# 实验小作业-作业3

梁峻滔 PB19051175

## 1. 分别使用 SMT 和 pure SAT 求解 N-Queen 问题

- SMT

  由于每行都要求恰有一个皇后，因此可以使用每行皇后的列位置来表示一个布局，添加变量取值约束 `val_c` 表示。同时要求一列也恰有一个皇后，因此添加列约束`col_c`。最后是要求对角线方向上也最多只有一个皇后，添加对角线约束`diag_c`。最后使用 Z3 求解。

- SAT

  采用一个8 x 8二维数组 p 来表示布局, 当 p[i][j] == True 时表示第 i 行第 j 列位置上放置一个皇后。在此基础上添加行约束 `row_c1` 和 `row_c2`，列约束 `col_c1` 和 `col_c2`，以及对角线约束 `diag_c`，然后使用 Z3 求解。

使用 `timeit.default_timer` 来对两种方法的求解过程计时(不包括约束建立过程)，比对 n 从8增加到40时，两者求解所费时间(单位为秒)：

| n | SMT Time | SAT Time |
|---|---|---|
| 8 | 0.04619 | 0.00614 |
| 9 | 0.01489 | 0.00442 |
| 10 | 0.01087 | 0.00521 |
| 11 | 0.02291 | 0.00541 |
| 12 | 0.01933 | 0.00599 |
| ... | | |
| 22 | 0.43354 | 0.02070 |
| 23 | 0.12790 | 0.02295 |
| 24 | 1.54494 | 0.02779 |
| 25 | 0.91701 | 0.03081 |
| 26 | 0.32301 | 0.03397 |
| ... | | |
| 36 | 11.59893 | 0.08651 |
| 37 | 3.15988 | 0.09133 |
| 38 | 9.48723 | 0.10826 |
| 39 | 14.81084 | 0.13409 |
| 40 | 17.12309 | 0.31495 |

从比对结果看来，随着 n 的增大，整体规律是两者的运行时间差别越来越显著，pure SAT的求解速度显著快于 SMT的求解速度。以下附上源码，使用python运行即可，如需查看求解结果，则需修改源码中的`print_result`字段为`True`。

```
from z3 import *
from timeit import default_timer as timer

def Solve_SMT(n, print_result):
    '''
    使用SMT求解n皇后问题，print_result设置为True时输出求解结果，否则不输出结果
    '''
    Q = [Int('Q%i' % (i+1)) for i in range(n)]
    val_c = [And(Q[i] >= 1, Q[i] <= n) for i in range(n)]
    col_c = [Distinct(Q)]
    diag_c = [If(i == j, True, And(i+Q[i] != j+Q[j], i+Q[j] != j+Q[i]))
              for i in range(n) for j in range(i)]
    s = Solver()
    s.add(val_c)
    s.add(col_c)
    s.add(diag_c)
    tic = timer()
    s.check()
    toc = timer()
    print('SMT Solver for %d queens Time(sec): '%n, end='')
    print(toc - tic)
    if print_result:
        print(s.model())


def Solve_SAT(n, print_result):
    '''
    使用SAT求解n皇后问题，print_result设置为True时输出求解结果，否则不输出结果
    '''
    p = [[] for i in range(n)]
    for i in range(n):
        for j in range(n):
            p[i].append(Bool('p_%i%i' % (i+1, j+1)))

    row_c1 = []
    for i in range(n):
        bool_vector = []
        for j in range(n):
            bool_vector.append(p[i][j])
        row_c1.append(Or(bool_vector))

    row_c2 = []
    for i in range(n):
        for k in range(2, n):
            for j in range(k):
                row_c2.append(Or(Not(p[i][j]), Not(p[i][k])))

    col_c1 = []
    for j in range(n):
        bool_vector = []
        for i in range(n):
            bool_vector.append(p[i][j])
        col_c1.append(Or(bool_vector))
```

```
    col_c2 = []
    for j in range(n):
        for k in range(2, n):
            for i in range(k):
                col_c2.append(Or(Not(p[i][j]), Not(p[k][j])))

    diag_c = []
    for i in range(n-1):
        for i1 in range(i+1, n):
            for j in range(n):
                for j1 in range(n):
                    if i+j == i1+j1 or i-j == i1-j1:
                        diag_c.append(Or(Not(p[i][j]), Not(p[i1][j1])))

    s = Solver()
    s.add(row_c1)
    s.add(row_c2)
    s.add(col_c1)
    s.add(col_c2)
    s.add(diag_c)
    tic = timer()
    s.check()
    toc = timer()
    print('SAT Solver for %d queens Time(sec): '%n, end='')
    print(toc - tic)
    if print_result:
        for i in range(n):
            for j in range(n):
                print(s.model().eval(p[i][j]), end=' ')
            print('')


# main
for n in range(8, 40):
    Solve_SMT(n, False)
    Solve_SAT(n, False)
    print('')
```

