max_len)]
8      B = [Bool('b_%i' % (i + 1)) for i in range (max_len)]
9      C = [Bool('c_%i' % i) for i in range (max_len + 1)]
10     D = [Bool('d_%i' % (i + 1)) for i in range (max_len)]
11     A_c = And([If(a_bin[i] == '0', Not(A[i]), A[i]) for i in
range(max_len)])
12     B_c = And([If(b_bin[i] == '0', Not(B[i]), B[i]) for i in
range(max_len)])
13     # 以上内容和加法相同
14     # 以下内容唯一不同在于由于  $d = a - b \Leftrightarrow a = b + d$ ，所以需要把加法中的a换成b，d换成a
15     D_c = And([A[i] == (D[i] == (B[i] == C[i+1])) for i in range(max_len)])
16     Carry_c = And([C[i] == Or(And(D[i], B[i]), And(D[i], C[i+1]), And(B[i],
C[i+1])) for i in range(max_len)])
17     s = Solver()
18     s.add(A_c, B_c, D_c, Carry_c, Not(C[0]), Not(C[max_len]))
19     result = s.check()
20     if result == sat:
21         d = ""
22         for i in range(max_len):
23             if s.model()[D[i]] == True:
24                 d += '1'
25             else:
26                 d += '0'
27             print(int(d, 2))
28             return int(d, 2)
29     return None

```

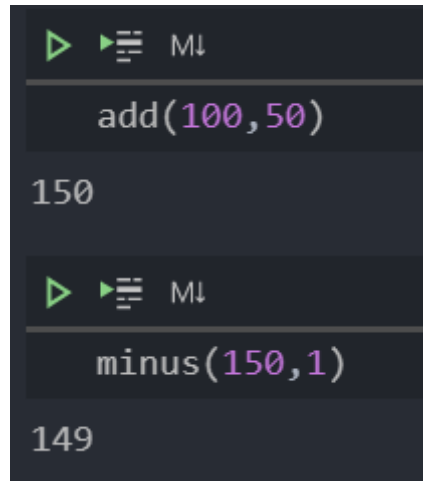
## 代码使用

`add(a, b)` 和 `minus(a, b)` 均为两个函数，输入值为 `a` 和 `b`，在加法中两者顺序无所谓，在减法中 `minus(a,b)` 表示计算 `a-b` 的结果并打印到显示器上，返回值为计算得到的10进制结果，如果计算失败则返回None。

直接调用 `add()` 或 `minus()`，其中 `a,b` 默认为20和7，作为示例和测试用途。

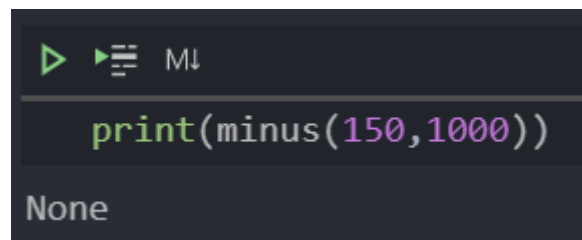
## 实验结果

最终加法和减法都能够正常运行，运行结果如下



```
▶ ▶≡ M↓  
add(100,50)  
150  
▶ ▶≡ M↓  
minus(150,1)  
149
```

如果减法出现负数，则会运行失败，返回None



```
▶ ▶≡ M↓  
print(minus(150,1000))  
None
```