

实验文档

PB20061338 柯志伟

N-Queen问题

1. 问题如下:

```
1 The eight queens puzzle is the problem of placing eight chess queens on
2 an 8 × 8 chessboard so that no two queens attack each other. Thus, a
3 solution requires that no two queens share the same row, column, or
4 diagonal.
```

2. 使用SAT描述问题中的约束如下:

```
1 (1) At least one queen on row i, i in [1,n]
2 (2) At most one queen on row i, i in [1,n]
3 (3) At least one queen on column i, i in [1,n]
4 (4) At most one queen on column i, i in [1,n]
5 (5) on (i, j) and (i', j') being two distinct positions on a diagonal, no
    two queens are allowed
```

3. 使用C++调用Z3的API编写pure-SAT代码如下:

```
1 void solve_nqueen_puresat(int n) {
2     auto start = std::chrono::high_resolution_clock::now();
3     cout << "The num of queen is " << n << endl;
4     context ctx;
5     solver slv(ctx);
6     vector<vector<expr>>> queens;
7
8
9     for (int i = 0; i < n; i++) {
10         vector<expr> temp;
11         for (int j = 0; j < n; j++) {
12             temp.push_back(ctx.bool_const(("queen_" + to_string(i) + "_" +
13 to_string(j)).c_str()));
14         }
15         queens.push_back(temp);
16     }
17     vector<expr> rows, cols;
18     for (int i = 0; i < n; i++) {
19         for (int j = 0; j < n; j++) {
20             // 增加行列约束: 至少一个为真
21             if (j == 0) rows.push_back(queens[i][0]);
22             else rows[i] = rows[i] || queens[i][j];
```

```

22         if (i == 0) cols.push_back(queens[0][j]);
23         else cols[j] = cols[j] || queens[i][j];
24         if (i == n - 1) slv.add(cols[j]);
25     }
26     slv.add(rows[i]);
27 }
28 // 增加行列约束：至多一个为真
29 expr temp_expr_1(ctx);
30 for (int i = 0; i < n; i++) {
31     for (int j = 1; j < n; j++) {
32         for (int k = 0; k < j; k++) {
33             if (i == 0 && j == 1 && k == 0) temp_expr_1 = !queens[0]
[1] || !queens[0][0];
34             else temp_expr_1 = (temp_expr_1) && (!queens[i][j] ||
!queens[i][k]);
35         }
36     }
37 }
38 slv.add(temp_expr_1);
39
40 for (int j = 0; j < n; j++) {
41     for (int i = 1; i < n; i++) {
42         for (int k = 0; k < i; k++) {
43             if (j == 0 && i == 1 && k == 0) temp_expr_1 = !queens[1]
[0] || !queens[0][0];
44             else temp_expr_1 = (temp_expr_1) && (!queens[i][j] ||
!queens[k][j]);
45         }
46     }
47 }
48 slv.add(temp_expr_1);
49 // 增加对角约束
50 expr temp_expr_2(ctx);
51
52 for (int i = 1; i < n; i++) {
53     for (int j = 0; j < n; j++) {
54         bool init = false;
55         for (int k = i - 1; k >= 0 && (j + k - i) >= 0; k--) {
56             if (k == i - 1) temp_expr_1 = !queens[k][j+k-i] ||
!queens[i][j];
57             else temp_expr_1 = (temp_expr_1) && (!queens[k][j + k - i]
|| !queens[i][j]);
58             init = true;
59         }
60         for (int k = i + 1; k < n && (i + j - k) >= 0; k++) {
61             if (!init) {
62                 temp_expr_1 = (!queens[k][i + j - k] || !queens[i]
[j]);