## 2. 使用 pure SAT 求解 d = a - b(a, b为正整数且a > b)

先把输入的 a 和 b 转换成二进制数，使用0-1数组表示，将它们零扩展成相同的长度，并在最高位前再补一个0，由于要实现减 b 计算，所以再将 b 的二进制表示转换成补码表示，再使用将两者相加的 SAT 方法。添加以下约束：

(1) 对$i = 1, \ldots, n$，要求$d_i \leftrightarrow (a_i \leftrightarrow (b_i \leftrightarrow c_i))$

(2) 对$i = 1, \ldots, n$，要求$c_{i-1} \leftrightarrow ((a_i \wedge b_i) \vee (a_i \wedge c_i) \vee (b_i \wedge c_i))$

(3) 从最右边开始计算，最右边的进位应为0，所以有$\neg c_n$

(4) 需要注意的是，加法中最高位的进位要求为0，即要求有$\neg c_0$，但是在补码计算中可能会有溢出，不能有这个约束

(5) 对于a的二进制表示的每一位$a_i$，如果$a_i = 1$，就将 $a_i$ 加入到合取式中，否则将 $\neg a_i$ 加入到合取式中

(6) 对于b的二进制补码表示的每一位$b_i$，如果$b_i = 1$，就将 $b_i$ 加入到合取式中，否则将 $\neg b_i$ 加入到合取式中

(7) 将(1)~(3)的合取式作为$\phi$，再与(5)(6)做一个合取，得到要求解的 SAT 问题，最后使用 Z3 求解该问题，将求解结果中的 $d_i(i = 1, \ldots, n)$ 再转换成十进制数表示然后输出

以下附上源码，求解过程由函数`subtraction(a, b)`实现，参数 a, b 即为被减数和减数，设置参数使用python运行后就会输出减法计算结果。

```
from z3 import *
from timeit import default_timer as timer

def subtraction(a, b):
    '''
    输入被减数a和减数b，使用SAT方法计算a-b后输出计算结果
    '''
    if a < b:
        print("a < b, try again")
        return

    Abit = []
    while True:
        bit = a % 2
        Abit.append(bit)
        a = a // 2
        if a == 0:
            break
    Abit.append(0)
    Abit.reverse()

    Bbit = []
    while True:
        bit = b % 2
        Bbit.append(bit)
        b = b // 2
        if b == 0:
            break
    Bbit.append(0)
    Bbit.reverse()

    if len(Abit) > len(Bbit):
        Bbit.reverse()
        while len(Bbit) < len(Abit):
            Bbit.append(0)
        Bbit.reverse()
    elif len(Bbit) > len(Abit):
        Abit.reverse()
        while len(Abit) < len(Bbit):
            Abit.append(0)
        Abit.reverse()

    print(Abit)
    print(Bbit)
    # 取B的补码
    for i in range(len(Bbit)):
        Bbit[i] = 1 - Bbit[i]
    c = 1
```

```
    Bbit.reverse()
    for i in range(len(Abit)):
        temp = Bbit[i]
        Bbit[i] = (Bbit[i] + c) % 2
        c = (temp + c) // 2
    Bbit.reverse()


    print(Bbit)

    n = len(Abit)
    A = [Bool('a%i'%i) for i in range(n+1)]
    B = [Bool('b%i'%i) for i in range(n+1)]
    C = [Bool('c%i'%i) for i in range(n+1)]
    D = [Bool('d%i'%i) for i in range(n+1)]

    cons1 = True
    for i in range(1, n+1):
        cons1 = And(cons1, D[i] == (A[i] == (B[i] == C[i])))

    cons2 = True
    for i in range(1, n+1):
        cons2 = And(cons2, C[i-1] == (Or(And(A[i], B[i]), And(A[i], C[i]), And(B[i],
C[i]))))

    cons3 = Not(C[n])
    # cons4 = Not(C[0])

    cons5 = True
    for i in range(1, n+1):
        if Abit[i-1] == 1:
            cons5 = And(cons5, A[i])
        else:
            cons5 = And(cons5, Not(A[i]))

    cons6 = True
    for i in range(1, n+1):
        if Bbit[i-1] == 1:
            cons6 = And(cons6, B[i])
        else:
            cons6 = And(cons6, Not(B[i]))

    s = Solver()
    # cons = [cons1, cons2, cons3, cons4, cons5, cons6]
    cons = [cons1, cons2, cons3, cons5, cons6]
    s.add(cons)

    tic = timer()
    s.check()
    toc = timer()
    print(toc - tic)
    result_bit = []
    for i in range(1, n+1):
        if s.model().eval(D[i]) == True:
            result_bit.append(1)
        else:
            result_bit.append(0)
    exp = 1
```

```
    result = 0
    for i in range(n-1, -1, -1):
        result += exp * result_bit[i]
        exp *= 2
    print(result)

subtraction(186, 23)
```