形式化方法导引第三次作业

PB20000156 徐亦昶

注：我在之前已经提交了一份报告，但加法的实现没有来得及写详细说明文档，同时没有做减法。在这份报告中，我写了减法程序，并对编程思路做了说明。

Problem1 N皇后问题

使用SAT求解器进行求解：

```
from z3 import *
import time
N=int(input('Please input N:'))
T1 = time.process_time()
p=[[Bool('Q_%s%s' % (i+1,j+1)) for j in range(N)] for i in range(N)]
existrow_c=[Or([p[i][j] for j in range(N)]) for i in range(N)]
row_c=[And([Or(Not(p[i][j]),Not(p[i][k])) for j in range(N-1) for k in
range(j+1,N)]) for i in range(N)]
existcol_c=[Or([p[i][j] for i in range(N)]) for j in range(N)]
col_c=[And([Or(Not(p[i][j]),Not(p[k][j])) for i in range(N-1) for k in
range(i+1,N)]) for j in range(N)]
diag_c=[And([If(Or(i+j==i1+j1,i-j==i1-j1),Or(Not(p[i][j]),Not(p[i1][j1])),True) for
j in range(N) for j1 in range(N)]) for i in range(N-1) for i1 in range(i+1,N)]
s=Solver()
s.add(existrow_c+existcol_c+row_c+diag_c+col_c)
if s.check()==sat:
    m=s.model()
    r=[[m.evaluate(p[i][j]) for j in range(N)] for i in range(N)]
    print_matrix(r)
T2 = time.process_time()
print('程序运行时间:%s秒' % (T2 - T1))
```

和SMT进行对比，取不同的N值：
N从5开始往后依次增加：

N	SMT求解速度	SAT求解速度
5	0.0.015625	0.0625
6	0.015625	0.078125
7	0.015625	0.203125
8	0.03125	0.296875
9	0.046875	0.484375
10	0.046875	0.734375
11	0.046875	1.140625
12	0.078125	1.59375
13	0.0625	2.140625
14	0.0625	2.953125
15	0.078125	3.734375
16	0.1875	5.078125
17	0.078125	6.5
18	0.1875	7.84375
19	0.28125	10.140625
20	0.1875	12.921875

可以看到，SMT的求解时间变化特别缓慢，而SAT在问题规模增大时耗时增加较快。可能的原因是对于N皇后问题，SMT中只需要N个变量，但SAT需要建立关于N^2个变量的方程式，而且约束关系更为复杂。

Problem2 a+b求解

假设两个加数分别为$a_1, a_2, \ldots, a_N$和$b_1, b_2, \ldots, b_N$，进位是$c_0, c_1, \ldots, c_N$，结果是$d_1, \ldots, d_N$则有如下关系式：

$$d_i \leftrightarrow (a_i \leftrightarrow (b_i \leftrightarrow c_i))$$

$$c_{i-1} \leftrightarrow ((a_i \wedge b_i) \vee (a_i \wedge c_i) \vee (b_i \wedge c_i))$$

由此建立条件即可求解。

```
from z3 import *
N=5
a=[Bool('a_%s' % (i+1)) for i in range(N)]
b=[Bool('b_%s' % (i+1)) for i in range(N)]
c=[Bool('c_%s' % i) for i in range(N+1)]
d=[Bool('d_%s' % (i+1)) for i in range(N)]
def Eq(u,v):
    return And(Implies(u,v),Implies(v,u))
s=Solver()
```

```
d_c=[Eq(d[i],Eq(a[i],Eq(b[i],c[i+1]))) for i in range(N)]
carry_c=[Eq(c[i-1],Or(And(a[i-1],b[i-1]),And(a[i-1],c[i]),And(b[i-1],c[i]))) for i
in range(1,N+1)]
c_c=[Not(c[N]),Not(c[0])]
input_c=
[And(Not(a[0]),a[1],a[2],Not(a[3]),a[4],Not(b[0]),Not(b[1]),b[2],b[3],b[4])] #13+7
s.add(d_c+carry_c+c_c+input_c)
if s.check()==sat:
    m=s.model()
    r=[m.evaluate(d[i]) for i in range(N)]
    print_matrix(r)
```

运行结果：[True, False, True, False, False]，与实际结果一致。

Problem 2+ 减法器

d的计算方法不变，需要改变的是原来的c由进位为改为借位位。
新的c计算规则是：

$$c_{i-1} \leftrightarrow ((\neg a_i \wedge b_i) \vee (\neg a_i \wedge c_i) \vee (a_i \wedge b_i \wedge c_i))$$

同时取消对c[0]的限制，允许结果为负，在代码最后添加对c[0]的检测来显示符号。
如果是符号，则交换a和b，即使用另外一种c的计算方法，该计算方法中a和b调换。

```
from z3 import *
N=5 #位数
a=[Bool('a_%s' % (i+1)) for i in range(N)]
b=[Bool('b_%s' % (i+1)) for i in range(N)]
c=[Bool('c_%s' % i) for i in range(N+1)]
d=[Bool('d_%s' % (i+1)) for i in range(N)]
def Eq(u,v):
    return And(Implies(u,v),Implies(v,u))
negative=0
s=Solver()
d_c=[Eq(d[i],Eq(a[i],Eq(b[i],c[i+1]))) for i in range(N)]
carry_c=[Eq(c[i-1],Or(And(Not(a[i-1]),b[i-1]),And(Not(a[i-1]),c[i]),And(a[i-1],b[i-
1],c[i]))) for i in range(1,N+1)]
carry2_c=[Eq(c[i-1],Or(And(Not(b[i-1]),a[i-1]),And(Not(b[i-1]),c[i]),And(b[i-
1],a[i-1],c[i]))) for i in range(1,N+1)]
c_c=[Not(c[N])]
input_c=
[And(Not(a[0]),a[1],a[2],Not(a[3]),a[4],Not(b[0]),Not(b[1]),b[2],b[3],b[4])] #a和b的
值在这里设定。
s.add(d_c+carry_c+c_c+input_c)
s.check()
m=s.model()
r=[m.evaluate(d[i]) for i in range(N)]
neg=m.evaluate(c[0])
if(neg==False):
    print_matrix(r)
else:
    negative=1
if(negative):
    s2=Solver()
    s2.add(d_c+carry2_c+c_c+input_c)
```

```
    s2.check()
    m=s2.model()
    r=[m.evaluate(d[i]) for i in range(N)]
    print('negative')
    print_matrix(r)
```

运行代码，输出是[False, False, True, True, False]，即13-7=6。将7的最高位Not去掉，则变成23。新的结果是negative [False, True, False, True, False]，这和13-12=-6吻合。