```

```

63         }
64         else {
65             temp_expr_1 = (temp_expr_1) && (!queens[k][i + j - k]
|| !queens[i][j]);
66         }
67     }
68     if (i == 1 && j == 0) temp_expr_2 = temp_expr_1;
69     else temp_expr_2 = (temp_expr_2) && (temp_expr_1);
70 }
71 }
72
73
74 slv.add(temp_expr_2);
75 auto res = slv.check();
76 cout << res << endl;
77 switch (res) {
78     case unsat:
79         cout << "The problem is unsat" << endl;
80         break;
81     case sat: {
82         model m = slv.get_model();
83         for (int i = 0; i < n; i++) {
84             for (int j = 0; j < n; j++) {
85                 cout << m.eval(ctx.bool_const(("queen_" + to_string(i)
+ "_" + to_string(j)).c_str())) << "\t" ;
86             }
87             cout << endl;
88         }
89     }
90     break;
91     case unknown:
92         cout << "The problem is unknown" << endl;
93         break;
94 }
95 auto end = std::chrono::high_resolution_clock::now();
96 std::chrono::duration<double> elapsed = end - start;
97 std::cout << "Elapsed time: " << elapsed.count() << " s\n";
98 }

```

4. 按照上课时SMT建立约束的方法在C++中实现SMT求解N皇后问题

```

1 void solve_nqueen_smt(int n) {
2     auto start = std::chrono::high_resolution_clock::now();
3     cout << "The num of queen is " << n << endl;
4     context ctx;
5     solver slv(ctx);
6     vector<expr> queen_cols;
7     for (int i = 0; i < n; i++) {

```

```

8      queen_cols.push_back(ctx.int_const(("queen_" +
to_string(i)).c_str()));
9      }
10
11     expr_vector temp_expr_vec(ctx);
12     for (int i = 0; i < n; i++) {
13         slv.add((queen_cols[i] >= 1 && queen_cols[i] <= n));
14         temp_expr_vec.push_back(queen_cols[i]);
15     }
16     slv.add(distinct(temp_expr_vec));
17     for (int i = 0; i < n; i++) {
18         for (int j = 0; j < i; j++) {
19             slv.add((queen_cols[j] != (queen_cols[i] + i-j)) &&
(queen_cols[j] != (queen_cols[i] + j-i)));
20         }
21     }
22     auto res = slv.check();
23     cout << res << endl;
24     switch (res) {
25     case unsat:
26         cout << "The problem is unsat" << endl;
27         break;
28     case sat: {
29         model m = slv.get_model();
30         for (int i = 0; i < n; i++) {
31             cout << m.eval(ctx.int_const(("queen_" +
to_string(i)).c_str())) << "\t";
32         }
33         cout << endl;
34     }
35         break;
36     case unknown:
37         cout << "The problem is unknown" << endl;
38         break;
39     }
40     auto end = std::chrono::high_resolution_clock::now();
41     std::chrono::duration<double> elapsed = end - start;
42     std::cout << "Elapsed time: " << elapsed.count() << " s\n";
43 }

```

5. 在不同问题规模下SAT, SMT运行时间如下

问题规模	SAT耗时(s)	SMT耗时(s)
8	0.196	0.11
16	2.39	0.39

问题规模	SAT耗时(s)	SMT耗时(s)
24	2.79	3.72
32	25.59	5.27
40	40.19	8.48

通过对比可以发现使用pure-SAT解决N皇后问题的效率要高于SMT

[备注说明] C++使用的Z3库为从网上下载的源码，经过CMake编译为Visual Studio下使用的库并链接到通过Visual Studio创建的z3-solver项目来使用

两数减法问题

1. 问题

1 | 使用 pure SAT 求解 $d=a-b$ ，其中 a,b 为正整数。

2. 为问题建立SAT的约束

```

1 | 首先使用二进制表示整数(包括输入和结果),先考虑两数的加法,由于使用二进制表示整数,对于二进制加法有使用如下符号 $a_i, b_i, c_i, d_i$ ,分别代表两个输入数字的第 $i$ 位输入以及两位二进制数相加的进位和结果,在SAT中约束如下(设两个输入数字 $a,b$ 的使用二进制表示的最大位数为 $max$ ):
2 |     (1)  $d_i \leftrightarrow (a_i \leftrightarrow (b_i \leftrightarrow c_i))$ ,  $i \in [0, max]$ 
3 |     (2)  $c_{i-1} \leftrightarrow ((a_i \wedge b_i) \vee (a_i \wedge c_i) \vee (b_i \wedge c_i))$ ,  $i \in [0, max]$ 
4 |     (3)  $\neg c_n$ ,  $n = max$ 
5 |     (4)  $\neg c_0$ 
6 |     (5) 对于 $a_i, b_i$ 的约束,  $i \in [0, max]$ 
7 | 默认求解的结果使用二进制表示后位数为 $max$ (即可能会发生溢出)
8 | 由于对于二进制减法来说,减去一个数相当于加上该数字的补码,故计算减法时只需将输入的 $b$ 改为其对应的补码,剩余约束均对新的 $b$ 进行即可,这样便构成两个数相减的SAT约束

```

3. 使用C++调用Z3的API编写两数相减的SAT代码如下:

```

1 | void get_num_base2(int num, vector<int>& num_vec) {
2 |     if (num == 0) num_vec.push_back(0);
3 |     else {
4 |         while (num != 0) {
5 |             num_vec.push_back(num % 2);
6 |             num = num / 2;
7 |         }
8 |     }
9 |     return;
10 | }

```

```

11
12 void solve_sub(int a, int b) {
13     context ctx;
14     solver slv(ctx);
15
16     vector<int> a_vec, b_vec;
17     get_num_base2(a, a_vec);
18     get_num_base2(b, b_vec);
19
20
21     int max_digit_num = a_vec.size() > b_vec.size() ? a_vec.size() :
b_vec.size();
22
23     vector<expr> a_bools, b_bools, c_bools, d_bools;
24     for (int i = 0; i < max_digit_num; i++) {
25         a_bools.push_back(ctx.bool_const(("a_" + to_string(i)).c_str()));
26         b_bools.push_back(ctx.bool_const(("b_" + to_string(i)).c_str()));
27         c_bools.push_back(ctx.bool_const(("c_" + to_string(i)).c_str()));
28         d_bools.push_back(ctx.bool_const(("d_" + to_string(i)).c_str()));
29     }
30     c_bools.push_back(ctx.bool_const(("c_" +
to_string(c_bools.size()).c_str()));
31
32     expr temp(ctx);
33     for (int i = 0; i < max_digit_num; i++) {
34         if (i == 0) {
35             temp = (a_vec[0] == 0) ? (!a_bools[0]) : (a_bools[0]);
36             temp = (b_vec[0] == 1) ? (temp && (!b_bools[0])) : (temp &&
(b_bools[0]));
37         }
38         else {
39             if (i > a_vec.size() - 1) {
40                 temp = (temp) && (!a_bools[i]);
41             }
42             else {
43                 if (a_vec[i] == 0) {
44                     temp = (temp) && (!a_bools[i]);
45                 }
46                 else {
47                     temp = (temp) && (a_bools[i]);
48                 }
49             }
50             if (i > b_vec.size() - 1) {
51                 temp = (temp) && (b_bools[i]);
52             }
53             else {
54                 if (b_vec[i] == 1) {
55                     temp = (temp) && (!b_bools[i]);

```

```

56         }
57         else {
58             temp = (temp) && (b_bools[i]);
59         }
60     }
61 }
62 slv.add(temp);
63 }
64 slv.add(c_bools[0]);
65
66 for (int i = 0; i < max_digit_num; i++) {
67     slv.add(d_bools[i] == (a_bools[i] == (b_bools[i] ==
c_bools[i]))));
68     if (i != max_digit_num - 1) {
69         slv.add(c_bools[i + 1] == ((a_bools[i] && b_bools[i]) ||
(a_bools[i] && c_bools[i]) || (b_bools[i] && c_bools[i]))));
70     }
71     else {
72         slv.add(!c_bools[i + 1]);
73     }
74 }
75 auto start = std::chrono::high_resolution_clock::now();
76 auto res = slv.check();
77 cout << res << endl;
78 switch (res) {
79 case unsat:
80     cout << "The problem is unsat" << endl;
81     break;
82 case sat: {
83     model m = slv.get_model();
84     cout << "a is: " << a << "\t";
85     for (int i = max_digit_num - 1; i >= 0; i--) {
86         cout << m.eval(ctx.bool_const(("a_" + to_string(i)).c_str()))
<< "\t";
87     }
88     cout << endl;
89     cout << "b is: " << b << "\t";
90     for (int i = max_digit_num - 1; i >= 0; i--) {
91         cout << m.eval(ctx.bool_const(("b_" + to_string(i)).c_str()))
<< "\t";
92     }
93     cout << endl;
94     cout << "sum is: " << "\t";
95     for (int i = max_digit_num - 1; i >= 0; i--) {
96         cout << m.eval(ctx.bool_const(("d_" + to_string(i)).c_str()))
<< "\t";
97     }
98     cout << endl;

```

```

99
100     }
101         break;
102     case unknown:
103         cout << "The problem is unknown" << endl;
104         break;
105     }
106     auto end = std::chrono::high_resolution_clock::now();
107     std::chrono::duration<double> elapsed = end - start;
108     std::cout << "Elapsed time: " << elapsed.count() << " s\n";
109 }

```

4. 实验结果及说明

The screenshot displays the Visual Studio IDE with a C++ source file and the Microsoft Visual Studio 调试控制台 (Debug Console) window. The code defines a function `solve_sub` and a `main` function that calls it with different parameters. The debug console shows the output of the program, including the elapsed time and the results of the SAT solver.

First Run (Line 367): `solve_sub(15, 14);`

Debug Console Output:

```

a is: 15      true   true   true   true
b is: 14      false  false  false  true
sum is:       false  false  false  true
Elapsed time: 0.038737 s

```

Second Run (Line 367): `solve_sub(15, 1);`

Debug Console Output:

```

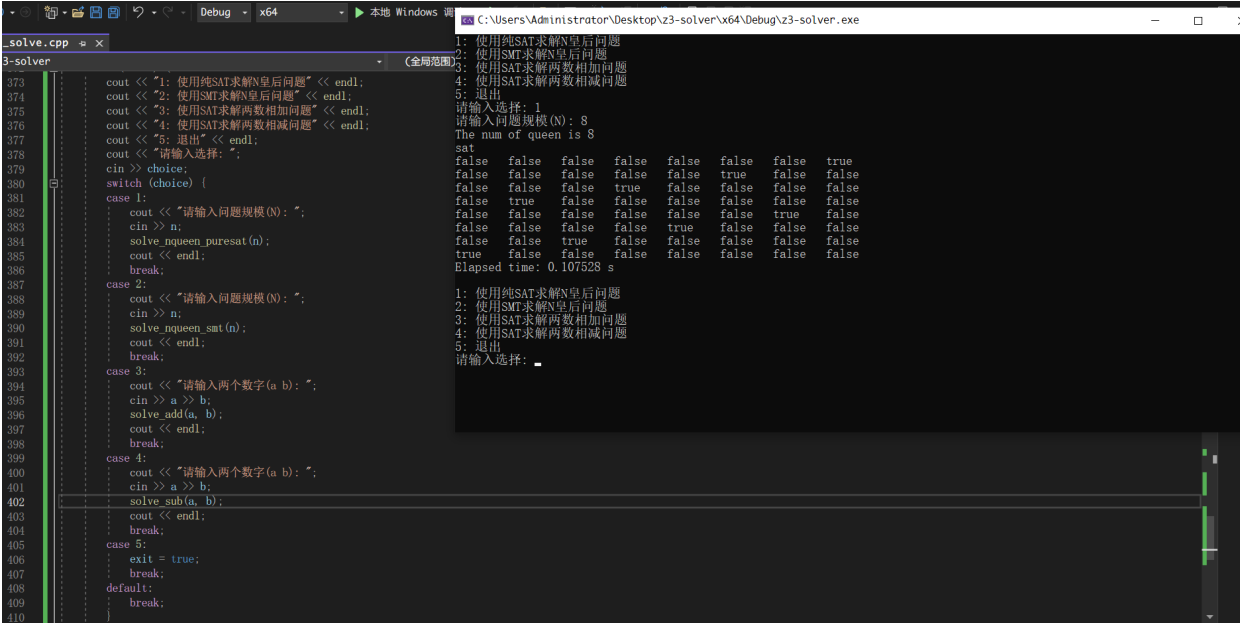
a is: 15      true   true   true   true
b is: 1 true   true   true   false
sum is:       true   true   true   false
Elapsed time: 0.0378077 s

```

由实验结果,最后模型解出的约束分别代表 **a** 的二进制表示, **b** 的反码的二进制表示,结果的二进制表示,并且结果正确

代码使用

为方便检验,简单封装使用界面如下:



编译好的可执行程序位于提交的z3-solver\x64\Debug\z3-solver.exe,其依赖bin目录下的z3动态链接库,使用Visual Studio运行